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

BACHELOR OF SCIENCE THESIS

GENERATING COVERINGS USING VIRTUAL SUBSTITUTION FOR EXPLANATIONS IN MCSAT

Max Harder

Examiners:
Prof. Dr. Erika Ábrahám
Prof. Dr. Jürgen Giesl

Additional Advisor:
Jasper Nalbach

Aachen, 23.08.2022

Abstract

Satisfiability modulo theories (SMT) refers to the problem of checking whether a quantifier-free formula over some theories is satisfiable. In recent times, the *model-constructing satisfiability calculus (mcSAT)* was presented as a new approach for SMT solving which makes use of an explanation function in order to describe conflicts that occur during the search for a solution. Since the choice of this function has a strong influence on the time needed to solve an input formula, this thesis presents a method to generate explanations by generalizing coverings using virtual substitution. In conclusion, the results of implementing this method into the *SMT-RAT* framework are compared and discussed.

Acknowledgements

I am very thankful to Prof. Erika Ábrahám for the opportunity to write my bachelor thesis about this very interesting topic. Many thanks to Jasper Nalbach for supervising me throughout the time. Furthermore, I would like to thank Prof. Jürgen Giesl for agreeing to be the second examiner.

Contents

Chapter 1

Introduction

The *Boolean satisfiability problem (SAT)* is the first problem that was proven to be NP-complete; therefore, solving instances of SAT requires an exponential time effort in the worst case. The increasing number of use-cases, however, led to a continuous improvement of SAT solvers. As a result, there are many approaches nowadays which allow for a reasonably fast solving of instances of SAT for problems like model checking or artificial intelligence. A well known algorithm is *DPLL*, which is extended to the most commonly used *conflict-driven clause learning (CDCL)* framework.

Having many applications, the Boolean satisfiability problem was extended to *satisfiability modulo theories (SMT)*, which allows the usage of first-order logic theories like *linear real arithmetic (LRA)* and *non-linear real arithmetic (NRA)*. Solvers for SMT can generally be separated into two classes. The first one, being *eager SMT solving*, describes the approach of transforming a logical formula over a theory into a satisfiability-equivalent propositional formula, which then can be solved using SAT solvers. The other method is called *lazy SMT solving* and makes use of a SAT solver as well as a theory solver. While the SAT solver focusses on the satisfaction of the Boolean skeleton of a given formula, the theory solver validates the consistency with the underlying theory and returns an explanation in case of conflicts. The *DPLL(T)* framework, which is an extension to *DPLL*, is such a lazy SMT solver that is quite successful.

In the recent time, a new approach called *model-constructing satisfiability calculus (mcSAT)* was developed as an extension to DPLL(T). The main difference is that mcSAT is not restricted to Boolean decisions only but allows the assignment of variables to concrete values, which then can be involved in the search for a solution and in explaining possible conflicts. Since the returned explanation from the theory solver has a significant influence on the performance of the algorithm, this thesis presents a method that generates explanations by generalizing coverings using virtual substitution.

Therefore, we start in Chapter 2 with a brief introduction into NRA and the principle of mcSAT, after which an overview of virtual substitution is given. In Chapter 3, the procedure for generating explanations using virtual substitution is explained and illustrated on a few examples. Having understood the theoretical idea, Chapter 4 deals with the implementation into the *SMT-RAT* framework and the comparison to other approaches. Finally, Chapter 5 concludes the results and mentions opportunities for improvement.

Chapter 2

Preliminaries

2.1 Non-linear real arithmetic

In the following, let $\mathbb{R}$ be the set of real numbers, let $\mathbb{Q}$ be the set of rational numbers and let $\mathbb{N}$ denote the set of natural numbers excluding 0. Furthermore, let $dom(f)$ be the domain of a function $f : X \to Y$, i.e., $dom(f) = X$.

Definition 2.1.1 (Non-linear real arithmetic)**.** *A formula φ from the theory of quantifier-free non-linear real arithmetic (QFNRA) is defined as follows:*

$$\begin{aligned} \varphi &:= c \mid (\varphi \wedge \varphi) \mid \neg\varphi \\ c &:= p < 0 \mid p = 0 \\ p &:= const \mid x \mid (p + p) \mid (p - p) \mid (p \cdot p) \end{aligned}$$

where x is an arbitrary variable that gets assigned a real value and $const \in \mathbb{Q}$. The terms c are called constraints*, while the terms p are called* polynomials*.*

Since the set $\{\neg, \wedge\}$ is functionally complete, every other operator like $\vee$ or $\to$ can be represented. Likewise, we allow the usage of the operators $\{\leq, \geq, >, \neq\}$ as syntactic sugar and write, e.g., $p \geq 0$ instead of $\neg(p < 0)$. Moreover, the operators $\{\leq, \geq, =\}$ are called *weak*, while $\{<, >, \neq\}$ are called *strict*.

Let $Vars(p)$ be the set of all variables that appear in a polynomial p. We call p *univariate* if $|Vars(p)| = 1$ and *multivariate* if $|Vars(p)| \geq 2$. Additionally, every polynomial p with $Vars(p) = \{x_1, x_2, \ldots, x_n\}$ can be transformed into a normal form that looks as follows:

$$p = \sum_{j=1}^{m} c_j \cdot x_1^{e_{1,j}} \cdot x_2^{e_{2,j}} \cdot \cdots \cdot x_n^{e_{n,j}}$$

where $c_j \in \mathbb{Q}$, $m \in \mathbb{N}$ and $e_{i,j} \in \mathbb{N} \cup \{0\}$ with $1 \leq i \leq n, 1 \leq j \leq m$. The *degree of a variable* x_i in p is defined in the following way:

$$deg_{x_i}(p) = max(\{e_{i,j} \mid 1 \leq j \leq m\}).$$

We call p linear in x_i if $deg_{x_i}(p) = 1$. Similarly, p is quadratic in x_i if $deg_{x_i}(p) = 2$ holds.

Definition 2.1.2 (Assignment)**.** *Let V be a set of variables. An* assignment *is a function $\alpha : V \to \mathbb{R}$ which assigns each variable in V a real value. Additionally, we call an assignment $\beta : V \rightharpoonup \mathbb{R}$* partial *if there exists an $x_i \in V$ such that $\beta(x_i)$ is undefined; in this case, we also write $\beta(x_i) \notin \mathbb{R}$.*

Definition 2.1.3 (Extension of an assignment)**.** *An* extension *of an assignment α is another assignment $\alpha' : V \to \mathbb{R}$ where $dom(\alpha) \subseteq dom(\alpha')$ and $\alpha'(x) = \alpha(x)$ for all $x \in dom(\alpha)$ with $\alpha(x) \in \mathbb{R}$.*

The extension of an assignment α which maps the variable y to the value $k \in \mathbb{R}$ is denoted as $\alpha_{y \mapsto k}$, i.e.,

$$\alpha_{y \mapsto k}(x_i) = \begin{cases} \alpha(x_i) & x_i \neq y \\ k & else \end{cases}.$$

Definition 2.1.4 (Evaluating a formula)**.** *Let α be an assignment. The* evaluation *of a formula φ is written as $[\![\varphi]\!]^\alpha$ and is done in the standard way. We say that α* satisfies *φ, written as $\alpha \models \varphi$, if $[\![\varphi]\!]^\alpha$ evaluates to true. A formula φ is called* satisfiable *if there exists an assignment that satisfies φ. Furthermore, φ is called* valid*, written as $\models \varphi$, if φ is satisfied by every assignment. The symbol $\equiv$ is used to denote logical equivalence of formulas.*

2.2 mcSAT

The *model-constructing satisfiability calculus (mcSAT)* [MJ13] is a recent development in SMT solving, which extends the functionality of DPLL(T). As in DPLL(T), mcSAT makes use of both a SAT solver and a theory solver in order to find a satisfying solution. The crucial difference is that mcSAT allows the theory solver to assign variables to concrete values that can then be used to find an assignment and to explain upcoming conflicts.

If such a conflict arises, the theory solver has to produce an explanation based on the set of constraints together with the currently chosen partial assignment but is also allowed to introduce new literals, in contrast to DPLL(T). These literals have to be from a finite set $\mathbb{B}$, which is called *finite basis*, so that the termination of the procedure is guaranteed. Formally, a conflict and an explanation are defined as follows:

Definition 2.2.1 (Conflict between a set of constraints and an assignment)**.** *Let R be a set of constraints, called* reason set*, that contain the variables $x_1, \ldots, x_n, y$ and let α be the current partial assignment that assigns every $x_i \in \{x_1, \ldots, x_n\}$ a value in $\mathbb{R}$. The set R and assignment α are* conflicting *if no extension α' of α with $\alpha'(y) \in \mathbb{R}$ and $\alpha' \models \bigwedge_{c \in R} c$ exists. In other words, this means that the variable y is not and cannot be assigned a value in $\mathbb{R}$ without causing a* conflict between R and α.

Definition 2.2.2 (Explanation)**.** *Let R be a set of constraints and α an assignment such that R and α are conflicting. An* explanation *for this conflict is a formula E that fulfils the following properties:*

- *E is valid, i.e., $\models E$ holds*
- *E has the form $\varphi \to \psi$ where $\varphi \equiv \bigwedge_{c \in R' \subseteq R} c$ and $\alpha \not\models \psi$*
- *every literal l in E is part of the finite basis $\mathbb{B}$*

2.3 Virtual substitution

Virtual substitution (VS) [LW93, Wei97] is a quantifier elimination procedure for non-linear real arithmetic formulas in which the variables are only allowed to have a degree of at most 4. In this thesis, however, we will only focus on variables with a degree not higher than 2 and limit our view on only some parts of VS.

2.3.1 Symbolic zeros

One part of virtual substitution consists in the calculation of zeros for univariate and multivariate polynomials. These zeros are expressed by *square root expressions* and are calculated as described in the following.

Definition 2.3.1 (Square root expression)**.** *A* square root expression *is an expression of the form*

$$\frac{p + q\sqrt{r}}{s}$$

where p,q,r,s are polynomials.

In the beginning, we will have a look at the univariate case. Therefore, let $p = ax^2 + bx + c$ be a polynomial with a single variable x and values $a,b,c \in \mathbb{Q}$. If we are interested in the real zeros, we can calculate them using the following rules:

Definition 2.3.2 (Zeros of a polynomial)**.** *Let $p = ax^2 + bx + c$ be a polynomial with coefficients a, b and c that do not contain x. The* zeros of *p can be calculated as follows:*

$$\begin{aligned} z_0 &= -\tfrac{c}{b} && \text{, if } a = 0 \text{ and } b \neq 0 \\ z_1 &= \tfrac{-b+\sqrt{b^2-4ac}}{2a} && \text{, if } a \neq 0 \text{ and } b^2 - 4ac \geq 0 \\ z_2 &= \tfrac{-b-\sqrt{b^2-4ac}}{2a} && \text{, if } a \neq 0 \text{ and } b^2 - 4ac > 0 \end{aligned}$$

As seen in table above, the existence of a zero depends on some constraints called *side conditions* that have to be fulfilled. The first zero, for example, does only exist if the given polynomial is linear in x. We abbreviate these side conditions using the function *sc* which maps a zero to its side conditions.

In the multivariate case, we can also apply the above-mentioned rules. The only difference is that the coefficients of x in the polynomial p, namely a,b and c, are now polynomials themselves. Using this approach, the resulting zeros will not be necessarily real values but rather new expressions which are called *symbolic zeros* from now on. Additionally, the symbolic zero z_i of a polynomial p can be written as $z_i(p)$ for $i \in \{0,1,2\}$.

Example 2.3.1. *Let $p = ax^2 + bx + c$ with $a = z$, $b = -6$ and $c = 1$ be a multivariate polynomial. Using the upper rules, we can calculate the following symbolic zeros:*

$$\begin{aligned} z_0 &= -\frac{1}{-6} = \frac{1}{6} \\ z_1 &= \frac{-(-6) + \sqrt{(-6)^2 - 4 \cdot z}}{2 \cdot z} = \frac{6 + \sqrt{36 - 4 \cdot z}}{2 \cdot z} \\ z_2 &= \frac{-(-6) - \sqrt{(-6)^2 - 4 \cdot z}}{2 \cdot z} = \frac{6 - \sqrt{36 - 4 \cdot z}}{2 \cdot z} \end{aligned}$$

In order to determine which of these zeros exist, we consider the assignment α *with* $\alpha(z) = 5$ *and check the side conditions of each zero. The side conditions of* z_0 *are*

$$sc(z_0) \equiv (a = 0 \wedge b \neq 0) \equiv (z = 0 \wedge -6 \neq 0) \equiv (z = 0).$$

Since α *assigns* z *to 5,* $\alpha \not\models sc(z_0)$ *holds and* z_0 *is not a zero of* p*. The side conditions of* z_1 *and* z_2 *are*

$$sc(z_1) \equiv (a \neq 0 \wedge b^2 - 4ac \geq 0) \equiv (z \neq 0 \wedge (-6)^2 - 4 \cdot z \geq 0),$$
$$sc(z_2) \equiv (z \neq 0 \wedge (-6)^2 - 4 \cdot z > 0).$$

As a result, $\alpha \models sc(z_1)$ *and* $\alpha \models sc(z_2)$ *holds. Therefore,* p *has the two zeros* z_1 *and* z_2 *if* α *is taken into account.*

2.3.2 Substituting variables virtually

Another part of the virtual substitution method is the substitution of variables. Therefore, let $-\infty$ be the number that is smaller and ∞ be the number that is larger than every other number. In contrast to substituting a variable x directly, the variable gets substituted virtually. The reason is that x could be replaced by $-\infty$, for example, such that the resulting expression would not necessarily be a real arithmetic formula as described in Definition 2.1.1.

For this thesis, it is sufficient to restrict to the substitution rules for $-\infty$ and for square root expressions. An overview of all substitution rules can be found in [Cor16].

Example 2.3.2. *Let* $\varphi := (ax^2 + bx + c < 0)$ *be a formula with* $a = 2$*,* $b = -5$ *and* $c = 3$*. The result of virtually substituting* x *by* $-\infty$ *in* φ*, written as* $\varphi[-\infty//x]$*, looks as follows:*

$$\begin{aligned}\varphi[-\infty//x] &\equiv (a < 0) \vee (a = 0 \wedge b > 0) \vee (a = 0 \wedge b = 0 \wedge c < 0)\\ &\equiv (2 < 0) \vee (2 = 0 \wedge -5 > 0) \vee (2 = 0 \wedge -5 = 0 \wedge 3 < 0)\\ &\equiv false \vee false \vee false\\ &\equiv false\end{aligned}$$

2.3.3 Sign-invariant regions

The next concept that is introduced is called *sign-invariant regions*. Therefore, we will first look at the sign function and at regions. Combining them, we can define sign-invariant regions.

Definition 2.3.3 (Sign function)**.** *The* sign *function maps a real value to its sign, i.e.:*

$$sgn : \mathbb{R} \to \{-1,0,1\}, x \mapsto \begin{cases} -1 & , \text{if } x < 0 \\ 0 & , \text{if } x = 0 \\ 1 & , \text{if } x > 0 \end{cases}$$

Definition 2.3.4 (Region and interval)**.** *A set* I *is called a* region *if* I *is a connected subset of* $\mathbb{R}$*. Synonymously, regions are also called* intervals*, which can be written in the following ways:*

$$[a,b] := \{x \in \mathbb{R} \mid a \leq x \leq b\}, \quad (a,b) := \{x \in \mathbb{R} \mid a < x < b\},$$
$$(a,b] := \{x \in \mathbb{R} \mid a < x \leq b\}, \quad [a,b) := \{x \in \mathbb{R} \mid a \leq x < b\}.$$

The two endpoints of an interval I are called bounds. *We call a bound z of an interval I* closed *if $z \in I$; otherwise, it is called* open.

Definition 2.3.5 (Symbolic interval)**.** *A* symbolic interval *$\tilde{I}$ is an interval that has at least one symbolic zero as its bound. Hence, $\tilde{I}$ does not directly correspond to a subset of $\mathbb{R}$ but needs to be evaluated in relation to an assignment that assigns all variables in the bounds. This evaluation is written as $\tilde{I}_\alpha$ where α is an assignment.*

Example 2.3.3. *Let $\tilde{I} := (-\infty, 2x)$ be a symbolic interval and α with $\alpha(x) = 4$ an assignment. Evaluating the bounds in $\tilde{I}$ using α results in $\tilde{I}_\alpha = (-\infty, 2 \cdot 4) = (-\infty, 8)$.*

Definition 2.3.6 (Sign-invariant region)**.** *A region I is called* sign-invariant *for a univariate polynomial p if for all values $a, b \in I$ holds:*

$$sgn(p(a)) = sgn(p(b)).$$

Definition 2.3.7 (Covering)**.** *A covering $\mathcal{COV}$ of a set X is a collection of sets whose union includes X as a subset. Formally, $\mathcal{COV} = \{I_i \mid i \in \mathbb{N}\}$ is a covering of X if $X \subseteq \bigcup_{I_i \in \mathcal{COV}} I_i$.*

Throughout this thesis, the focus lies on coverings of $\mathbb{R}$.

2.3.4 Quantifier elimination

Finally, the quantifier elimination procedure, which eliminates a variable that is bound to an existential or universal quantifier, is briefly explained. A more detailed description about this elimination procedure can be found in [Cor16]. Initially, we introduce ε as a positive number that is infinitesimally close to 0. In order to perform the quantifier elimination, we first have a look at *test candidates*.

Definition 2.3.8 (Test candidates)**.** *Let p be a polynomial that is at most quadratic in the variable x. Let $c := (p \sim 0)$ be a constraint where $\sim \in \{\leq, \geq, <, >, =, \neq\}$ and let z_0, z_1, z_2 be the symbolic zeros of p. The set of* test candidates *for x in c is defined by*

$$tcs(x, p \sim 0) = \begin{cases} \{-\infty, z_0, z_1, z_2\} & \text{, if } \sim \in \{\leq, \geq, =\} \\ \{-\infty, z_0 + \varepsilon, z_1 + \varepsilon, z_2 + \varepsilon\} & \text{, if } \sim \in \{<, >, \neq\} \end{cases}$$

The side conditions *of a test candidate t are defined by*

$$sc(t) = \begin{cases} sc(t') & \text{, if } t = t' + \varepsilon \\ sc(t) & \text{, if } t \text{ is a zero} \\ true & \text{, if } t = -\infty \end{cases}$$

The set of test candidates *for a variable x in a formula φ is defined by*

$$tcs(x, \varphi) = \bigcup_{Constraint\ c\ in\ \varphi} tcs(x, c).$$

Theorem 2.3.1 (Eliminating a quantifier)**.** *Let φ be a quantifier-free formula in which the variable x has a degree of at most 2. Then it holds:*

$$\exists x.\varphi \equiv \bigvee_{t \in tcs(x,\varphi)} (\varphi[t//x] \wedge sc(t)).$$

The elimination of a universal quantifier can be done similarly.

This theorem shows that it is sufficient to insert a finite set of test candidates into a formula φ, which results in a satisfiability-equivalent formula ψ that has one quantifier less than φ.

Chapter 3

Generating explanations using virtual substitution

This chapter presents a method to generate explanations for mcSAT by generalizing a covering using virtual substitution. In order to get a better understanding of the steps that follow, an intuitive description is given after which the three phases of the procedure, namely

1. finding excluded intervals,
2. constructing a covering and
3. generating an explanation,

are explained.

3.1 Idea

The goal of this method is to generate for a given set of constraints R and a partial assignment α an explanation as described in Definition 2.2.2. The set R and α are conflicting such that $\alpha \not\models \exists y(\bigwedge_{c \in R} c)$ holds. The conflict could be explained if VS is used directly by substituting all test candidates into the constraints as given by Theorem 2.3.1. For this conflict, however, not all test candidates are needed necessarily; thus, we try to reduce the effort of VS by constructing a covering instead.

Intuitively, every constraint in R that contains the variable y restricts the values that y can be assigned to. Since the given input is conflicting, the solution set of y is empty.

If we look at the constraint $c_1 = (xy < 0)$ and the assignment β with $\beta(x) = 1$, for example, then c_1 currently equals $(y < 0)$ and the interval $[0, \infty)$ is excluded from the initial solution set $\mathbb{R}$ of y, resulting in $\mathbb{R} \setminus [0, \infty) = (-\infty, 0)$. If we consider a second constraint $c_2 = (y > 2)$, the interval $(-\infty, 2]$ gets excluded. Combining these restrictions, the solution set of y becomes $\mathbb{R} \setminus ((-\infty, 2] \cup [0, \infty)) = \mathbb{R} \setminus (-\infty, \infty) = \emptyset$. Thus, there is no extension of β that assigns y a value and satisfies both constraints. This example can also be visualized by drawing the excluded intervals as seen in Figure 3.1.

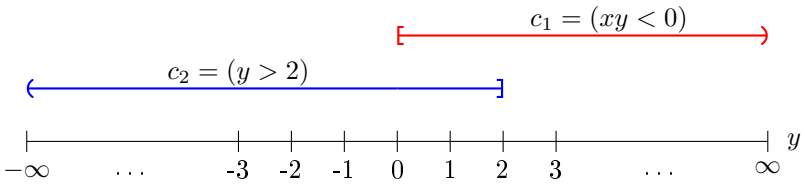

Figure 3.1: Intervals excluded by c_1, c_2 and β

It is noticeable that the intervals are overlapping and forming a covering of $\mathbb{R}$ as given in Definition 2.3.7. One way to explain the conflict is to formalize this concrete covering. However, this explanation would be fairly simple and not applicable for similar conflicts. A more general explanation, which we are aiming for, could use the fact that this covering will remain as long as the bounds are overlapping, no matter at which specific position they are. Thus, the target is to first identify the concrete covering given by R and α, which then gets abstracted to a more general description of the conflict. For the abstraction, we use *interval data structures*, which are defined as follows:

Definition 3.1.1 (Truth-invariant region)**.** *Let c be a constraint that contains the variable y. A region I is called* truth-invariant *for c if for an assignment α and for values $a, b \in I$ holds:*

$$\alpha_{y \mapsto a} \models c \iff \alpha_{y \mapsto b} \models c.$$

Definition 3.1.2 (Interval data structure)**.** *An* interval data structure *can be represented by a tuple $\mathfrak{I} = (\tilde{I}, C, r)$ where $\tilde{I}$ is a symbolic interval, C is a set of constraints called* side conditions*, and r is the constraint from which $\tilde{I}$ is derived and for which $\tilde{I}$ is a truth-invariant region. Typically, r is part of a reason set R as specified in Definition 2.2.2. The existence of the symbolic interval $\tilde{I}$ depends on an assignment α; thus, $\tilde{I}$ exists if and only if α satisfies the constraints in C.*

Additionally, we introduce the function si that maps an interval data structure $\mathfrak{I} = (\tilde{I}, C, r)$ to its symbolic interval $\tilde{I}$.

3.2 Finding intervals

The first step is to find the intervals that are excluded by the given parameters. Therefore, let R be a set of constraints and α a partial assignment as described before. The constraints in R have the form $p \sim 0$ where p is a polynomial with $deg_y(p) \leq 2$ and $\sim \in \{\leq, \geq, <, >, =, \neq\}$. Also, p can be written as $p = ay^2 + by + c$ with a,b,c being polynomials not containing y. The reason why we do not allow $deg_y(p) > 2$ is that we will need to calculate the zeros of p in dependence of y, which requires more effort for higher degrees and can, in general, only be done up to a degree of 4 as explained in [Nic93] and [Shm11]. Lastly, the assignment α assigns all variables x_i in R except y.

In order to find the excluded intervals, we first have to understand that such intervals correspond to sign-invariant regions as described in Definition 2.3.6. Therefore, let $p \sim 0$ be a constraint in R from now on and $p_\alpha := p(\alpha(x_1), \ldots, \alpha(x_n))$ be the polynomial that is the result of replacing the variables in p by their current assignment in α; thus, p_α is univariate in y. The following theorem states the relation between excluded intervals and sign-invariant regions by using truth-invariant regions.

Theorem 3.2.1. *Let p be a polynomial and let α be an assignment. Then it holds that a sign-invariant region I for p_α is also a truth-invariant region for $p \sim 0$ where $\sim \in \{\leq, \geq, <, >, =, \neq\}$.*

In other words, if two values y_1 and y_2 are chosen from a sign-invariant region I, either both satisfy a constraint or none of them does. This property applies to every pair in I; hence, it applies to the entire region I, and I is either completely excluded from the solution set or no value within I is excluded at all. In summary, this means that every excluded interval is equivalent to a sign-invariant region.

Proof. Let p be a polynomial and let α be an assignment. Furthermore, let I be a sign-invariant region for p_α and $y_1, y_2 \in I$. We show that I is truth-invariant for $p \sim 0$, i.e., $\alpha_{y \mapsto y_1} \models (p \sim 0)$ if and only if $\alpha_{y \mapsto y_2} \models (p \sim 0)$.

$$\begin{aligned} \alpha_{y \mapsto y_1} \models (p \sim 0) &\Leftrightarrow p_\alpha(y_1) \sim 0 \\ &\Leftrightarrow sgn(p_\alpha(y_1)) \sim 0 \\ &\Leftrightarrow sgn(p_\alpha(y_2)) \sim 0 \\ &\Leftrightarrow p_\alpha(y_2) \sim 0 \\ &\Leftrightarrow \alpha_{y \mapsto y_2} \models (p \sim 0) \end{aligned}$$ □

Using this theorem, we can infer the following corollary.

Corollary 3.2.2. *Let I be a sign-invariant region for p_α and $k \in I$.*

$$\alpha_{y \mapsto k} \models (p \sim 0) \text{ holds if and only if } \alpha_{y \mapsto j} \models (p \sim 0) \text{ for every } j \in I \text{ holds.}$$

This corollary shows that it is sufficient to only check one value $k \in I$ in order to determine whether all values in I are suitable for satisfying $p \sim 0$. This value k is called a *representative*.

Proof. Let I be a sign-invariant region for p_α and $k \in I$. Assume, $\alpha_{y \mapsto k} \models (p \sim 0)$ holds. By applying Theorem 3.2.1, $\alpha_{y \mapsto j} \models (p \sim 0)$ holds for any $j \in I$; therefore, it holds for every $j \in I$. Furthermore, if $\alpha_{y \mapsto j} \models (p \sim 0)$ holds for every $j \in I$, then especially for $j = k$. □

Referring to Theorem 3.2.1, we can find the excluded intervals by determining all sign-invariant regions. Since a polynomial only changes its sign at its zeros, these are calculated in the following by using the rules given in Definition 2.3.2, namely:

$$\begin{aligned} z_0 &= -\tfrac{c}{b} && \text{, if } a = 0 \text{ and } b \neq 0 \\ z_1 &= \tfrac{-b+\sqrt{b^2-4ac}}{2a} && \text{, if } a \neq 0 \text{ and } b^2 - 4ac \geq 0 \\ z_2 &= \tfrac{-b-\sqrt{b^2-4ac}}{2a} && \text{, if } a \neq 0 \text{ and } b^2 - 4ac > 0 \end{aligned}$$

3.2.1 Linear polynomial with one zero

If p together with the partial assignment α satisfy the side conditions of the first symbolic zero z_0, then p_α is linear and has the zero z_0. As a result, we know that there exist the three sign-invariant regions $(-\infty, z_0), [z_0, z_0]$ and (z_0, ∞).

In order to determine which of these regions relate to excluded intervals, we could have a look at each sign-invariant region one after the other, and by picking a representative from it, check whether it is an excluded interval as given by Corollary 3.2.2.

$\alpha \models (p \sim 0)[-\infty//y]$	$p \sim 0$	excluded intervals
yes	$\sim \in \{\leq, \geq\}$	(z_0, ∞)
	$\sim \in \{<, >\}$	$[z_0, \infty)$
	$\sim \in \{=\}$	$(-\infty, z_0), (z_0, \infty)$
	$\sim \in \{\neq\}$	$[z_0, z_0]$
no	$\sim \in \{\leq, \geq\}$	$(-\infty, z_0)$
	$\sim \in \{<, >\}$	$(-\infty, z_0]$
	$\sim \in \{=\}$	$(-\infty, z_0), (z_0, \infty)$
	$\sim \in \{\neq\}$	$[z_0, z_0]$

Table 3.1: Excluded intervals of a *linear* polynomial having a symbolic zero z_0

On the other hand, it is sufficient to check only the first region and deduce the other regions by making use of the operator $\sim$ and the fact that p_α is linear. An example is given in the following.

Example 3.2.1. *Let $r_1 = (xy - 2 < 0)$ be a constraint and α with $\alpha(x) = 1$ a partial assignment. Thus, the polynomial to be considered is $p = xy - 2$ and has the normal form $p = ay^2 + by + c$ with $a = 0$, $b = x$, $c = -2$. Applying α to p results in $p_\alpha = y - 2$.*

Since p_α is linear in y, the side conditions of z_0 are satisfied and $z_0 = -\frac{c}{b} = -\frac{-2}{x} = 2x^{-1}$. Therefore, the sign-invariant regions are $(-\infty, z_0), [z_0, z_0]$ and (z_0, ∞). Using the current assignment α, we could evaluate z_0 to $2x^{-1} = 2 \cdot 1^{-1} = 2$ in order to find the concrete regions; however, we will already abstract from α and continue with z_0 as a symbolic zero. Now we check if the first region is an excluded interval by picking the representative $-\infty$. This value is chosen because it is always a representative of the leftmost interval. Since $-\infty$ is not a real value, we cannot directly assign it to y but have to substitute it virtually. Hence, we evaluate $r_1[-\infty//y]$ as described in Section 2.3.2:

$$\begin{aligned} \varphi := r_1[-\infty//y] &\equiv (b > 0) \vee (b = 0 \wedge c < 0) \\ &\equiv (x > 0) \vee (x = 0 \wedge -2 < 0) \\ &\equiv (x > 0) \vee (x = 0 \wedge \mathit{true}) \\ &\equiv (x > 0) \vee (x = 0) \end{aligned}$$

The result is that $\alpha \models \varphi$, so $-\infty$ is suitable for satisfying r_1 and the region $(-\infty, z_0)$ is not an excluded interval. However, we can infer due to the monotonicity of linear functions that the region (z_0, ∞) is excluded from the solution set. In order to evaluate the point interval $[z_0, z_0]$, the operator $\sim$ has to be checked. Since $\sim \in \{<\}$ is strict, zeros are no feasible solutions and $[z_0, z_0]$ is excluded, too. All in all, the excluded intervals for the constraint r_1 and the current assignment α are $[z_0, z_0]$ and (z_0, ∞), which can be combined to the interval $[z_0, \infty) = [2x^{-1}, \infty)$.

An overview of all rules for the linear case can be found in Table 3.1. Once each excluded interval I for this concrete constraint and assignment is found, I is transformed into an interval data structure, which encodes the side conditions that need to be satisfied in order for I to be excluded by $p \sim 0$.

Definition 3.2.1 (Interval data structure for a linear polynomial)**.** *Let p be a polynomial and α an assignment such that p_α is linear, i.e., $\alpha \models sc(z_0(p))$. Furthermore, let $p \sim 0$ be a constraint and $\varphi := (p \sim 0)[-\infty//y]$. Then we define the* interval data structure $(\tilde{I}, C, p \sim 0)$ *such that:*

- *$\tilde{I}$ is chosen according to Table 3.1*
- $C = \{sc(z_0(p)), \neg sc(z_1(p))\} \cup \{\varphi \mid \alpha \text{ satisfies } \varphi\} \cup \{\neg\varphi \mid \alpha \text{ satisfies } \neg\varphi\}$

As a side note, $\neg sc(z_2(p))$ does not need to be part of C because $\neg sc(z_1(p))$ already implies $\neg sc(z_2(p))$.

All the constraints in C are satisfied by the current assignment α, and every other assignment that satisfies these side conditions, too, excludes an interval similar to I.

Example 3.2.2. *Assume, we are given the former example with $r_1 = (xy - 2 < 0)$ and $\alpha(x) = 1$ such that the excluded interval is $\tilde{I}_1 := [2x^{-1}, \infty)$, and let $\varphi := r_1[-\infty//y]$. The corresponding interval data structure is $\mathfrak{I}_1 := (\tilde{I}_1, C_1, r_1)$, where C_1 contains the following side conditions:*

- $sc(z_0) \equiv (0 = 0 \wedge x \neq 0) \equiv (x \neq 0)$
- $\neg sc(z_1) \equiv \neg(0 \neq 0 \wedge x^2 - 0 \geq 0) \equiv (0 = 0 \vee x^2 < 0) \equiv true$
- $\varphi \equiv (x > 0) \vee (x = 0)$, *because* $\alpha \models \varphi$

The second constraint can be ignored, because it evaluates to true.

3.2.2 Quadratic polynomial with one or two zeros

If the side conditions $sc(z_0(p))$ are not satisfied, we check $sc(z_1(p))$ instead. Should the conditions be fulfilled, then p_α is quadratic and has at least the one zero z_1. Whether the polynomial has a second zero can be found out by having a look at the side conditions of z_2.

Supposing $sc(z_2(p))$ are not fulfilled, then p_α only has the zero z_1; therefore, the sign-invariant regions are $(-\infty, z_1), [z_1, z_1]$ and (z_1, ∞). By picking $-\infty$ as a representative for the first region, we check whether this region is an excluded interval. Having this knowledge, the other intervals can be inferred similarly as described for the linear case by using the parabola form of quadratic polynomials. Thus, the regions $(-\infty, z_1)$ and (z_1, ∞) are either both excluded or none of them is. The point interval $[z_1, z_1]$ is excluded if $\sim$ is strict.

In case the side conditions of z_2 are fulfilled, then p_α has the two different zeros z_1, z_2. The resulting sign-invariant regions are $(-\infty, z_i), [z_i, z_i], (z_i, z_j), [z_j, z_j]$ and (z_j, ∞) with $i, j \in \{1,2\}, i \neq j$. Since the regions now consist of two symbolic zeros, we need to find out which of the zeros is the leftmost. Intuitively, one could define the formula $\varphi := (z_1 < z_2)$ and check if $\alpha \models \varphi$ holds. The problem, however, is that z_1 and z_2 are square root expressions such that φ is not a formula from the theory of quantifier-free non-linear real arithmetic as given in Definition 2.1.1; therefore, it requires a different approach to check which zero is smaller than the other. An alternative is to define a formula $\psi := (a < b)$ with variables a and b, and then use virtual substitution to get the satisfiability-equivalent formula $\theta := (\psi[z_1//a])[z_2//b]$.

In this case, we use the rules for substituting a square root expression virtually into a formula in order to receive the QFNRA-formula θ for which holds:

$$\alpha \models \varphi \Longleftrightarrow \alpha \models \theta.$$

Should $\alpha \models \theta$ hold, then z_1 is smaller than z_2 and z_i becomes z_1, while z_j is z_2. On condition that it does not hold, z_i is z_2 and z_j is z_1. This formula θ will later also be part of the side conditions in an interval data structure. Having finally found all sign-invariant regions of the polynomial p_α, we can again verify whether the first sign-invariant region $(-\infty, z_i)$ is an excluded interval by using $-\infty$ as a representative and derive the other intervals using the operator $\sim$ and the parabola form of p_α.

Example 3.2.3. *Let $r_2 := (y^2 - 8y + 15x \leq 0)$ be a constraint and α with $\alpha(x) = 1$ the same assignment as in the previous examples. The normal form of $p = y^2 - 8y + 15x$ is $ay^2 + by + c$ with $a = 1$, $b = -8$, $c = 15x$. The side conditions of the first zero $z_0(p)$ are not satisfied, because $a = 1$ does not equal 0; thus, $sc(z_1(p))$ are tested next. It turns out that they are fulfilled by p and α, which results in p_α being quadratic and having at least one zero. Testing $sc(z_2(p))$ for satisfiability is also successful such that p_α has the two symbolic zeros*

$$\begin{aligned} z_1 &= \frac{-b + \sqrt{b^2 - 4ac}}{2a} = \frac{-(-8) + \sqrt{(-8)^2 - 4 \cdot 1 \cdot 15x}}{2 \cdot 1} \\ &= \frac{8 + \sqrt{64 - 60x}}{2} = 4 + \sqrt{16 - 15x} \\ z_2 &= \frac{-b - \sqrt{b^2 - 4ac}}{2a} = \frac{-(-8) - \sqrt{(-8)^2 - 4 \cdot 1 \cdot 15x}}{2 \cdot 1} \\ &= \frac{8 - \sqrt{64 - 60x}}{2} = 4 - \sqrt{16 - 15x} \end{aligned}$$

The sign-invariant regions are $(-\infty, z_i), [z_i, z_i], (z_i, z_j), [z_j, z_j]$ and (z_j, ∞) with $i, j \in \{1,2\}, i \neq j$. The next step is to find out which zero z_1, z_2 relates to either z_i or z_j. This can be done as described before by first determining $\theta = (\psi[z_1//a])[z_2//b]$ and then checking if $\alpha \models \theta$. The concrete calculation of θ is skipped; however, $\alpha \models \theta$ does not hold. This can also be seen by evaluating z_1 to 5 and z_2 to 3 if α is considered. Thus, z_1 is the rightmost zero such that $z_i = z_2$ and $z_j = z_1$ holds. For this example, the sign-invariant regions result in $(-\infty, z_2), [z_2, z_2], (z_2, z_1), [z_1, z_1]$ and (z_1, ∞). By using $-\infty$ as a representative for the first region, it follows:

$$\begin{aligned} \varphi := r_2[-\infty//y] &\equiv (a < 0) \vee (a = 0 \wedge b > 0) \vee (a = 0 \wedge b = 0 \wedge c \leq 0) \\ &\equiv (1 < 0) \vee (-8 = 0 \wedge -8 > 0) \vee (1 = 0 \wedge -8 = 0 \wedge 15 \leq 0) \\ &\equiv false \end{aligned}$$

The assignment α does not satisfy φ which makes the first region $(-\infty, z_2)$ an excluded interval. Due to the parabola form of p_α, the last region (z_1, ∞) is also excluded, but the third region (z_2, z_1) is not. Finally, we check the point intervals $[z_2, z_2]$ and $[z_1, z_1]$. Since $\sim$ is weak, zeros are feasible solutions for r and these two intervals do not correspond to excluded intervals. As a result, the excluded intervals are $(-\infty, z_2) = (-\infty, 4 - \sqrt{16 - 15x})$ and $(z_1, \infty) = (4 + \sqrt{16 - 15x}, \infty)$.

All in all, Table 3.2 lists up the cases and excluded intervals for a quadratic polynomial. As for linear polynomials, the excluded intervals now need to be abstracted to interval data structures.

$\alpha \models sc(z_2)$	$\alpha \models (p \sim 0)[-\infty//y]$	$p \sim 0$	excluded intervals
no	yes	$\sim \in \{\leq, \geq\}$	no interval
		$\sim \in \{<, >\}$	$[z_1, z_1]$
		$\sim \in \{=\}$	$(-\infty, z_1), (z_1, \infty)$
		$\sim \in \{\neq\}$	$[z_1, z_1]$
	no	$\sim \in \{\leq, \geq\}$	$(-\infty, z_1), (z_1, \infty)$
		$\sim \in \{<, >\}$	$(-\infty, \infty)$
		$\sim \in \{=\}$	$(-\infty, z_1), (z_1, \infty)$
		$\sim \in \{\neq\}$	$[z_1, z_1]$
yes	yes	$\sim \in \{\leq, \geq\}$	(z_i, z_j)
		$\sim \in \{<, >\}$	$[z_i, z_j]$
		$\sim \in \{=\}$	$(-\infty, z_i), (z_i, z_j), (z_j, \infty)$
		$\sim \in \{\neq\}$	$[z_i, z_i], [z_j, z_j]$
	no	$\sim \in \{\leq, \geq\}$	$(-\infty, z_i), (z_j, \infty)$
		$\sim \in \{<, >\}$	$(-\infty, z_i], [z_j, \infty)$
		$\sim \in \{=\}$	$(-\infty, z_i), (z_i, z_j), (z_j, \infty)$
		$\sim \in \{\neq\}$	$[z_i, z_i], [z_j, z_j]$

Table 3.2: Excluded intervals of a *quadratic* polynomial having a zero z_1 or zeros z_i, z_j with $z_i < z_j$

Definition 3.2.2 (Interval data structure for a quadratic polynomial)**.** *Let p be a polynomial and α an assignment such that p_α is quadratic, i.e., $\alpha \models sc(z_1(p))$. Furthermore, let $p \sim 0$ be a constraint, $\varphi := (p \sim 0)[-\infty//y]$ and $\theta := (\psi[z_1(p)//a])[z_2(p)//b]$ with $\psi := (a < b)$. Then we define the* interval data structure $(\tilde{I}, C, p \sim 0)$ *such that:*

- *$\tilde{I}$ is chosen according to Table 3.2*
- $C = \{\neg sc(z_0(p)), sc(z_1(p))\} \cup \{\varphi \mid \alpha \text{ satisfies } \varphi\} \cup \{\neg\varphi \mid \alpha \text{ satisfies } \neg\varphi\}$
  $\cup \{sc(z_2(p)) \mid \alpha \text{ satisfies } sc(z_2(p))\} \cup \{\neg sc(z_2(p)) \mid \alpha \text{ satisfies } \neg sc(z_2(p))\}$
  $\cup \{\theta \mid \alpha \text{ satisfies } \theta \wedge sc(z_2(p))\} \cup \{\neg\theta \mid \alpha \text{ satisfies } \neg\theta \wedge sc(z_2(p))\}$

Example 3.2.4. *Let $r_2 = (y^2 - 8y + 15x \leq 0)$ be the previous example with $\alpha(x) = 1$ and the excluded intervals $\tilde{I}_2 := (-\infty, 4 - \sqrt{16 - 15x})$ and $\tilde{I}_3 := (4 + \sqrt{16 - 15x}, \infty)$. Transforming them into interval data structures results in $\mathfrak{I}_2 = (\tilde{I}_2, C_2, r_2)$ and $\mathfrak{I}_3 = (\tilde{I}_3, C_2, r_2)$, where the side conditions of both data structures are as follows:*

- $\neg sc(z_0(p)) \equiv \neg(1 = 0 \wedge -8 \neq 0) \equiv true$
- $sc(z_1(p)) \equiv (1 \neq 0 \wedge (-8)^2 - 4 \cdot 15x \geq 0) \equiv (64 - 60x \geq 0)$
- $\neg\varphi \equiv \neg(r_2[-\infty//y])$, *because* $\alpha \models \neg\varphi$
- $sc(z_2(p)) \equiv (1 \neq 0 \wedge (-8)^2 - 4 \cdot 15x > 0) \equiv (64 - 60x > 0)$, *because* $\alpha \models sc(z_2(p))$
- $\neg\theta \equiv \neg(((a < b)[z_1(p)//a])[z_2(p)//b])$, *because* $\alpha \models \neg\theta$

$\alpha \models (p \sim 0)[-\infty//y]$	excluded intervals
$true$	no interval
$false$	$(-\infty, \infty)$

Table 3.3: Excluded intervals of a polynomial without zeros

3.2.3 Polynomial without zeros

If none of the zeros $z_0(p), z_1(p), z_2(p)$ exist because their side conditions are not satisfied, the polynomial p_α does not have any zeros. Thus, the only sign-invariant region that needs to be checked is $(-\infty, \infty)$. Assuming that $\alpha \models (p \sim 0)[-\infty//y]$ holds, there is no interval being excluded by the current constraint and the partial assignment. Otherwise, the interval $(-\infty, \infty)$, which corresponds to $\mathbb{R}$, is excluded as given in Table 3.3.

Definition 3.2.3 (Interval data structure for a polynomial without zeros)**.** *Let p be a polynomial and α an assignment such that p_α has no zeros, i.e., $\alpha \models \neg sc(z_0(p))$ and $\alpha \models \neg sc(z_1(p))$. Furthermore, let $p \sim 0$ be a constraint and $\varphi := (p \sim 0)[-\infty//y]$. If $\alpha \models \varphi$, then the* interval data structure *$(\tilde{I}, C, p \sim 0)$ is defined such that:*

- $\tilde{I} = (-\infty, \infty)$
- $C = \{\neg sc(z_0), \neg sc(z_1)\}$

Otherwise, no interval data structure is needed.

Algorithm 1 Algorithm for finding excluded intervals

```
1: function findIntervals(ReasonSet R, Assignment α, Variable var)
2:     IntervalDataStructures IDS = ∅
3:     for Constraint (p ~ 0) ∈ R do
4:         Calculate symbolic zeros z_0(p), z_1(p), z_2(p)
5:         if sc(z_0(p)) are fulfilled then                    ▷ linear polynomial
6:             Get excluded intervals from Table 3.1 using φ and α
7:             Transform intervals to interval data structures
8:             Add interval data structures to IDS
9:         else if sc(z_1(p)) are fulfilled then               ▷ quadratic polynomial
10:            if sc(z_2(p)) are fulfilled then
11:                Determine leftmost zero z_i
12:                Get excluded intervals from Table 3.2 using φ and α
13:            else
14:                Get excluded intervals from Table 3.2 using φ and α
15:            end if
16:            Transform intervals to interval data structures
17:            Add interval data structures to IDS
18:        else                                                ▷ No zeros
19:            Get excluded intervals from Table 3.3 using φ and α
20:            Transform intervals to interval data structures
21:            Add interval data structures to IDS
22:        end if
23:    end for
24:    return IDS
25: end function
```

Having considered the different cases for a constraint $p \sim 0$ and assignment α, this step is repeated for every other constraint $r \in R$ as shown in Algorithm 1. Thus, this phase of finding excluded intervals results in a set $\mathcal{IDS}$ of interval data structures, which is used in the following to find a covering.

3.3 Constructing a covering

Once all excluded intervals have been identified and a set of interval data structures was collected, the next step is to construct a covering using this set. It is important to mention that the set may contain more data structures than needed to construct such a covering. Additionally, there might be multiple combinations of intervals such that different coverings could result. In the following, however, the covering being constructed uses as few intervals as possible in order to reduce the length of the generated explanation; therefore, the intervals with the largest diameters between their bounds are preferred. Furthermore, it is guaranteed that at least one covering can be found since the given set R of constraints and the partial assignment α are conflicting. This property is shown by the following theorem.

Theorem 3.3.1. *Let R be a set of constraints and α an assignment. If R and α are conflicting, i.e., $\alpha \not\models \exists y(\bigwedge_{c \in R} c)$ holds, then there exists a covering of $\mathbb{R}$ that is formed by the excluded intervals of constraints in R.*

Proof. Let the set R and the assignment α be conflicting. We assume, it does not exist a covering of $\mathbb{R}$ using excluded intervals. Therefore, there has to exist an interval $I \subseteq \mathbb{R}$ that is not excluded by any of the constraints. By picking any value $k \in I$ and assigning it to y, the extension $\alpha_{y \mapsto k}$ satisfies $\varphi := \bigwedge_{c \in R} c$. It cannot be the case that $\alpha_{y \mapsto k}$ does not satisfy φ, because k would be excluded otherwise by a constraint in R, which contradicts the fact that I is not excluded. Thus, φ and $\alpha_{y \mapsto k}$ are not conflicting such that φ and α cannot be conflicting either, because $\alpha_{y \mapsto k}$ is an extension of α. This is a contradiction to the initial assumption; hence, there must exist a covering of $\mathbb{R}$. □

Now, let $\mathcal{IDS}$ be the set of interval data structures that is the result of the previous phase. As described in Section 3.1, the concrete covering for the given constraint and assignment is built first, which then gets abstracted to a general one. In the following, let $\mathcal{COV}$ be the set of interval data structures that are actually needed to build a covering; initially, this set is empty. In the beginning, the set $\mathcal{IDS}$ is sorted to simplify the construction. Hence, the data structures whose intervals have the smallest left bound according to α come first. Should two intervals have the same left bound, the one with a closed left bound is preferred over one with an open left bound. If the left bounds of both intervals are open respectively closed, then the one with the larger right bound is chosen. Should two intervals have the same right bound, then a closed bound is again preferred over an open one. The following example shows the application of this sorting principle.

Example 3.3.1. *Let $\mathcal{IDS}$ be the set containing the interval data structures $\mathfrak{I}_1 = (\tilde{I}_1, C_1, r_1)$, $\mathfrak{I}_2 = (\tilde{I}_2, C_2, r_2)$ and $\mathfrak{I}_3 = (\tilde{I}_3, C_2, r_2)$ with $\tilde{I}_1 = [2x^{-1}, \infty)$, $\tilde{I}_2 = (-\infty, 4 - \sqrt{16 - 15x})$ and $\tilde{I}_3 = (4 + \sqrt{16 - 15x}, \infty)$ from the former examples, and let α with $\alpha(x) = 1$ be the previously used assignment. Sorting the data structures leads to the following order:*

1. $\mathfrak{I}_2$ *comes first because* $\tilde{I}_2$ *is the only symbolic interval whose left bound is* $-\infty$
2. $\mathfrak{I}_1$ *comes next because* $\alpha \models (2x^{-1} < 4 + \sqrt{16 - 15x})$*, i.e., the left bound of* $\tilde{I}_1$ *is smaller than the left bound of* $\tilde{I}_3$
3. $\mathfrak{I}_3$ *remains; thus, it is the last*

After $\mathcal{IDS}$ has been sorted, the construction of the covering can be done by selecting the first interval data structure $\mathfrak{I} = (\tilde{I}, C, r)$ in $\mathcal{IDS}$ and adding it to $\mathcal{COV}$. Due to the sorting, the left bound of $\tilde{I}$ is always $-\infty$. If the right bound of $\tilde{I}$ is not ∞, the next structure $\mathfrak{I}' = (\tilde{I}', C', r') \in \mathcal{IDS}$ is picked in order to extend the first interval and to form a covering. Whether $\tilde{I}'$ is suitable for extending the current covering can be found out by validating $\tilde{I}_\alpha \subset \tilde{I}_\alpha \cup \tilde{I}'_\alpha$. Providing it holds, $\mathfrak{I}'$ is added to $\mathcal{COV}$; otherwise, this data structure is skipped. This procedure continues with the next interval data structure in $\mathcal{IDS}$ until the first structure $\mathfrak{I}''$ is found in which $si(\mathfrak{I}'')$ has ∞ as its right bound. Since a covering does exist, this will be the case after some iterations and $\mathcal{COV}$ will contain enough data structures to build a covering. Then, every neighbouring pair of intervals in $\mathcal{COV}$ is overlapping, i.e., the right bound of $si(\mathfrak{I}_i)$ overlaps with the left bound of $si(\mathfrak{I}_{i+1})$ for every $\mathfrak{I}_i, \mathfrak{I}_{i+1} \in \mathcal{COV}$ with $1 \leq i < |\mathcal{COV}|$. Lastly, Algorithm 2 gives an overview of this phase.

Example 3.3.2. *Let* $\mathcal{IDS} = \{\mathfrak{I}_2, \mathfrak{I}_1, \mathfrak{I}_3\}$ *be the sorted set from the former example and* α *with* $\alpha(x) = 1$ *the partial assignment. Initially,* $\mathfrak{I}_2$ *is added to* $\mathcal{COV}$*. Since the right bound of* $si(\mathfrak{I}_2)$ *is not* ∞*, we continue with the next interval data structure* $\mathfrak{I}_1$*. Whether* $\mathfrak{I}_1$ *extends the current covering is checked by looking at* $si(\mathfrak{I}_2)_\alpha \subset si(\mathfrak{I}_2)_\alpha \cup si(\mathfrak{I}_1)_\alpha$*. Since* $si(\mathfrak{I}_2)_\alpha = (-\infty, 3)$ *and* $si(\mathfrak{I}_1)_\alpha = [2, \infty)$*, this relation holds such that* $\mathfrak{I}_2$ *is also inserted into* $\mathcal{COV}$*. The procedure stops now because the right bound of* $si(\mathfrak{I}_2)$ *is* ∞*; as a result,* $\mathcal{COV}$ *contains enough intervals for a covering. However, the interval data structure* $\mathfrak{I}_3$ *is skipped and no longer needed. All in all,* $\mathcal{COV} = \{\mathfrak{I}_2, \mathfrak{I}_1\}$*.*

Algorithm 2 Algorithm for constructing a covering

```
1:  function getCovering(IntervalDataStructures IDS, Assignment α)
2:      Covering COV = ∅
3:      Sort IDS using α
4:      for ℑ ∈ IDS do
5:          if ℑ extends current covering COV then
6:              Add ℑ to COV
7:          end if
8:          if COV is a covering of ℝ then
9:              Break loop
10:         end if
11:     end for
12:     return COV
13: end function
```

3.4 Generating an explanation

After having constructed a covering that consists of interval data structures, the last step of the procedure is to generate an explanation for the conflict between R and α; therefore, let $\mathcal{COV}$ be the covering from the former phase.

The explanation E needs to have the form $\varphi \to \psi$ as specified in Definition 2.2.2. In this case, φ is a conjunction over constraints in $R' \subseteq R$, while ψ equals the formula requiring that the current covering and coverings similar to it do not exist. In other words, if $\neg\psi$ is satisfied by an assignment β, then R and β are conflicting, and a covering over $\mathbb{R}$ can be found. The implication $\varphi \to \psi = (\bigwedge_{c_i \in R'} c_i) \to \neg(cov_1 \wedge \cdots \wedge cov_m)$ is equivalent to $(\neg c_1 \vee \cdots \vee \neg c_n) \vee (\neg cov_1 \vee \cdots \vee \neg cov_m)$ where cov_i are the conditions that describe a covering. As a result, the explanation is just a disjunction over negated constraints, for which we use the set *expl* in order to collect all the disjuncts that are then used to form E.

In the beginning, every constraint $r \in R$ that is needed for the covering is added to the explanation. These constraints were collected in the interval data structures; thus, $\neg r \in expl$ for every $\mathfrak{I} = (\tilde{I}, C, r) \in \mathcal{COV}$. We continue by adding the negation of all side conditions in an interval data structure $\mathfrak{I} \in \mathcal{COV}$ to *expl*, i.e., $\neg(\bigwedge_{c \in C} c) \equiv (\bigvee_{c \in C} \neg c) \in expl$ for every $\mathfrak{I} = (\tilde{I}, C, r) \in \mathcal{COV}$. Therefore, all the constraints that are needed for a symbolic interval to exist are part of *expl*.

Additionally, the overlapping of intervals needs to be considered. If we look at the two symbolic intervals $\tilde{I}_1 = (-\infty, a], \tilde{I}_2 = [b, \infty)$, e.g., then $\theta_1 := (b \leq a)$ expresses that $\tilde{I}_1$ and $\tilde{I}_2$ are overlapping. On the one hand, however, θ_1 might not be an QFNRA-formula if a or b is a square root expression. Therefore, the formula $\theta_2 := ((x \leq y)[b//x])[a//y]$ can be defined, in which x and y are variables that get virtually substituted by a and b. Then, θ_2 is a QFNRA-formula for which holds: $\alpha \models \theta_1 \iff \alpha \models \theta_2$. On the other hand, the bounds of $\tilde{I}_1$ and $\tilde{I}_2$ might be both open. In this case, the operator "$\leq$" in θ_2 needs to be "$<$"; otherwise, the intervals will not overlap, and a point interval that is not excluded from the solution set remains such that no covering is formed.

Since the interval data structures in $\mathcal{COV}$ are sorted, only neighbouring pairs are overlapping; hence, for every pair $\mathfrak{I}_i = (\tilde{I}_i, C_i, r_i)$ and $\mathfrak{I}_{i+1} = (\tilde{I}_{i+1}, C_{i+1}, r_{i+1})$ the formula $\theta_{(\mathfrak{I}_i, \mathfrak{I}_{i+1})}$ is defined similar to the former described formula θ_2. Therefore, let $\theta_{(\mathfrak{I}_i, \mathfrak{I}_{i+1})} := ((x \sim y)[a_{i+1}//x])[b_i//y]$ be a formula where b_i is the right bound of the interval $\tilde{I}_i$, a_{i+1} is the left bound of $\tilde{I}_{i+1}$, and $\sim$ equals "$<$" only if both the left bound of $\tilde{I}_i$ and the right bound of $\tilde{I}_{i+1}$ are open; otherwise, $\sim$ is "$\leq$". Finally, $\neg\theta_{(\mathfrak{I}_i, \mathfrak{I}_{i+1})}$ is added to *expl*.

Having done this for every neighbouring pair in $\mathcal{COV}$, the set *expl* now contains all constraints that are needed for the explanation E; thus, $E := \bigvee_{c \in expl} c$ is returned as an explanation for the conflict between R and α but also prevents that similar assignments are chosen which lead to a covering as in $\mathcal{COV}$. All in all, the returned explanation looks as follows:

$$E := \left(\bigvee_{\mathfrak{I}=(\tilde{I},C,r) \in \mathcal{COV}} \neg r \right) \vee \left(\bigvee_{\mathfrak{I}=(\tilde{I},C,r) \in \mathcal{COV}} (\bigvee_{c \in C} \neg c) \right) \vee \left(\bigvee_{\mathfrak{I}_i, \mathfrak{I}_{i+1} \in \mathcal{COV}} \neg\theta_{(\mathfrak{I}_i, \mathfrak{I}_{i+1})} \right)$$

3.5 Correctness

The generated explanation is correct if it fulfils the three properties given in Definition 2.2.2. Thus, the proof is outlined in the following.

Theorem 3.5.1. *Let R be a set of constraints and α an assignment such that R and α are conflicting. The explanation E that is generated by our procedure has the form $\varphi \to \psi$ where $\varphi \equiv \bigwedge_{c \in R' \subseteq R} c$ and $\alpha \not\models \psi$.*

Proof. Referring to Section 3.4, E has the structure $(\neg c_1 \vee \cdots \vee \neg c_n) \vee (\neg cov_1 \vee \cdots \vee \neg cov_m)$ where $c_i \in R$ and cov_i are the covering conditions. This structure is equivalent to

$$\begin{aligned} &\neg(c_1 \wedge \cdots \wedge c_n) \vee \neg(cov_1 \wedge \cdots \wedge cov_m) \\ \equiv &\underbrace{(c_1 \wedge \cdots \wedge c_n)}_{\varphi :=} \to \underbrace{\neg(cov_1 \wedge \cdots \wedge cov_m)}_{\psi :=} \end{aligned}$$

It is noticeable that $\varphi \equiv \bigwedge_{c \in R' \subseteq R} c$. Additionally, the constraints cov_i in ψ are chosen in Section 3.2 in such a way that $\alpha \models cov_i$ for $1 \leq i \leq m$. Hence, $\alpha \models cov_1 \wedge \cdots \wedge cov_m$ and α does not satisfy $\neg(cov_1 \wedge \cdots \wedge cov_m)$, which equals ψ. □

Theorem 3.5.2. *Let R be a set of constraints and α an assignment such that R and α are conflicting. The generated explanation E for R and α is valid.*

Proof. As shown in the previous theorem, E does have the form $\varphi \to \psi$ where $\varphi \equiv \bigwedge_{c \in R' \subseteq R} c$ and $\alpha \not\models \psi$. The explanation E is valid if for every assignment α holds that $\alpha \models E$. We proof this by looking at $\neg\psi \to \neg\varphi$ which is equivalent to $\varphi \to \psi$.

Assume, E is not valid. Then there exists an assignment α with $\alpha \models \neg\psi$ and $\alpha \not\models \neg\varphi$, i.e., $\alpha \models \varphi$. Since $\neg\psi \equiv \neg(\neg(cov_1 \wedge \cdots \wedge cov_m)) \equiv (cov_1 \wedge \cdots \wedge cov_m)$ and $\alpha \models \neg\psi$, α satisfies all the constraints cov_i; thus, a covering exists which excludes all possible values in $\mathbb{R}$ from the solution set of the variable y. As a result, R and α are conflicting such that α cannot satisfy the constraints c_i in φ. This is a contradiction to the assumption that $\alpha \models \varphi$. Therefore, the assumption cannot hold and E has to be valid. □

Theorem 3.5.3. *Let R be a set of constraints and α an assignment such that R and α are conflicting. Let E be an explanation that is generated by our procedure. Then it holds that every literal l in E is part of the finite basis $\mathbb{B}$.*

Proof. In order to see that $\mathbb{B}$ is finite, it is sufficient to enumerate all literals in $\mathbb{B}$. Since every input formula in mcSAT consists of finitely many constraints, every reason set R is finite, too. Each constraint in R has at most 2 symbolic zeros; thus, there are only finitely many zeros for whole R and only finitely many sign-invariant regions. Some of these sign-invariant regions correspond to intervals that are excluded by a constraint. Hence, there is only a finite number of excluded intervals. Each excluded interval is transformed into an interval data structure, which contains a finite number of side conditions that are used in an explanation. Finally, there are only finitely many overlaps of intervals, because the number of intervals is limited. All in all, the set of literals used in any explanation can be collected in a finite set. □

As shown in the theorems above, every property of an explanation is respected by the presented procedure which leads to the correctness of this method.

Chapter 4

Experimental results

In this chapter, the efficiency of the procedure that was presented in Chapter 3 is analysed. Therefore, it was implemented into the SMT-RAT framework, which is an open source C++ toolbox that allows the checking of quantifier-free real and integer arithmetic formulas for satisfiability [CLJÁ12] [CKJ+15]. SMT-RAT provides several modules which can be combined to different strategies. One of the offered modules is the mcSAT-module [Kb18], which implements the approach of the model-constructing satisfiability calculus as explained in Section 2.2 and is used in the following tests.

In order to evaluate our procedure, it is tested with different inputs such that the result can be compared to the results of other explanation functions. All in all, the following combinations are considered:

1. NLSAT: the initially provided NLSAT-style [JM12] explanation function from the mcSAT-module,
2. VS+NLSAT: an explanation function that uses the traditional VS quantifier elimination without taking the assignment into account [Nal17], and NLSAT as a backend,
3. VSCovering+NLSAT: our approach with NLSAT as an additional backend,
4. VSCovering+VS+NLSAT: our approach using VS and NLSAT as backends.

NLSAT is used as a backend in order to guarantee that at least one of the explanation functions is able to describe the conflict. The reason is that VSCovering, e.g., can only handle constraints that are at most quadratic in the unassigned variable. Additionally, VSCovering requires that only a single variable is not assigned. Should more than one variable be not assigned, then no explanation can be generated using VSCovering. That is why we look at VSCovering+NLSAT as well as VSCovering+VS+NLSAT; VS can handle multiple unassigned variables and could lead to a better result than using NLSAT only. In case that VS also fails in VSCovering+VS+NLSAT, then NLSAT is chosen next.

Since many varying problems as input allow for a better understanding of the performance, we use the QF_NRA benchmark set that is provided by SMT-LIB [BFT16], an international initiative for research and development in SMT. This set consists of round about 12000 different input problems which in the following are tried to be solved with a time limit of 60 seconds and a memory limit of 4 gigabyte. The testing system uses the following processors: 4x 2.1 GHz AMD Opteron with 12 cores each.

	SAT	UNSAT	TIMEOUT	MEMOUT	SEGFAULT
NLSAT	4818	4666	2569	78	3
VS+NLSAT	4955	4782	2325	70	2
VSCovering+NLSAT	4923	4730	2399	82	0
VSCovering+VS + NLSAT	4931	4779	2352	72	0

Table 4.1: Benchmark results of 12134 test files

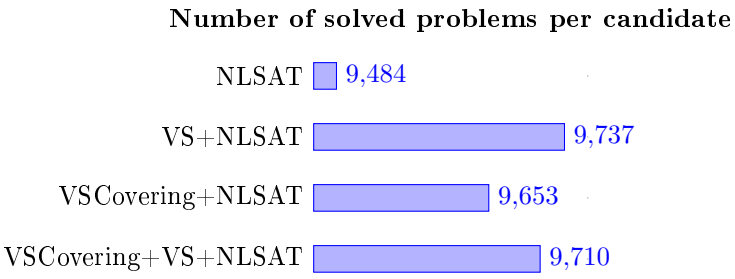

Figure 4.1: Total number of solved problems per candidate

4.1 Overall performance

Firstly, the general performance is analysed. Therefore, the numbers of solved problems, i.e., the inputs that are evaluated to SAT or UNSAT, and the numbers of problems that could not be solved due to the time or memory restriction are collected for each of the above-mentioned alternatives. The results are depicted in Table 4.1. It is important to notice that no wrong answers were returned in any of the tests.

In order to see the relative differences between the explanation functions better, the results are additionally drawn in the bar chart shown in Figure 4.1.

As one can see, NLSAT solves the least number of inputs; VS+NLSAT the most. Our explanation function using VS and NLSAT as backends lies closely behind VS+NLSAT, after which the VSCovering+NLSAT combination follows on the third place. Thus, our implementation is not the best but also not the worst when comparing the total number of solved instances.

4.2 Comparing single instances

The previous results showed the overall performance of each explanation function; however, it was not considered yet how individual inputs are solved. Therefore, we continue by comparing pairwise which instances are solved by which explanation procedure and in which time, but especially have a look at those instances that are only solved by one of the alternatives.

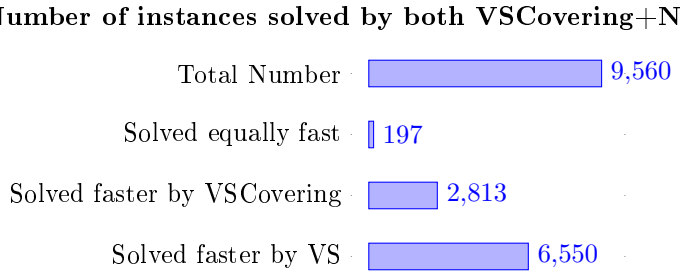

Figure 4.2: Comparison of instances solved by both VSCovering+NLSAT and VS+NLSAT

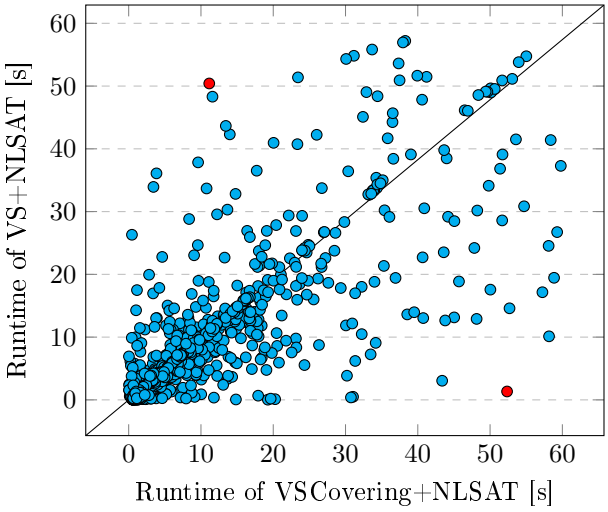

Figure 4.3: Comparison of runtime for instances solved by both VSCovering+NLSAT and VS+NLSAT

4.2.1 VSCovering+NLSAT and VS+NLSAT

Initially, VSCovering+NLSAT and VS+NLSAT are inspected, since they are both using virtual substitution for explanations. As seen in Figure 4.2, there are 9560 input instances that are solved by both procedures. From these instances are 6550 solved faster using VS+NLSAT; the other 3010 examples are solved equally fast or faster by VSCovering+NLSAT. In order to see how much faster the one explanation function over the other is, Figure 4.3 illustrates all instances and their corresponding runtimes. Every point laying above the diagonal line represents an input file that is solved faster by VSCovering+NLSAT; accordingly, every point below the diagonal line shows an input file being solved faster by VS+NLSAT. Every point that lies on the diagonal is solved by both alternatives in the same time.

On the one hand, it stands out that most of the problems are solved within 15 to 20 seconds by both approaches and are laying close to the diagonal. On the other hand, there are instances like the ones marked red which stick out by being evaluated

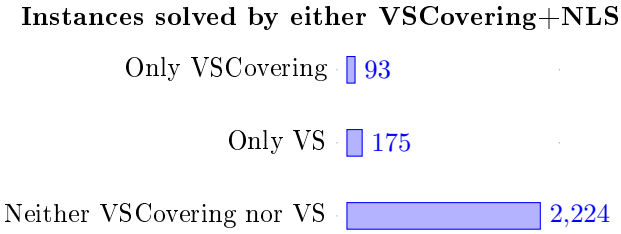

Figure 4.4: Instances solved by either VSCovering+NLSAT or VS+NLSAT

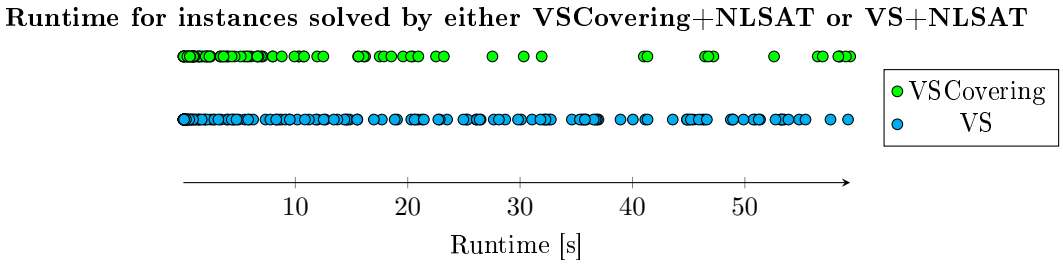

Figure 4.5: Runtime for instances that are solved either by VSCovering+NLSAT or VS+NLSAT

much faster using the one explanation function than by the other and vice-versa.

Moreover, if the instances are considered that can either be solved by VS+NLSAT or VSCovering+NLSAT, then the following numbers, as shown in Figure 4.4, result. There are in total 93 instances that can only be solved by VSCovering+NLSAT without exceeding the given time limit of 60 seconds. The comparison of the computation times of these 93 instances shows that a duration of 13 seconds is needed on average. Similarly, VS+NLSAT can solve 175 input files that cause a timeout if VSCovering+NLSAT is chosen. The average computation time of these lies around 18 seconds. Thus, the instances are in general not close to a timeout but are solved within a quarter of the given time limit. The exact time distribution can be found in Figure 4.5. All in all, both VSCovering+NLSAT and VS+NLSAT do perform on some inputs better and on some inputs worse than the respective other.

4.2.2 VSCovering+NLSAT and NLSAT

The second pair being considered is VSCovering+NLSAT and NLSAT. As done in the former comparison, we start by having a look at the number and runtime of instances that are solved by VSCovering+NLSAT as well as NLSAT, and then continue with inputs that can only be solved by one of them or are even unsolvable. Therefore, Figure 4.6 states that most of the benchmark files can be evaluated faster if NLSAT is used as the explanation function. In Figure 4.7, however, it is shown that the general runtimes between the alternatives are very close, since the values accumulate around the diagonal line.

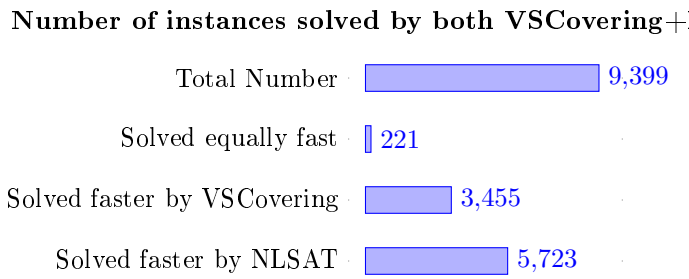

Figure 4.6: Comparison of instances solved by both VSCovering+NLSAT and NLSAT

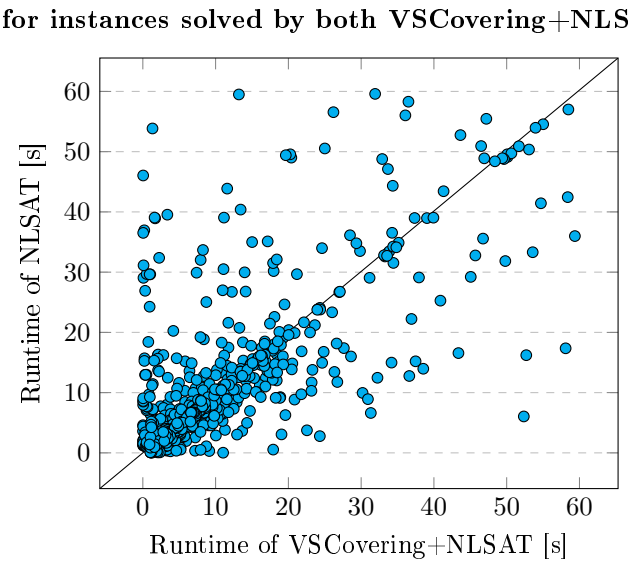

Figure 4.7: Comparison of runtime for instances solved by both VSCovering+NLSAT and NLSAT

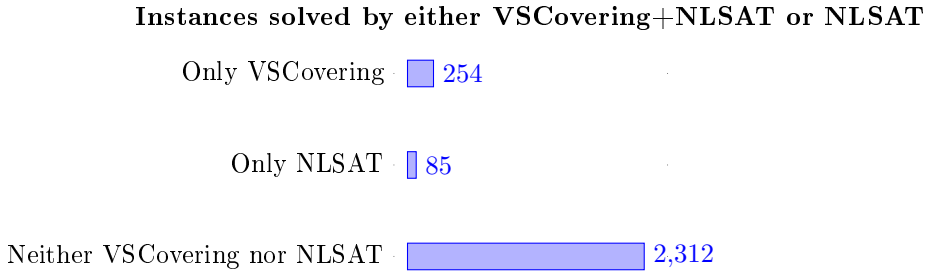

Figure 4.8: Instances solved by either VSCovering+NLSAT or NLSAT

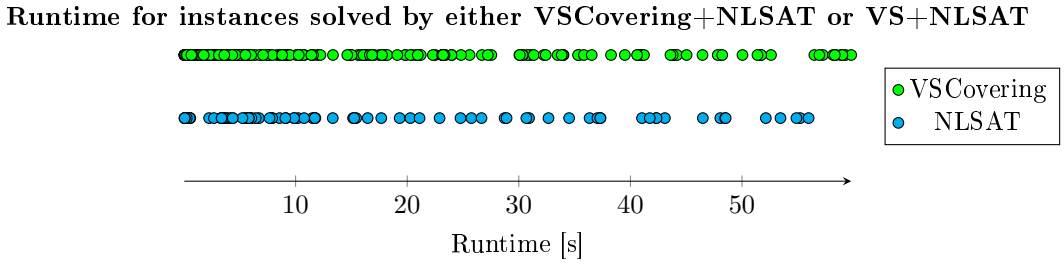

Figure 4.9: Runtime for instances that are solved by either VSCovering+NLSAT or NLSAT+NLSAT

If we look at the files that are solved by only one of the combinations, then there are 254 inputs that VSCovering+NLSAT solves exclusively; NLSAT does only 85 as shown in Figure 4.8 and Figure 4.9. All in all, the difference between VSCovering+NLSAT and NLSAT is larger than between VSCovering+NLSAT and VS+NLSAT. Nevertheless, there is no explanation function that performs better on every single instance compared to the others.

4.3 Discussion

The previous results show that our approach does not solve the problem of generating explanations fundamentally. On the first sight it seems that VS+NLSAT is better than VSCovering+NLSAT. By looking at the individual instances, however, we see that there are inputs which are solved much faster by VSCovering+NLSAT compared to VS+NLSAT. Also, it is noticeable that VSCovering+VS+NLSAT solves more instances than VSCovering+NLSAT. The reason is that VSCovering can only be used for conflicts in which a single variable is unassigned and whose degree is not higher than 2. Thus, VS can handle more cases such that VSCovering+VS+NLSAT leads to better results than VSCovering+NLSAT. Additionally, this holds also for VSCovering+NLSAT against VS+NLSAT; VS is able to handle more cases which results in more solved instances. Surprisingly, VSCovering+VS+NLSAT does not solve as many inputs as VS+NLSAT. One reason could be that the explanation returned by VSCovering+VS+NLSAT is not general enough and does not exclude as many assignments as VS+NLSAT.

Chapter 5

Conclusion

5.1 Future work

Having seen the general idea of constructing coverings and the results of implementing it into the SMT-RAT framework, there are now some options that can be considered in order to improve the procedure. One possibility is to change the way how coverings are constructed. The presented approach aims to use as few intervals as possible for the covering, but other metrics could also be used. As an example, those interval data structures $\mathfrak{I} = (\tilde{I}, C, r)$ could be selected whose constraint r has the smallest degree in the unassigned variable y. Since substituting variables virtually causes a rise in the degree sometimes, we would avoid that the degree of y becomes larger than 2 in order to be able to still calculate symbolic zeros.

Furthermore, extending the computation of zeros to a degree of 4 could also lead to an increasing number of solved inputs. Currently, constraints with degrees larger than 2 are handed over to the backends.

Lastly, one could try to simplify the generated explanation. The idea is to leave out disjunctions that are introduced by virtual substitution but are not satisfied by the current assignment. Therefore, these disjuncts do not influence the satisfiability of the explanation and can be omitted.

5.2 Summary

This thesis started with an introduction to SAT and SMT solving. It was explained why these problems are relevant and which use cases they address. Furthermore, the principle of DPLL(T) was presented and compared to the model-constructing satisfiability calculus (mcSAT), which is a recent approach for SMT solving.

It was explained that one part of mcSAT is a theory solver, whose task it is to assign values to variables, to check for conflicts and especially to resolve these conflicts by generating explanations. Since the explanations have a significant influence in terms of performance, this thesis focussed in the following on an approach for generating such explanations by generalizing coverings using virtual substitution (VS).

After the theoretical requirements were presented and an introduction to VS was given, the three phases of the main procedure were described, which are the following: finding the intervals that are excluded by the given constraints, constructing a covering

using these intervals, and transforming this covering into an explanation for mcSAT.

In the last chapter, the presented procedure was implemented into the SMT-RAT framework and evaluated on round about 12000 benchmark files. During the tests, the runtimes and numbers of solved inputs were collected. These values showed that this procedure can compete with other explanation functions but is not outperforming them when comparing single instances. Thus, it cannot be decided for sure which method should be preferred. In fact, they complement one another such that in total an even higher number of instances could be solved if the approaches are used in parallel, instead of using only one at a time.

All in all, the ideas and results that were achieved may lead to a further development of this procedure or could even improve other approaches, resulting in faster SMT solvers.

Bibliography

[BFT16] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.

[CKJ+15] Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp, and Erika Ábrahám. SMT-RAT: an open source C++ toolbox for strategic and parallel SMT solving. In *LNCS*, volume 9340, pages 360–368. Springer, 2015.

[CLJÁ12] Florian Corzilius, Ulrich Loup, Sebastian Junges, and Erika Ábrahám. Smt-rat: An SMT-compliant nonlinear real arithmetic toolbox. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 442–448. Springer, 2012.

[Cor16] Florian Corzilius. *Integrating virtual substitution into strategic SMT solving.* PhD thesis, RWTH Aachen University, Germany, 2016.

[JM12] Dejan Jovanović and Leonardo de Moura. Solving non-linear arithmetic. In *International Joint Conference on Automated Reasoning*, pages 339–354. Springer, 2012.

[Kb18] Gereon Kremer and Erika Ábrahám. Modular strategic SMT solving with SMT-RAT. *Acta Universitatis Sapientiae / Informatica*, 10(1):pages 5–25, 2018.

[LW93] Rüdiger Loos and Volker Weispfenning. Applying linear quantifier elimination. *The computer journal*, 36(5):450–462, 1993.

[MJ13] Leonardo de Moura and Dejan Jovanović. A model-constructing satisfiability calculus. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 1–12. Springer, 2013.

[Nal17] Jasper Nalbach. Embedding the virtual substitution in the MCSAT framework. Bachelor's thesis, RWTH Aachen University, 2017.

[Nic93] Richard WD Nickalls. A new approach to solving the cubic: Cardan's solution revealed. *The Mathematical Gazette*, 77(480):354–359, 1993.

[Shm11] Sergei L Shmakov. A universal method of solving quartic equations. *Int. J. Pure Appl. Math*, 71(2):251–259, 2011.

[Wei97] Volker Weispfenning. Quantifier elimination for real algebra—the quadratic case and beyond. *Applicable Algebra in Engineering, Communication and Computing*, 8(2):85–101, 1997.