Embedding the Virtual Substitution Method in the Model Constructing Satisfiability Calculus Framework (Work-in-progress Paper)

Erika Ábrahám, Jasper Nalbach, and Gereon Kremer

RWTH Aachen University, Germany

Abstract. *Satisfiability-modulo-theories* (*SMT*) *solving* is a technique to check the satisfiability of logical formulas. In the context of SMT solving, recently a novel technique called the *model-constructing satisfiability calculus* (*MCSAT*) was introduced in [13, 9], with a nice embedding of the cylindrical algebraic decomposition method as a theory solving module for *non-linear real arithmetic* [7]. In this paper we report on our work in progress on the embedding of the *virtual substitution method* in the MCSAT framework.

1 Problem Statement

Satisfiability-modulo-theories (*SMT*) solving [1, 11] is a technique for checking the satisfiability of (usually quantifier-free) first-order logic formulas, i.e., Boolean combinations of constraints from an underlying theory. Typically, SMT solvers have two main components: a DPLL-CDCL-style *SAT solver* [4, 3, 12], which is responsible for the satisfaction of the Boolean structure of the formula, and one or more *theory solvers*, which assure consistency in the underlying theory.

These two components work together as follows: First, a *Boolean abstraction* of the input formula is built by replacing each theory constraint in the quantifier-free input formula by a fresh Boolean proposition, which encodes the truth value of the given theory constraint. The *SAT solver* searches for a satisfying solution for this Boolean abstraction, mainly by *deciding* to try a certain value for a propositional variable, *propagating* this decision to detect implications, and *resolving conflicts* (via Boolean resolution, learning and backtracking) if the current partial assignment is detected to violate the formula. During its search, the SAT solver consults the theory solver(s) from time to time to check whether the constraints, whose abstraction variables are true under the current (possibly partial) Boolean assignment, together with the negation of those constraints whose abstraction variables are set to false[1] are consistent.

If the SAT solver finds a complete Boolean solution for which the theory solver reports consistency then a solution for the input problem is found. If the

[1] Under some mild conditions, only the constraints for the true abstraction variables need to be considered.

SAT solver finds no further solutions for the Boolean abstraction then the input problem is unsatisfiable. Otherwise, if the theory solver reports inconsistency for a (possibly partial) Boolean solution then it generates a *theory lemma* that explains the inconsistency. Learning the Boolean abstraction of this lemma (i.e., building the conjunction of the previous Boolean abstraction with the Boolean abstraction of the lemma) refines the abstraction and should exclude at least the current (theory-conflicting) Boolean assignment from the further search. The search continues until either satisfiability on unsatisfiability is detected.

When solving *quantifier-free non-linear real arithmetic* formulas, the theory solver(s) need to implement a decision procedure for checking the consistency (satisfiability) of conjunctions of polynomial equalities and inequalities over the real domain. Besides the complete *cylindrical algebraic decomposition* (*CAD*) method [2], also some incomplete algorithms like *interval constraint propagation* [5, 6] or the *virtual substitution* (*VS*) *method* [14] can be adapted for this purpose.

Recently, a novel technique called the *model-constructing satisfiability calculus* (*MCSAT*) was introduced in [13, 9]. The main difference to the previously described SMT framework is that MCSAT integrates the search for a theory model into the CDCL-style solving process. The MCSAT solving engine not only generates an assignment to satisfy the Boolean abstraction, but also produces values for the theory variables that are *consistent* with the current Boolean assignment. The future Boolean search is then *directed* by the theory assignment. Intuitively, *consistent* means in this context that the Boolean assignment of an abstraction variable and the evaluation of the corresponding constraint under the theory assignment never disagree; *directed* means that a Boolean decision will only be made if the corresponding constraint is still satisfiable under the current theory assignment.

If the solver wants to extend the current theory assignment by assigning a value to a theory variable x, it may be the case that there exists no such extension that would be consistent with the current Boolean assignment in the above sense. Then a theory solver is invoked with the current partial theory assignment and a conjunction (set) of theory constraints. These constraints do not need to be a priori conflicting, but if we substitute the current partial theory assignment then they become univariate in x and have no common solution. The theory solver produces an explanation (in form of a theory lemma) why the current partial theory assignment cannot be extended to a solution that satisfies all of its input constraints. This essentially corresponds to the role of theory lemmas in a regular SMT setting.

Different arithmetic decision procedures have been embedded as theory solving modules in the MCSAT framework, like the Fourier-Motzkin variable elimination procedure for quantifier-free linear real arithmetic [13], the CAD method for quantifier-free non-linear real arithmetic [7], the CAD method in combination with the branch-and-bound approach for quantifier-free non-linear integer arithmetic [8], and a mixed arithmetic and bit-blasting procedure for fixed-size bit-vector arithmetic [15].

In the following we restrict ourselves to the theory of *quantifier-free non-linear real arithmetic*. The goal of our work is to understand how the *virtual substitution method* can be efficiently embedded in the MCSAT framework. In the next section we explain the virtual substitution method at a high level. In Section 3 we describe our work-in-progress on the embedding of the virtual substitution method in MCSAT. In Section 4 we discuss future work.

2 Real Arithmetic and the Virtual Substitution Method

2.1 Quantifier-free Non-linear Real Arithmetic

Given a finite set $V = \{x_1, \ldots, x_n\}$ of *variables* and a coefficient ring R, a *polynomial* $p \in R[x_1, \ldots, x_n]$ is an expression of the form $p = \sum_{i=1}^{d} a_i \Pi_{j=1}^{n} x_j^{e_{ij}}$ with $a_i \in R$ and $e_{i,j} \in \mathbb{N}_0$ for all $i = 1, \ldots, d$ and $j = 1, \ldots, n$. The expressions $a_i \Pi_{j=1}^{n} x_j^{e_{ij}}$ are called *terms*, in which $\Pi_{j=1}^{n} x_j^{e_{ij}}$ is a *monomial* and a_i its *coefficient*. If $n = 1$ then we call p *univariate*, otherwise *multivariate*.

A multivariate polynomial $p \in R[x_1, \ldots, x_n]$ in variables $x_1, \ldots, x_n$ and coefficients from R can also be seen as a univariate polynomial $p \in R[x_1, \ldots, x_{n-1}][x_n]$ in the *main variable* x_n having polynomial coefficients from the polynomial ring $R[x_1, \ldots, x_{n-1}]$.

The *degree* of a monomial $\Pi_{j=1}^{n} x_j^{e_{ij}}$ is the sum $\sum_{j=i}^{n} e_{i,j}$ of the variables' exponents. The degree of a term is the degree of its monomial. The degree $deg(p)$ of a polynomial p is the highest degree of its terms.

A *polynomial constraint* $p \sim 0$ compares a polynomial to zero by an operator $\sim \in \{=, <, >, \leq, \geq, \neq\}$. A *quantifier-free non-linear real arithmetic formula* is a Boolean combination (using conjunction and negation) of polynomial constraints.

2.2 The Virtual Substitution Method

The *virtual substitution* (*VS*) *method* is a quantifier elimination method for non-linear real arithmetic. In its original form, VS uses *solution equations* and therefore its applicability is restricted in the degree of the polynomials; novel developments allow, however, to extend the method to arbitrary degrees [10]. In this work we consider its application to the elimination of variables that appear at most quadratically in the input constraints as described in [14]. Furthermore, as we consider VS in the SMT solving context, we assume that the input formula for the VS method is a conjunction of polynomial constraints.

For a set of input polynomial constraints with polynomials from $\mathbb{Z}[x_1, \ldots, x_n]$ in which x_n appears at most quadratically, the basic idea is to interpret the polynomials to be univariate in x_n with polynomial coefficients (i.e., as polynomials from $\mathbb{Z}[x_1, \ldots, x_{n-1}][x_n]$) and to use the solution equation for quadratic polynomials to determine their zeros (see the top of Fig. 1).

If we would assign fixed values to the coefficient variables $x_1, \ldots, x_{n-1}$ then all coefficients would evaluate to real values, and each input polynomial would

$p \sim 0 \quad p = ax^2 + bx + c \quad \sim \in \{=, <, >, \leq, \geq, \neq\}$

case	zeros	side condition
constant	$-\infty$	$a = b = 0$
linear	$\xi_0 = -\frac{c}{b}$	$a = 0 \wedge b \neq 0$
quadratic	$\xi_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$	$a \neq 0 \wedge b^2 - 4ac \geq 0$

case	test candidates (samples)						
$p = 0,\ p \leq 0,\ p \geq 0$	$-\infty$	ξ_0	$\xi_0 + \varepsilon$	ξ_1	$\xi_1 + \varepsilon$	ξ_2	$\xi_2 + \varepsilon$
$p \neq 0,\ p < 0,\ p > 0$	$-\infty$	ξ_0	$\xi_0 + \varepsilon$	ξ_1	$\xi_1 + \varepsilon$	ξ_2	$\xi_2 + \varepsilon$

Fig. 1. Test candidates (in black) generated by the VS for an input constraint $p \sim 0$

be (1) either constant zero or constant non-zero, (2) or linear with exactly one real zero, (3) or quadratic with at most two real zeros. These zeros would divide the real domain for x_n into a finite set of regions over which the signs of the polynomials do not change. These regions are given by the zeros themselves, the regions between two successive zeros, and the regions below the smallest zero and above the largest zero. Thus we could pick a *test candidate* from each of the regions and check whether one of them satisfies all input constraints. If it is the case them we have found a solution, otherwise the input constraints are inconsistent.

However, as the coefficients are polynomials, the degree of the polynomials, their zeros and the order of their zeros cannot be determined explicitly. Therefore, the VS method handles them *symbolically*: For each input constraint $p \sim 0$, and for the cases where p is constant, linear or quadratic it specifies all possible real zeros of p along with side conditions for their existence (see the middle of Fig. 1).

Still, this symbolic description does not fix the values of the zeros, nor their order. The VS method makes an elegant trick to overcome this problem: as a test candidate it takes the *smallest* point from each of the above-defined regions. To do so, it allows to use the additional expressions $-\infty$ and $\xi + \epsilon$ to express test candidates symbolically, where $-\infty$ stands for a value smaller than all zeros of all considered polynomials (in x_n), ξ is a (symbolic description of a) zero of one of the input polynomials, and $\xi + \epsilon$ represents a value larger than ξ but smaller than the next largest zero of any of the polynomials.

For each of the input constraints, the symbolic description gives us 7 possible test candidates that cover all possible sign-invariant regions for all possible evaluations of the coefficients: (1) for a constant polynomial, being sign-invariant over the whole real domain, we take $-\infty$ as a representative, (2) for a linear polynomial, having exactly one zero ξ_0, the candidate $-\infty$ represents the region left of this zero, ξ_0 the zero itself, and $\xi_0 + \epsilon$ the region to the right of it, and (3) for the quadratic case with at most two zeros ξ_1 and ξ_2, under the assumption that their existence is assured by the corresponding side conditions,

the candidates $-\infty$, ξ_1, $\xi_1+\epsilon$, ξ_2 and $\xi_2+\epsilon$ represent all sign-invariant regions, independently of the order of the zeros ξ_1 and ξ_2 (see bottom of Fig. 1).

However, VS does not need to consider all these test candidates, as for each constraint $p \sim 0$ the test candidates generated for p must satisfy $p \sim 0$. This implies that for strict inequalities $p \neq 0$, $p < 0$ and $p > 0$ we do not need to consider the zeros of p. Furthermore, for non-strict inequalities it is sufficient to consider $-\infty$ and the zeros of p only and the terms containing ϵ can be neglected. For further explanations we refer to [14].

Example 1. Assume that the VS gets two input constraints[2] $(x-2)^2+(y-2)^2-1<0$ and $x-y=0$. The varieties (zeros) of the polynomials are depicted in Fig. 2. We compute the following test candidates for the elimination of y:

- $-\infty$ with side condition *true*,
- $\xi_0 = x$ with side condition *true* for $x-y=0$,
- $\xi_1+\epsilon$ with $\xi_1 = 2-\sqrt{-x^2+4x-3}$ and side condition $sc_1 = (-x^2+4x-3 \geq 0)$ for $(x-2)^2+(y-2)^2-1<0$, and
- $\xi_2+\epsilon$ with $\xi_2 = 2+\sqrt{-x^2+4x-3}$ and side condition $sc_2 = (-x^2+4x-3 \geq 0)$ for $(x-2)^2+(y-2)^2-1<0$. □

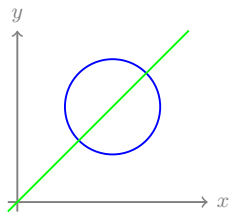

Fig. 2. The varieties of the polynomials $(x-2)^2+(y-2)^2-1$ and $x-y$.

Assume that for a set of input constraints in the variables $x_1, \ldots, x_n$ (at most quadratic in x_n) the VS method generates a set of test candidates for x_n. Now, the set of input constraints is satisfiable if and only if there exists a test candidate for x_n such that if we substitute this test candidate into the constraints than the resulting constraint system is satisfiable. However, there is a last obstacle that needs to be overcome: the test candidates contain the expressions $-\infty$ and ϵ, and they might contain square roots and polynomial fractions. Thus substituting the test candidates directly will yield non-arithmetic formulas. Therefore, the *virtual* substitution defines substitution rules that yield satisfiability-equivalent arithmetic expressions. We do not describe the virtual substitution rules here, but give some examples and refer to [14] for further reading.

Example 2. Virtually substituting the test candidates from Example 1 into the formula $(x-2)^2+(y-2)^2-1<0 \wedge x-y=0$ yields:

- Substituting $-\infty$ for y evaluates to *false*, because the quadratic term will evaluate to $+\infty$ which is not negative.

[2] Our computations do not demonstrate the full power of the VS method, as the equation $x-y=0$ could be used to eliminate x without considering test candidates for the quadratic equation. However, here we show the full computations to illustrate the basic mechanisms of the VS method.

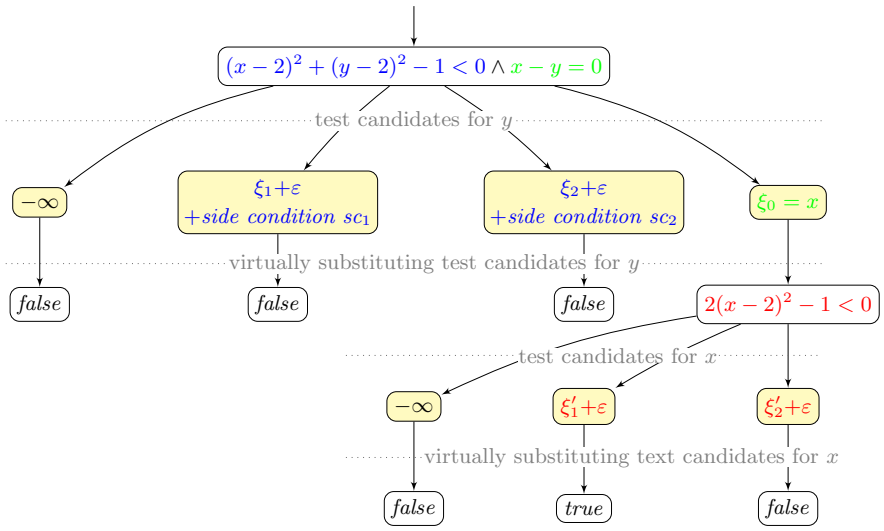

Fig. 3. The VS search tree for the input $(x-2)^2+(y-2)^2-1<0 \wedge x-y=0$ with $\xi_1 = 2-\sqrt{-x^2+4x-3}$, $\xi_2 = 2+\sqrt{-x^2+4x-3}$, $sc_1 = sc_2 = (x^2-4x+3<0)$, $\xi_1' = 2-\sqrt{2}/2$, and $\xi_2' = 2+\sqrt{2}/2$.

- Substituting $\xi_0 = x$ for y evaluates to $2(x-2)^2-1<0$ (by standard substitution).
- Substituting $\xi_1+\epsilon$ with $\xi_1 = 2-\sqrt{-x^2+4x-3}$ and side condition $-x^2+4x-3 \geq 0$ for y evaluates to *false*, because no equality can be satisfied by any value involving an infinitesimal. □

Thus the virtual substitution method generates a search tree: having the input formula φ as root and a variable x_1 (appearing at most quadratically) to be eliminated first, a child node is generated for each test candidate for x_1. The child node for a given test candidate t with side condition sc contains the formula $\varphi' \wedge sc$ after the application of the virtual substitution of t for x_1.

This variable elimination procedure is repeated for all nodes containing non-trivial (*true* or *false*) formulas. The input formula is satisfiable if at least one of the leafs contains the formula *true*.

Example 3. The search tree generated by the virtual substitution for our running example (input constraints $(x-2)^2+(y-2)^2-1<0 \wedge x-y=0$) is shown in Fig. 3. □

3 Embedding the Virtual Substitution Method in the MCSAT Framework

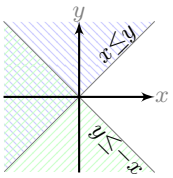

Fig. 4. Illustration for Example 4.

Basically, any *quantifier elimination procedure* can be embedded as a theory solving module in MCSAT. Assume for simplicity a conjunction φ of theory constraints. Assume furthermore an assignment $\nu : \{x_1, \ldots, x_n\} \to \mathbb{R}$ that assigns some real values to the variables $x_1, \ldots, x_n$ such that ν does not evaluate φ to *false*, and such that each extension of ν with an additional value assignment to a variable y evaluates φ to *false*. Then we can apply quantifier elimination to eliminate y in φ, and use the resulting formula as an explanation for non-extensibility.

Example 4. Assume $\varphi \equiv x \leq y \wedge y \leq -x$, and let ν assign the value 1 to x. At this point there is no value for y with which we could extend the theory assignment such that φ is satisfied (see Fig. 4).

If we use the Fourier-Motzkin variable elimination procedure as a theory solving module to explain the conflict, we could eliminate y in $x \leq y \wedge y \leq -x$, which gives us the explanation $(x \leq y \wedge y \leq -x) \to x \leq 0$. □

However, the explanation we get this way might be stronger than we aim for: it might not only explain why the *current* theory assignment cannot be extended to y, but in general it rather gives the weakest precondition for the extensibility of *any* partial assignment with a value for y under the satisfaction of φ.

At the first sight it might seem optimal to derive and learn such weakest preconditions, rather than an explanation for the non-extensibility of the current theory assignment. However, some theory assignments might already be excluded for other reasons (e.g. by previously learnt lemmas). Though in theory it makes no difference, for practical applicability we might want to reduce the size and/or the expressivity of the explanations to reduce the effort for the further search.

Our idea for generating more precise explanations by the VS is based on the computation of a partial VS tree $T(\varphi, \nu, y)$ as follows.

1. We first eliminate by VS the variable y from the input formula φ for which the partial theory assignment ν could not be extended. This results in a set of child formulas $\varphi_1, \ldots, \varphi_I$.
2. For $i = 1, \ldots, I$, we do the following.
   (a) Let $m := 0$ and $\varphi_{i,0} = \varphi_i$.
   (b) If $\varphi_{i,m}$ has no variables (in which case $\varphi_{i,m}$ is *false*), continue with the next formula (i.e., increase i) starting at (2a).
   (c) Otherwise, we select the largest variable x_i appearing in $\varphi_{i,m}$ according to the same variable order in which the partial theory assignment was extended.
   (d) We generate the list $t_1, \ldots, t_l$ containing $-\infty$, and for all zeros ξ of polynomials in $\varphi_{i,m}$ both ξ as well as $\xi + \epsilon$. Let $sc_1, \ldots, sc_l$ be the corresponding side conditions.

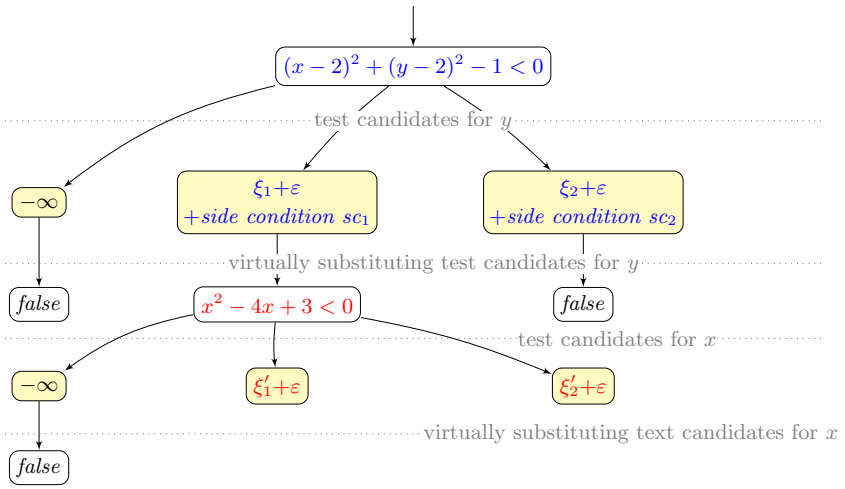

Fig. 5. Illustration for Example 5 with $\xi_1 = 2 - \sqrt{-x^2+4x-3}$, $\xi_2 = 2 + \sqrt{-x^2+4x-3}$, $sc_1 = sc_2 = (x^2 - 4x + 3 < 0)$, $\xi'_1 = 2 - \sqrt{2}/2 \approx 1.29$, and $\xi'_2 = 2 + \sqrt{2}/2 \approx 2.70$.

(e) Remove those elements from the list whose side condition evaluates to *false* under ν.
(f) We evaluate each of the remaining elements t from the list under ν to constants $\nu(t)$, and determine an element $L(\varphi_{i,m}, \nu, x_i) = t^*$ with $\nu(t^*) \leq \nu(x_i)$ and for all other list elements t either $\nu(t) \leq \nu(t^*)$ or $\nu(x_i) < \nu(t)$; this element is a representative for the current x_i-value $\nu(x_i)$ along the considered path.
(g) If t^* is a test candidate according to the VS rules then we virtually substitute this test candidate t^* in $\varphi_{i,m}$, which results in a new child formula $\varphi_{i,m+1}$. We increase m by 1 and continue at (2b).
(h) Otherwise, if t^* is not a test candidate according to the VS rules then we stop the elimination for this path. We continue at (2) with the next formula (increasing i by 1) if there is any formula left to process (i.e., if $i \leq I$); otherwise the algorithm terminates.

This way we want to get a VS path from the root over each of the formulas $\varphi_1, \ldots, \varphi_I$ to a leaf. If we could now symbolically describe each of these paths, then we could generate an explanation stating that if φ holds then each of these path descriptions implies the formula in the leaf node of the corresponding path.

Example 5. This example is illustrated in Fig. 5. Assume a formula $\varphi = ((x-2)^2 + (y-2)^2 - 1 < 0)$ and a partial theory assignment ν with $\nu(x) = 0.5$. This assignment cannot be extended to y, the reason being that the constraint $((x-2)^2 + (y-2)^2 - 1 < 0)[0.5/x] \equiv (y-2)^2 + 1.25 < 0$ is unsatisfiable.

We use the virtual substitution method to eliminate y from $(x-2)^2+(y-2)^2-1<0$, which results in three child formulas $\varphi_1 = (\mathit{false})$, $\varphi_2 = (x^2-4x+3<0)$ and $\varphi_3 = (\mathit{false})$. At this point we could already return the explanation $\varphi \rightarrow (\varphi_1 \vee \varphi_2 \vee \varphi_3)$, i.e., $(x-2)^2+(y-2)^2-1<0 \rightarrow x^2-4x+3<0$, for the theory conflict.

But we could also continue the VS computations along the path that corresponds to the currently selected value $\nu(x) = 0.5$. For the only non-false child node $\varphi_2 = (x^2-4x+3<0)$ we get $L(\varphi_2, \nu, x) = -\infty$. We virtually substitute this test candidate in φ_2, resulting in the formula *false*. Based on this VS computation we could learn the explanation $(x-2)^2+(y-2)^2-1<0 \rightarrow \neg(x<\xi_1)$. □

Next we explain how we can give a general symbolic description D of a path from a non-root node $\varphi_{i,0} = \varphi_i$ of our partial VS tree over the nodes $\varphi_{i,1}, \ldots, \varphi_{i,m-1}$ to a leaf $\varphi_{i,m}$, based on a partial assignment ν to the variables of φ_i such that ν cannot be extended to satisfy φ.

Each such path represents a set of satisfiability-equivalent assignments, but these assignment sets are not cylindrically arranged like it is the case for the cells of the CAD method. This means that there might be two different assignments, represented by the same path, which induce different orders over the test candidates for the variables. Though for each assignment we can uniquely identify the path (up to formula equality) that represents the current theory assignment, the symbolic description of a path is not quite straightforward.

We define the path description $D(\varphi_{i,0}, \ldots, \varphi_{i,M})$ to be $D(\varphi_{i,0}) = \varphi_i$ for $M = 0$ and $\wedge_{m=1}^{M} \Psi(\varphi_{i,m-1}, \varphi_{i,m})$ otherwise. For the basic part, we propose three different definitions for $\Psi(\varphi_{i,m-1}, \varphi_{i,m})$, which we distinguish as Ψ_1, Ψ_2 and Ψ_3; similarly, we will write D_1, D_2 resp. D_3 when using Ψ_1, Ψ_2 resp. Ψ_3. Note that $\varphi_{i,m}$ is a child node of $\varphi_{i,m-1}$. Assume that in the node $\varphi_{i,m-1}$ the VS method generated test candidates for a variable x_i. We consider all (symbolic) zeros $T_{\mathit{true}} = \{t_1, \ldots, t_l, t'_1, \ldots, t'_u\}$ of all polynomials in $\varphi_{i,m-1}$ whose side conditions are *true* under ν, and assume for them w.l.o.g. the order

$$\nu(t_1) \sim_1 \nu(t_2) \sim_2 \ldots \sim_{l-1} \nu(t_l) \sim_l \nu(x_i) < \nu(t'_1) \sim'_1 \ldots \sim'_{u-1} \nu(t'_u)$$

under the current partial theory assignment ν with $\sim_k, \sim'_k \in \{<, =\}$ for all k. If $l = 0$ then there is no test candidate below $\nu(x_i)$; similarly, $u = 0$ represents the case where there is no test candidate larger than $\nu(x_i)$. Let analogously T_{false} be the set of all (symbolic) zeros of all polynomials in $\varphi_{i,m-1}$ whose side conditions are *false* under ν.

$\Psi_1(\varphi_{i,m-1}, \varphi_{i,m})$

$$= \begin{cases} sc(t_l) \wedge x_i = t_l & \text{if } l > 0 \text{ and } \sim_l \text{ is } = \\ \bigwedge_{t \in T_{true}} sc(t) \wedge \bigwedge_{t \in T_{false}} \neg sc(t) \wedge & \\ t_l < x_i \wedge x_i < t'_1 \wedge (\bigwedge_{k=1}^{l-1} t_k \sim_k t_{k+1}) \wedge (\bigwedge_{k=1}^{u-1} t'_k \sim'_k t'_{k+1}) & \\ & \text{if } l > 0,\ u > 0 \text{ and } \sim_l \text{ is } < \\ \bigwedge_{t \in T_{true}} sc(t) \wedge \bigwedge_{t \in T_{false}} \neg sc(t) \wedge t_l < x_i \wedge (\bigwedge_{k=1}^{l-1} t_k \sim_k t_{k+1}) & \\ & \text{if } l > 0,\ u = 0 \text{ and } \sim_l \text{ is } < \\ \bigwedge_{t \in T_{true}} sc(t) \wedge \bigwedge_{t \in T_{false}} \neg sc(t) \wedge x_i < t'_1 \wedge (\bigwedge_{k=1}^{u-1} t'_k \sim_k t'_{k+1}) & \\ & \text{if } l = 0 \text{ and } u > 0 \\ \bigwedge_{t \in T_{false}} \neg sc(t) & \text{else (if } l = 0 \text{ and } u = 0\text{)} \end{cases}$$

$\Psi_2(\varphi_{i,m-1}, \varphi_{i,m})$

$$= \begin{cases} sc(t_l) \wedge x_i = t_l & \text{if } l > 0 \text{ and } \sim_l \text{ is } = \\ sc(t_l) \wedge sc(t'_1) \wedge t_l < x_i \wedge x_i < t'_1 \wedge \bigwedge_{t \in T_{false} \cup T_{true}} sc(t) \rightarrow (t \leq t_l \vee t'_1 \leq t) & \\ & \text{if } l > 0,\ u > 0 \text{ and } \sim_l \text{ is } < \\ sc(t_l) \wedge t_l < x_i \wedge \bigwedge_{t \in T_{false} \cup T_{true}} sc(t) \rightarrow t \leq t_l & \\ & \text{if } l > 0,\ u = 0 \text{ and } \sim_l \text{ is } < \\ sc(t'_1) \wedge x_i < t'_1 \wedge \bigwedge_{t \in T_{false} \cup T_{true}} sc(t) \rightarrow t'_1 \leq t & \text{if } l = 0 \text{ and } u > 0 \\ \bigwedge_{t \in T_{false}} \neg sc(t) & \text{else (if } l = 0 \text{ and } u = 0\text{)} \end{cases}$$

$\Psi_3(\varphi_{i,m-1}, \varphi_{i,m})$

$$= \begin{cases} sc(t_l) \wedge x_i = t_l & \text{if } l > 0 \text{ and } \sim_l \text{ is } = \\ sc(t_l) \wedge t_l < x_i \wedge \bigwedge_{t \in T_{false} \cup T_{true}} sc(t) \rightarrow (t \leq t_l \vee x_i < t) & \\ & \text{if } l > 0 \text{ and } \sim_l \text{ is } < \\ \bigwedge_{t \in T_{false} \cup T_{true}} sc(t) \rightarrow (x_i < t) & \text{else (if } l = 0\text{)} \end{cases}$$

The formula Ψ_1 provides the most strict description of why the current theory assignment ν is not extensible, whereas Ψ_2 and Ψ_3 offer a more general explanation. More formally,

$$(\exists x_i.\ \varphi_{i,m-1} \wedge \Psi_1(\varphi_{i,m-1}, \varphi_{i,m})) \rightarrow \varphi_{i,m}$$
$$(\exists x_i.\ \varphi_{i,m-1} \wedge \Psi_2(\varphi_{i,m-1}, \varphi_{i,m})) \rightarrow \varphi_{i,m}$$
$$(\exists x_i.\ \varphi_{i,m-1} \wedge \Psi_3(\varphi_{i,m-1}, \varphi_{i,m})) \leftrightarrow \varphi_{i,m}$$

and by the definition of D_1, D_2 and D_3 we get

$$(\exists x_1, \ldots, x_i.\ \varphi_{i,0} \wedge D_1(\varphi_{i,0}, \ldots, \varphi_{i,m})) \rightarrow \varphi_{i,m}$$
$$(\exists x_1, \ldots, x_i.\ \varphi_{i,0} \wedge D_2(\varphi_{i,0}, \ldots, \varphi_{i,m})) \rightarrow \varphi_{i,m}$$
$$(\exists x_1, \ldots, x_i.\ \varphi_{i,0} \wedge D_3(\varphi_{i,0}, \ldots, \varphi_{i,m})) \leftrightarrow \varphi_{i,m}$$

Finally, let $\Pi = \{\pi_1, \ldots, \pi_k\}$ be all paths (as formula sequences) in the partial VS tree $T(\varphi, \nu, y)$ from the I children of the root node to the leaves φ_{i,m_i} (note

that all generated leafs are *false* and that there are exactly I leaf nodes). Then

$$\varphi \to (\bigvee_{i=1}^{I} \neg D(\pi_i) \vee \varphi_{i,m_i})$$

is a tautology for $D \in \{D_1, D_2, D_3\}$.

As a last remark, perhaps the reader has recognised that we explicitly referred to zeros of polynomials in the above descriptions. Instead of explicit references, in the implementation we will use a special syntax $zero(i, p)$ as in [7] to refer to the ith zero of a polynomial p.

4 Future Work

The work described in this progress report is ongoing. We do believe that the presented statements are true, but of course the presented ideas need to be formalised more precisely, and proofs for the statements need to be given. After this step, we will develop a pseudo-algorithmic description of the explanation generation for the VS method in the MCSAT context, and implement the approach based on our SMT solver SMT-RAT. Currently we work on the implementation of the CAD embedding in MCSAT. Once that implementation is finished, we can reuse modules (like the SAT module extended with theory decision and propagation) for the embedding of the VS method. The implementation of the MCSAT-embedding of the virtual substitution will allow us to compare the different explanation generation mechanisms discussed in this paper, and thus to collect experiences which ideas are the most promising from the perspective of (time and memory) efficiency.

Though the MCSAT embedding of the CAD method in the Z3 SMT solver has been shown to be very efficient, we believe that harder arithmetic problems can be solved most efficiently by combinations of different decision procedures. To do so, we will use the possibilities offered by SMT-RAT for the strategic combinations of theory solving modules: for quadratic problems we will employ the VS and for (sub-)problems beyond the capabilities of our VS module we will use the CAD method in the context of the MCSAT framework.

References

1. Biere, A., Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
2. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Automata Theory and Formal Languages. LNCS, vol. 33, pp. 134–183. Springer (1975)
3. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM 5(7), 394–397 (1962)
4. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM 7(3), 201–215 (Jul 1960)

5. Gao, S., Ganai, M., Ivančić, F., Gupta, A., Sankaranarayanan, S., Clarke, E.M.: Integrating ICP and LRA solvers for deciding nonlinear real arithmetic problems. In: Proc. of FMCAD'10. pp. 81–90. IEEE (2010)
6. Herbort, S., Ratz, D.: Improving the efficiency of a nonlinear-system-solver using a componentwise Newton method. Tech. Rep. 2/1997, Inst. für Angewandte Mathematik, University of Karlsruhe (1997)
7. Jovanović, D., de Moura, L.: Solving non-linear arithmetic. In: Proc. of IJCAR'12. LNAI, vol. 7364, pp. 339–354. Springer (2012)
8. Jovanović, D.: Solving nonlinear integer arithmetic with MCSAT. In: Proc. of VMCAI'17. LNCS, vol. 10145, pp. 330–346. Springer (2017)
9. Jovanović, D., Barrett, C., de Moura, L.: The design and implementation of the model constructing satisfiability calculus. In: Proc. of FMCAD'13 (2013)
10. Košta, M.: New Concepts for Real Quantifier Elimination by Virtual Substitution. Ph.D. thesis, Saarland University (2016)
11. Kroening, D., Strichman, O.: Decision Procedures: An Algorithmic Point of View. Springer (2008)
12. Marques-silva, J.P., Sakallah, K.A.: Grasp: A search algorithm for propositional satisfiability. IEEE Transactions on Computers 48, 506–521 (1999)
13. de Moura, L., Jovanović, D.: A model-constructing satisfiability calculus. In: Proc. of VMCAI'13. pp. 1–12. Springer (2013)
14. Weispfenning, V.: Quantifier elimination for real algebra - the quadratic case and beyond. Appl. Algebra Eng. Commun. Comput. 8(2), 85–101 (1997)
15. Zeljić, A., Wintersteiger, C.M., Rümmer, P.: Deciding bit-vector formulas with mcSAT. In: Proc. of SAT'16. pp. 249–266. Springer International Publishing (2016)