A note on real quantifier elimination by virtual term substitution of unbounded degree

Kristjan Liiva[1], Grant Olney Passmore[1,2], and Paul B. Jackson[1]

[1]LFCS, School of Informatics, University of Edinburgh
[2]Clare Hall, University of Cambridge

***Abstract*—We describe work in progress on constructing a complete implementation of Weispfenning's virtual term substitution method for real closed field formulas of unbounded degree. We build upon a recent high-performance library for computing over non-Archimedean real closed fields using Thom's Lemma.**

I. Introduction

Weispfenning's virtual term substitution (vts) approach to real quantifier elimination is a compelling alternative to cylindrical algebraic decomposition, with many distinct advantages. High-performance, widely-used implementations of vts exist in tools such as Mathematica [6], Reduce/Redlog [1] and Z3 [3]. However, modulo various special cases, these implementations are all limited to the class of "essentially quadratic" formulas, a vast restriction of the theory of real closed fields.

By exploiting Thom's Lemma, it has long been known that vts can in principle be uniformly lifted beyond the quadratic case to formulas of unbounded degree. Unfortunately, this use of Thom's Lemma is rather intricate, and, to date, we know of no implementation of the complete vts method.

In this note, we present our progress towards this goal. By building upon a recent library for efficiently computing over non-Archimedean real closed fields via a variant of Thom encodings [4], we have completed an implementation of unbounded vts for univariate formulas. Our hope is that experimenting with this implementation will be instructive for understanding the difficulties involved in general vts. We describe this work and what remains in lifting it to the full multivariate case.

II. Preliminaries

We begin by recalling some basic real algebra. Let $\mathbb{K}$ be a computable, finite-degree ordered field extension of $\mathbb{Q}$, and let $\widetilde{\mathbb{K}}$ be its real closure.

(Sign assignment)**.** *Given a set of polynomials $G \subset \mathbb{K}[x]$, a sign assignment σ is a mapping $\sigma : G \to \{1, 0, -1\}$. Sign assignments may be represented by sets of atomic formulas, e.g., $\{g_1 \mapsto -1, g_2 \mapsto 0, g_3 \mapsto 1\}$ by $\{g_1 < 0, g_2 = 0, -g_3 < 0\}$.*

(Sturm sequence)**.** *A Sturm sequence $[s_1, s_2, \ldots, s_k]$ for $f, g \in \mathbb{K}[x]$ is defined inductively as $s_1 = f$, $s_2 = g$, $s_i = -rem(s_{i-2}, s_{i-1})$, s.t. $rem(s_{k-1}, s_k) = 0$. We use $sturm(f, g)$ to denote this Sturm sequence. Given a sequence S, we use $sv(S, a)$ for the number of sign changes (ignoring zeros) in the sequence when each polynomial is evaluated at a. We use $sv(S, a, b)$ for $sv(S, a) - sv(S, b)$.*

(Tarski query)**.** *Given $f, g \in \mathbb{K}[x]$, we define*

$$TaQ(g, f) = \sum_{x \in \mathbb{R}, f(x)=0} \text{sign}(g(x)),$$

where $\text{sign}(\alpha)$ is $-1, 0$ or 1 for α negative, zero or positive respectively. By the Sturm-Tarski theorem we have that $TaQ(g, f) = sv(S, -\infty) - sv(S, +\infty)$, where $S = sturm(f, f'g)$.

(Sign determination)**.** *Given a polynomial f and a set of polynomials $G = \{g_1, \ldots, g_n\}$, we can calculate the list of sign conditions realized by G on the roots of f by using Tarski queries with either [2, Algorithm 10.9. Naive Sign Determination] or more efficiently with [2, Algorithm 10.11. Sign Determination]. We can do that by first computing the number of distinct roots of f that satisfy a sign assignment σ. We denote this quantity by $\#(\sigma, f)$.*

III. Virtual Term Substitution

Let us describe the basic idea of Weispfenning's method[1]. Consider

$$\exists x \varphi(x, y_1, \ldots, y_n)$$

where φ is a $\wedge$, $\vee$-combination of atomic formulas $(f_i \odot_i 0)$, s.t. $f_i \in \mathbb{K}[x, y_1 \ldots, y_n]$, $\odot_i \in \{<, \leq, =, \neq\}$ and $\{1 \leq i \leq m\}$. Then, vts relies upon the following observations [7]:

- When $y_1, \ldots, y_n$ are fixed, the set $S = \{x \in \widetilde{\mathbb{K}} \mid \varphi(x)\}$ is a finite disjoint union of intervals.
- The endpoints of these intervals are either roots of polynomials appearing in φ or $\pm\infty$. The type of interval (closed, half-open or open) depends upon the relation appearing with the polynomial contributing the root.
- In order to test non-emptiness of S, one needs to evaluate φ upon at least one sample point from each interval.
- By suitably extending the language of terms, this test can be expressed[2] as a finite disjunction of formulas $\varphi[t/x]$,

[1]The vts method [7] is classically presented with polynomials drawn from $\mathbb{Q}[x]$ and all roots represented residing in $\mathbb{R}$. However, given the more general class of real closed fields supported by the field arithmetic library upon which we build [4], our implementation is of a more general version. We present vts in this general setting, using $\mathbb{K}$ and $\widetilde{\mathbb{K}}$ instead of $\mathbb{Q}$ and $\mathbb{R}$.

[2]Let α be a root of f_i, then t is of form α if $\tau_i \in \{\leq, =\}$, $\alpha + \epsilon$ if $\tau_i \in \{<, \neq\}$. $t = -\infty$ is substituted without any connection to any atomic formulas and only once.

where $t \in \{\alpha, \alpha+\epsilon, -\infty\}$, $\alpha \in Zer(f_i)$ for some f_i appearing in φ, and ϵ an infinitesimal. Here, $Zer(f_i)$ consists of extended terms representing the set of roots of f_i. We call the set of such t's in the extended term language *test terms* for φ.

- When f_i contains some y_j, this set of roots need not be a fixed subset of $\widetilde{\mathbb{K}}$, but can be represented parametrically in an extended term language, modulo certain non-degeneracy assumptions. For instance, under the non-degeneracy assumption that $y_2^2 - 4y_1y_3 \geq 0$, the set $Zer(y_1x^2 + y_2x + y_3)$ can be represented as $\{-y_2 \pm \frac{\sqrt[2]{y_2^2-4y_1y_3}}{2y_2}\}$. In addition to obtaining extended terms representing roots, one must also obtain extended terms representing sample points drawn from regions containing no roots. Infinitesimals and infinities are used for this in the obvious way. Indeed, extending the term language with a square-root operator $\sqrt[2]{}$, an infinitesimal ϵ, and $-\infty$ forms the basis of the vts method in the quadratic case.
- Finally, once test terms have been suitably substituted for x, simplification can be performed upon the resulting formulas to eliminate all occurrences of extended terms. Moreover, the QE process can be designed so that the non-degeneracy assumptions become encoded in the resulting formulas [7]. The result is a quantifier-free formula in the language of ordered rings over $\mathbb{K}[y_1, \ldots, y_n]$ equivalent to $\exists x \varphi(x, \vec{y})$. For instance, in the quadratic case, this simplification step removes all occurrences of $\sqrt[2]{}$, ϵ, and $-\infty$, and the overall QE process conjoins e.g. the non-negativity of the discriminant as appropriate.

The key difficulties of vts lie in:

1) Devising a method to unambiguously represent all *real*[3] roots of polynomials $p \in \mathbb{K}[x]$ in an extended term language. Recall that for Galois-theoretic reasons, radicals will not in general suffice for polynomials of degree 5 and higher. A general representation working for all polynomials can be derived from Thom's Lemma.
2) Making precise the method of substitution and simplification $\varphi[t/x]$ when t involves extended terms, e.g., infinity, radicals, expressions with infinitesimals, and other extended operators.

Let us first address (2) and the simplifications involving infinitesimals and $-\infty$. Following [7], we can rewrite substitutions on atomic formulas to formulas either with no extended terms or only extended terms involving exact roots.

First note[4] that for any positive infinitesimal ϵ and for all $0 \neq f(x) = \sum_{i=0}^{n} a_i x^i \in \mathbb{K}[x]$, $\alpha \in \mathbb{K}$:

1) $f(\alpha + \epsilon) \neq 0$,
2) $f(\alpha) \neq 0 \Rightarrow f(\alpha) \cdot f(\alpha+\epsilon) > 0$,
3) $0 = f(\alpha) = \ldots = f^{(k-1)}(\alpha) \neq f^{(k)}(\alpha) \Rightarrow f^{(k)}(\alpha) \cdot f(\alpha+\epsilon) > 0$.

[3]Here, *real* means an element of the real closure $\widetilde{\mathbb{K}}$.

[4]We'll assume that if $\mathbb{K}$ contains other infinitesimals than ϵ, then ϵ is infinitely smaller than any of them.

Defining ν as

$$\nu(f) = \begin{cases} f < 0 & \text{if } deg(f) = 0, \\ f < 0 \vee (f = 0 \wedge \nu(f')) & \text{if } deg(f) > 0, \end{cases}$$

For all $g = \sum_{i=0}^{n} b_i x^i \in \mathbb{K}[x]$, we get

$$\begin{aligned} (g = 0)[\alpha+\epsilon/x] &\Leftrightarrow \bigwedge_{i=0}^{n} b_i = 0, \\ (g < 0)[\alpha+\epsilon/x] &\Leftrightarrow \nu(g)[\alpha/x], \\ (g \leq 0)[\alpha+\epsilon/x] &\Leftrightarrow (g < 0)[\alpha+\epsilon/x] \vee (g = 0)[\alpha+\epsilon/x], \\ (g \neq 0)[\alpha+\epsilon/x] &\Leftrightarrow \bigvee_{i=0}^{n} b_i \neq 0. \end{aligned}$$

Keeping in mind that a polynomial $f(x)$ is eventually dominated by its leading term, and by defining μ as

$$\mu(f) = \begin{cases} b_0 < 0 & \\ & \text{if } deg(f) = 0, \\ (-1)^n b_n < 0 \vee (b_n = 0 \wedge \mu(\sum_{i=0}^{n-1} b_i x^i) & \\ & \text{if } deg(f) > 0. \end{cases}$$

we get

$$\begin{aligned} (g < 0)[-\infty/x] &\Leftrightarrow \bigwedge_{i=0}^{n} b_i = 0, \\ (g = 0)[-\infty/x] &\Leftrightarrow \mu(g)[\alpha/x], \\ (g \leq 0)[-\infty/x] &\Leftrightarrow (g < 0)[-\infty/x] \vee (g = 0)[-\infty/x], \\ (g \neq 0)[-\infty/x] &\Leftrightarrow \bigvee_{i=0}^{n} b_i \neq 0. \end{aligned}$$

The unaddressed difficulties — representing roots in an extended term language and performing simplifications to remove the extended terms once substituted — are closely related.

In order to support polynomials of arbitrary degree, we use Thom's lemma:

(Thom's lemma)**.** *Any real root α of $f(x) \in \mathbb{K}[x]$ is uniquely determined by the sign of its derivatives $f'(\alpha), \ldots, f^{d-1}(\alpha)$, where $d = deg(f(x))$. Such a representation of a root is called a* Thom encoding.

For a full QE procedure, we need:

1) A method for "finding" all roots of a (parametric) polynomial and representing them using Thom encodings,
2) A simplification method for eliminating extended terms from formulas.

Combining (1) and (2) we get in essence a specialized form of the quantifier elimination problem. And although we could use CAD or any other full QE procedure, we have reason to be believe that by using the special structure of the problem there is a more efficient way to solve it.

IV. Univariate VTS as Exemplar

To illustrate the machinery described thus far, we shall develop vts of unbounded degree for the restricted case of univariate formulas. That is, we shall present (1) and (2) for univariate polynomials.

Let us postpone (1) until the next section. For now, we shall assume that we have in hand a Thom encoding of each the root of all polynomials appearing in φ.

As in [4], we are only using a smaller discriminating sign assignment on derivatives of polynomial as long as there is only one root satisfying that assignment. Denote this discriminating sign assignment corresponding to root α as $Disc_{f,\alpha}$ and denote $Der_{f,\alpha}$ as a set of derivatives that are in $Disc_{f,\alpha}$ (both of these are trivially computable given Thom encodings of all the roots of a polynomial).

Let α be a root of $f(x)$. By using the algorithm for sign determination, we can determine $\#(Disc_{f,\alpha} \cup \{g = 0\}, f(x))$, $\#(Disc_{f,\alpha} \cup \{g > 0\}, f(x))$ and $\#(Disc_{f,\alpha} \cup \{g < 0\}, f(x))$. From them we can determine $\odot$ such that $(g(\alpha) \odot 0)$ holds in the obvious way:

$$\begin{aligned}
&\text{if } \#(Disc_{f,\alpha} \cup \{g = 0\}, f(x)) = 1 \text{ then } g(\alpha) = 0,\\
&\text{if } \#(Disc_{f,\alpha} \cup \{g > 0\}, f(x)) = 1 \text{ then } g(\alpha) > 0,\\
&\text{if } \#(Disc_{f,\alpha} \cup \{g < 0\}, f(x)) = 1 \text{ then } g(\alpha) < 0.
\end{aligned}$$

Since $Disc(\alpha)$ specifies a single root, then only one of the above conditions is true. Therefore, we get:

$$\begin{aligned}
(g < 0)[\alpha/x] &\Leftrightarrow \#(Disc_{f,\alpha} \cup \{g < 0\}, f(x)) = 1\\
(g = 0)[\alpha/x] &\Leftrightarrow \#(Disc_{f,\alpha} \cup \{g = 0\}, f(x)) = 1\\
(g \leq 0)[\alpha/x] &\Leftrightarrow (g = 0)[\alpha/x] \vee (g < 0)[\alpha/x]\\
(g \neq 0)[\alpha/x] &\Leftrightarrow \#(Disc_{f,\alpha} \cup \{g = 0\}, f(x)) = 0
\end{aligned}$$

Putting everything above together and taking into account the remark from [7] about when can intervals be closed, half-open or open we can define a quantifier elimination procedure for univariate polynomials:

(Quantifer elimination for univariate polynomials)**.** *Let us partition the set* $\{1 \leq i \leq m\}$ *into two disjoints sets* $I_1 = \{i \mid 1 \leq i \leq m \wedge \tau_i \in \{\leq, =\}\}$ *and* $I_2 = \{i \mid 1 \leq i \leq m \wedge \tau_i \in \{<, \neq\}\}$*, then we can eliminate variable* x *in the following manner:*

$\exists x \varphi(x)$ *is equivalent to*

$$\bigvee_{\substack{i \in I_1\\ \alpha \in Zer(f_i(x))}} \big(\varphi[\alpha/x]\big) \vee \bigvee_{\substack{i \in I_2\\ \alpha \in Zer(f_i(x))}} \big(\varphi[\alpha + \epsilon/x]\big) \vee \varphi[-\infty/x]$$

where the substitutions $\varphi[\alpha/x]$, $\varphi[\alpha + \epsilon/x]$ or $\varphi[-\infty/x]$ are performed in the obvious way.

V. VTS using infinitesimals library

In order to implement full VTS on formulas involving arbitrary degree univariate polynomials over $\mathbb{K}$, we need a procedure for finding Thom encoding representations of roots for arbitrary degree polynomials. We could either use [2, Algorithm 10.14. Thom Encoding] or the library implemented in [4] for this. We have chosen the library. The operations we can use the library for are:

1) Finding roots of polynomials $f \in \mathbb{K}$, where $\mathbb{K}$ may involve computable transcendentals and infinitesimals. Roots are given as triples involving the defining polynomial, sign assignments on the set of derivatives (of the defining polynomial), and an interval containing the root. The triples guarantee that the defining polynomials have a single root in the interval with the specified sign assignment on derivatives.
2) Basic computations involving these roots.
3) Constructing $f \in \widetilde{\mathbb{K}}[x]$, where $\widetilde{\mathbb{K}}$ is an RCF possibly containing infinitesimals and transcendentals.

Assuming that formula φ contains polynomials in $\widetilde{\mathbb{K}}[x]$, with the help of the library we can therefore:

(i) Find the elements of test terms for formula φ as elements in $\widetilde{\mathbb{K}}(\epsilon)$ (where ϵ is infinitely smaller than all infinitesimals appearing in $\widetilde{\mathbb{K}}$) and compute the concrete value for all substitutions $f[t/x]$ (taking $-\infty$ as $-\frac{1}{\epsilon}$), thus making it trivial to check the satisfiability of φ.

(ii) Find the symbolic representation of the set of test terms in an extended term language, where each test term is either a root of some polynomial appearing φ (as per Thom's lemma roots are represented as sign assignments on derivatives[5]), root + ϵ or $-\infty$, all of these are viewed as improper expressions and are going to be substituted by proper ones (as described in Section III). Thus making it again possible to check the satisfiability of φ.

In practice (i) is more efficient than (ii). It is also more intuitive to understand and easier to implement. On the other hand, (ii) generalizes to the multivariate case more easily, which is why we have pursued it. The key difficulty lies in obtaining (a) an efficient sign determination algorithm involving multivariate polynomials, and (b) a method of finding efficient parametric Thom encodings for roots. The rest of the QE process will remain the same as in the univariate case. Our hope is that by developing an implementation of unbounded degree vts for univariate formulas, we can learn how to best approach the multivariate case.

[5]For univariate polynomials, it would be more efficient to represent roots directly as they are represented in the infinitesimals library, combining intervals and (truncated) Thom encodings. However, we are trying to only utilize techniques that will lift directly to the multivariate case, and we are still exploring how this more efficient representation can be best used in the multivariate case. The main place where this would have an effect is in the sign determination algorithm. As that could be modified to still return $\#(\sigma, p)$, the difference is not fundamental - one would have to use $sv(S, a, b)$ (where a and b are the of the interval corresponding to the root) instead of $sv(S, -\infty, \infty)$ in Tarski Queries. The improvement in efficiency comes from the fact that by using an interval to uniquely determine a root, one might be able to use smaller collections of assignments on derivatives.

VI. EXAMPLES

In this section, we present a small set of examples using our implementation of unbounded univariate vts. The implementation depends on the library presented in [4] and is written in Python[6].

We are going to present some short examples of the implementation. Our library encodes polynomials as lists where the *i*-th element corresponds to *i*-th power (as in [4]). For example, $a_0 + a_1 \cdot x + a_2 \cdot x^2 + a_3 \cdot x^3$ is encoded as

`[a0, a1, a2, a3]`.

In order to check formulas we are going to use the function `internal_vts(le, lt, eq, ne)` (located in the module (`uni_vts`) which determines the satisfiability of formula consisting of conjuncts of the type of atomic formulas previously mentioned[7]. The function implements the (ii) part of the previous chapter ((i) is implemented by `internal_ts` in `uni_ts` module). The parameters of the function (and `internal_ts`) correspond to lists of polynomials that are less or equal to zero (`le`), strictly less than zero (`lt`), equal to zero (`eq`) and not equal to zero (`ne`).

In the following, we'll check the satisfiability of atomic formulas involving polynomials $x - 2$, $(x - 1)(x - 2)$ and $x - 1$. First we'll define all of them.

```
f_1 = [-2, 1]
f_2 = [2, -3, 1]
f_3 = [-1, 1]
```

Then, we'll check the satisfiability of the formula $x - 2 < 0 \wedge (x - 1)(x - 2) = 0$, which is $True$.

```
print internal_vts([], [f_1], [f_2], [])
>> True
```

And the satisfiability of the formula $x-2 < 0 \wedge (x-1)(x-2) = 0 \wedge x - 1 \neq 0$, which is $False$.

```
print internal_vts([], [f_1], [f_2], [f_3])
>> False
```

We can demonstrate the use of infinitesimals[8] by using the previous example, but substituting ϵ_1 for 1 and ϵ_2 for 2.

First, we need to define the new polynomials $x - \epsilon_2$, $(x - \epsilon_1)(x-\epsilon_2)$ and $x-\epsilon_1$ and infinitesimals used in them (ϵ_1 and ϵ_2)

```
eps_1 = z3rcf.MkInfinitesimal('eps_1')
eps_2 = z3rcf.MkInfinitesimal('eps_2')
h_1 = [-eps_2, 1]
h_2 = [eps_1 * eps_2, -(eps_1 + eps_2), 1]
h_3 = [-eps_1, 1]
```

Our first formula becomes $x-\epsilon_2 < 0 \wedge (x-\epsilon_1)(x-\epsilon_2) = 0$.

```
print internal_vts([], [h_1], [h_2], [])
>> False
```

Inspecting the output of this call, we see that it is now `False`, it might be useful to expand on this. The z3rcf library construct infinitesimals such that the constructed infinitesimal is infinitely smaller the all previously constructed one. Therefore, $\epsilon_2 < \epsilon_1$. It is clear that $x-\epsilon_2 < 0$ will not contribute any satisfying test terms, $(x-\epsilon_1)(x-\epsilon_2) = 0$ contributes two: ϵ_1 and ϵ_2. ϵ_2 is obviously a dead end, and ϵ_1 doesn't suit either.

We can make this example more like the previous one by switching polynomial $(x - \epsilon_2)$ with $(x - \epsilon_1)$.

```
print internal_vts([], [h_3], [h_2], [])
>> True
```

We now see that the output is `True` as expected.

Applying the substitution in the second formula we get $x - \epsilon_1 < 0 \wedge (x - \epsilon_1)(x - \epsilon_2) = 0 \wedge x - \epsilon_2 \neq 0$, the order of the infinitesimals does not matter here and the output is $False$ as before.

```
print internal_vts([], [h_1], [h_2], [h_3])
>> False
```

For demonstrating that we are not bounded by degree[9] we are going to check the formula $(x-1)(x-2)(x-3)(x-4) < 0 \wedge (x+2)(x+1)(x-1)(x-2) < 0$.

```
g_1 = [24, -50, 35, -10, 1]
g_2 = [4, 0, -5, 0, 1]
print internal_vts([], [g_1, g_2], [], [])
>> True
```

It is easy to see that the formula is true if $x \in (1, 2)$ We can illustrate this a bit by showing that there is no solution if we add either constraint $x <= 1$ or $x >= 2$:

```
g_3 = [-1, 1]
g_4 = [2, -1]
print internal_vts([g_3], [g_1, g_2], [], [])
>> False
print internal_vts([g_4], [g_1, g_2], [], [])
>> False
```

By adding either constraint $x < 1 + \epsilon$ or $x > 2 - \epsilon$, we can show that there is a solution ϵ-close to those bounds.

```
eps = z3rcf.MkInfinitesimal('eps')
g_5 = [-1 - eps, 1]
g_6 = [-2 + eps, -1]
print internal_vts([], [g_5, g_1, g_2], [], [])
>> True
print internal_vts([], [g_6, g_1, g_2], [], [])
>> True
```

[6]The code is available as Github repository - https://github.com/grantpassmore/Real-Algebra-Playground/tree/master/vts. Then current version of the code implements the simpler Naive Sign Algorithm

[7]This form is more restrictive than $\wedge$, $\vee$-combination of atomic formulas, but since we can transform any formula to its disjunctive normal form and then decide the satisfiability of disjuncts using the function `internal_vts`, we can still decide the satisfiability of any formula.

[8]Transcendental and algebraic elements are handled analogously.

[9]We do not factor the polynomials, so none of them are viewed as "essentially quadratic."

REFERENCES

[1] Th. Sturm A. Dolzmann. Redlog-computer algebra meets computer logic. Number MIP-9603. 1996.

[2] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*. Springer-Verlag, 2006.

[3] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer Berlin Heidelberg, 2008.

[4] Leonardo de Moura and Grant Olney Passmore. Computation in real closed infinitesimal and transcendental extensions of the rationals. In *Conference on Automated Deduction (CADE)*, 2013.

[5] Rüdiger Loos and Volker Weispfenning. Applying linear quantifier elimination, 1993.

[6] Wolfram Research. Mathematica 9.0, 2014.

[7] V. Weispfenning. Quantifier elimination for real algebra - the quadratic case and beyond. *AAECC*, 8:85–101, 1993.

[8] Volker Weispfenning. Quantifier elimination for real algebra - the cubic case, 1994.