Evolutionary Virtual Term Substitution in a Quantifier Elimination System

Zak Tonks

University of Bath, Bath, Somerset, BA2 7AY, UK
z.p.tonks@bath.ac.uk

Abstract

Quantifier Elimination over real closed fields (QE) is a topic on the borderline of both the Satisfiability Checking and Symbolic Computation communities, where quantified statements of polynomial constraints may be relevant to solving a Satisfiability Modulo Theory (SMT) problem. Two well known algorithms for QE are Cylindrical Algebraic Decomposition (CAD) and Virtual Term Substitution (VTS). While many softwares implement either or both of these algorithms, they are rarely used together. This paper focuses on new proof of concept methods for incremental and decremental VTS, and briefly explores implications of such methods for a poly-algorithmic system using both VTS and CAD to perform QE.

1 Introduction

Quantifier Elimination over real closed fields (QE) concerns the problem of eliminating quantifiers $(\forall, \exists)$ from a boolean formula of polynomial constraints. Such a formula is often called a "Tarski formula":

Definition 1 *A "Tarski Formula" is a polynomial constraint, ie.* $f \, \rho \, 0$ *where* $f \in \mathbb{Q}[x_1, \ldots, x_n]$, $n \in \mathbb{N}$, *and* $\rho \in \{<, \leq, \neq, =\}$, *or a boolean formula of Tarski formulae, where allowable boolean operators may feature* $\wedge, \vee, \rightarrow, \veebar$ *(these are* and, or, implies, exclusive or*).*

A "Quantified Tarski Formula" is a Tarski formula where any subformula, or indeed the whole formula may feature quantified variables within.

Readers from each of the $\mathcal{SC}^2$ communities may be aware of a "prenex" Tarski formula, and indeed throughout we will assume input formulae to QE of the form:

$$Q_{n-m+1} x_{n-m+1} \ldots Q_n x_n \; \Phi(x_1, \ldots, x_n) \tag{1}$$

where $n \geq m \in \mathbb{N}$, and Φ is quantifier free, and features at most the boolean operators $\wedge$ or $\vee$. We note the existence of conversion of any Tarski formula to prenex form, and as such we can generally speak of such formulae. This includes pushing of $\neg$ to the leaves of a formula such that relational operators get inverted ($=$ to $\neq$, $<$ to $\geq$, etc.). Additionally, one should note that the restriction of ρ to the above operators is purely to reflect Maple's use of such operators, and one can include more or fewer via usage of $\vee$, but this is largely irrelevant. The goal of QE is to eliminate all quantifiers $Q_{n-m+1} x_{n-m+1} \ldots Q_n x_n$ from (1) to obtain the quantifier free equivalent, which may be any Tarski formula over $x_{m+1}, \ldots, x_n$, or $\top$ or $\bot$. It will necessarily be one of the latter two in

In: J. Abbott, A. Griggio (eds.): Proceedings of the 4th SC-square Workshop, Bern, Switzerland, 10th July 2019, published at http://ceur-ws.org

the case where $m = n$, in which case there are no free variables and the problem is described as "fully quantified".

QE problems over the reals arise from a range of sources - most notably algebraic geometry, such as [WDEB13]. Recently QE has also seen applications for economics [MBD+18]. While this paper is not a comprehensive covering of all algorithms available to tackle QE, Virtual Term Substitution (VTS) and Cylindrical Algebraic Decomposition (CAD) have proven popular. VTS appears in `Redlog` [DSK], `SyNRAC` [YA07], and `SMT-RAT` [CKJ+15]. `Redlog` and `Mathematica` also feature CAD implementations for QE. `QEPCAD B` [Bro03] features a partial CAD method [HC91] tailored specifically to solution of a QE problem, and packages such as `Regular Chains` [ACL+] and `ProjectionCAD` [EWBD14] offer implementations of CAD in Maple that can be applied to QE.

Performing QE is well known to be doubly exponential worst case time complexity in the number of variables [DH88]. We briefly discuss CAD, considering the overview of a poly-algorithmic system in Section 1.2. Research on CAD has mainly seemed to focus on the projection operator, instrinsic to the often used first stage "projection", which aims to form bases of polynomials in successively fewer variables. This may involve taking resultants, discriminants, and other operations in order to calculate polynomials which may ascertain key roots to solving the input QE problem. However taking iterative resultants is costly, exponentially increasing the degree and quantity of the polynomials in each basis. Hence, successive projection operators owing to, for example, McCallum [McC88] and Lazard [MPP19] have been less costly, and research here is ongoing. The amount of lifting one does is commensurate with the amount of projection one does, however significant improvements for lifting owe to Partial CAD [HC91], which is specifically an improvement for QE that aims to terminate QE early upon finding a satisfying cell.

Loosely, VTS appears to be a more concise method than CAD. In depth analysis of time complexity is lacking so far, however many recent improvements are due to Košta [Ko16] for the univariate case. Regardless, in contrast to CAD, VTS is entirely degree limited, with current technology again owing to [Ko16] offering an approach that can eliminate quantified variables appearing with degree up to 3 in Φ.

Further, we go in depth to explain the mechanisms of these methods, before exploring a system using interplay between them, and a discussion of incremental and decremental VTS relevant to such a system. Considering the terms "incremental" and "decremental" are used liberally in this paper, the author provides the following definitions.

Definition 2 *An* evolutionary *method refers to an "incremental" or "decremental" method (as is defined below).*

With respect to QE, an incremental *method refers to a method that takes the results of a previously computed QE problem, and uses previously computed data structures to recompute the same quantifier eliminations for (1) (with significant performance gain implied by reusing previous data), with some Tarski formula inserted within Φ.*

Similarly, a decremental *method refers to a method that will recompute (again, with implied performance gain over recomputation from scratch) the same quantifier eliminations for a prenex quantified Tarski formula with some subformula of Φ deleted.*

Note that most, if not all QE literature may use "incremental" to encompass what is referred to in this work as "evolutionary". While the former may be appropriate especially for QE literature, for the purposes of this work it is especially important to make the above three terms distinct.

1.1 Virtual Term Substitution

The method of Virtual Term Substitution revolves around attempting to substitute all possible meaningful points from intervals derived from polynomials in the input formula to eliminate an existential quantifier. Referring to (1), if one takes all constraints from Φ, ignoring boolean structure, then one may obtain a set of relations $f_1 \, \rho_1 \, 0, \ldots, f_s \, \rho_s \, 0$, $s \in \mathbb{N}$, $\rho \in \{<, =, \leq, \neq\}$. For each f_i, $i \in \{1, \ldots, s\}$, if f_i is of degree d in existentially quantified x, we can attribute at most $d + 1$ intervals of $\mathbb{R}$ owing to its real roots. The idea is to substitute a point from each of these intervals from each f_i into Φ "virtually", hence covering all points that may satisfy

Φ. If one finds k test points $t_1, \ldots, t_k$ from such intervals, then to be precise, VTS provides elimination of an existentially quantified variable x via the following:

$$\exists x\ \Phi = G(t_1) \wedge \Phi[x//t_1] \vee \cdots \vee G(t_k) \wedge \Phi[x//t_k] \tag{2}$$

where the disjunction arises as a result of these substitutions looking to find any such point that satisfies the existence of a point for x, and G is a map from test points to "guards" — Tarski formulae ensuring that the substitution of a test point is valid. $\Phi[x//t_i]$ is notation for "the virtual substution of t_i for x in Φ". For example, to be able to substitute the second distinct real root of a quadratic $ax^2 + bx + c$, one must have that that polynomial genuinely is quadratic, and it has a strictly positive discriminant, which gives the Tarski formula $a \neq 0 \wedge b^2 - 4ac > 0$. Similarly, substitution from a linear polynomial must ensure that its coefficient in x does not vanish.

We now require further intuition as to what "virtual" means. When speaking of the intervals associated to some f_i, to cover $\mathbb{R}$, each interval being closed at one end and open at the other will suffice. Additionally, the end intervals must feature $-\infty$ and ∞ respectively. Also, note the claim that VTS allows elimination from polynomials of degree at most 3. This implies that if we are to take a point owing exactly to the real root of, say, a quadratic, then at present this implies we are substituting a potentially irrational term. Doing so would take polynomials in Φ out of $\mathbb{Q}[x_1, \ldots, x_n]$ after substitution, which would make further VTS inapplicable. As such, *virtual* substitution owes to the following concepts/solutions:

1. usage of, and hence technology to handle substitution of test points including ϵ, to represent the canonical infinitesimal to allow us to substitute points from "just inside" an open interval, where presence of other variables disallows us from attributing a real number to such a point in an open interval;
2. usage of, and hence technology to handle substitution of ∞ / $-\infty$;
3. and technology to allow us to substitute test points involving what would otherwise be irrational roots, including such technologies to bypass even using algebraic numbers completely.

Said technology is well covered by Košta in [Ko16], and additionally offers excellent optimisations for the univariate case. Much of the author's work-in-progress implementation is based on this, and uses such methods offered.

A brief overview of handling of ϵ and ∞ test points is important for proper understanding of evolutionary VTS. Substitution of a test point involving $x = \pm\infty$ into some relation $g = 0$ will essentially result in a Tarski formula insisting that g is flat with respect to x, or in other words all the coefficients of g in x are 0. Meanwhile substitution of, say, $x = \infty$ into the relation $g < 0$ will result in a Tarski formula insisting that the sign of the leading coefficient in x is negative, or it is identically zero and we form a conjunction for that with the case for substitution for the reductum.

Meanwhile test points featuring ϵ will essentially look of the form $x = t \pm \epsilon$ for some t which is an expression corresponding to the root of some polynomial (here, up to a cubic). The idea is that ϵ is in some sense close to the real infinitesimal object that one encounters in analytical mathematics, and as such means "some small value". This is as we want to substitute a point "just to the left" or "just to the right" of some root expression (which defines the end point of an interval), but the possible existence of free variables and lack of root isolation at this stage means we do not know precisely how much. Either way, if one is to substitute $x = t \pm \epsilon$ into $g\,\rho\,0$, one must perform an "expansion" on $g\,\rho\,0$ that results in a Tarski formula that describes the behaviour of $g\,\rho\,0$ just less than/more than $x = t$, and then one can regularly virtually substitute $x = t$ into $g\,\rho\,0$. As an example, substitution of $x = t + \epsilon$ into $g = 0$ will have $g = 0$ expand as a formula insisting all coefficients of g in x are 0, and then we substitute $x = t$ into that formula.

The above are essentially summaries of the algorithms `expand-eps-at` and `vs-inf-at` respectively from [Ko16]. Regarding item 3, Košta was the first to suggest a good method of bypassing usage of algebraic numbers here. When substituting a root of some description from a polynomial f in x into a relation $g\,\rho\,0$, one takes the pseudoremainder of g by f to obtain a polynomial of degree strictly less than that of f. As a result, if f is of degree d, one need only know how to substitute roots of degree d into polynomials of degree $d - 1$

or less to continue. In fact, Košta does so via formulae schemes, which describe the resulting logic obtained when substituting a root of some description from a polynomial of some degree into any relation $g \rho\ 0$. If $h := prem(g, f, x)$ is of degree at most 0, then virtual substitution $(g \rho\ 0)[x // \text{RootOf}(f)]$ returns $h \rho\ 0$. Note that usage of pseudoremainder for substitution is justified by f being 0 at a root of f. When substituting into a formula with boolean structure, the virtual substitution process is defined recursively, where each constraint is replaced with its virtual substitution (which may expand into a non atomic formula).

The important thing to note here is that substitution of a test point into a genuine formula Φ will act upon every constraint appearing in Φ, and that constraints may expand as a result. At first glance, with an appropriate simplifier of Tarski formulae, this will obfuscate the original boolean structure of Φ over time.

By invention, VTS is a tool to eliminate one existential quantifier. However, a universally quantified formula can be treated by the following equivalence:

$$\forall x\ \Phi \equiv \neg \exists x\ \neg \Phi \tag{3}$$

which hence offers an existential quantifier for VTS to act upon. In the spirit of (2), here elimination of a universal quantifier gives us:

$$\forall x\ \Phi \equiv \neg(G(t_1) \wedge \neg \Phi[x // t_1]) \wedge \cdots \wedge \neg(G(t_k) \wedge \neg \Phi[x // t_k]) \tag{4}$$

where here $t_1, \ldots, t_k$ arise from polynomials in $\neg\Phi$.

We arrive at discussion of multivariate VTS, or in other words full QE via VTS, where applicable (as such, later we must speak of CAD and its applications in a QE system). As can be seen from (2) and (4), propagation of one step of VTS forms a disjunction or conjunction of the results of virtual substitution on the previous input formula, for elimination of an existential or universal quantifier respectively. At this point, it is important to note that having started with (1), and assuming $Q_n = \exists$, the first step of VTS obtains:

$$Q_{n-m+1} x_{n-m+1} \ldots Q_{n-1} x_{n-1}\ G(t_1) \wedge \Phi[x_n // t_1] \vee \cdots \vee G(t_k) \wedge \Phi[x_n // t_k] \tag{5}$$

at which point one can essentially distribute $Q_{n-1}x_{n-1}$ into each operand of the disjunction to obtain a choice of VTS amenable problems, assuming each or any are of a suitable degree. In fact, considering quantifiers of the same type commute, if $Q_j = \cdots = Q_{n-1}$ for some $j < n-1$ then there is a choice of which quantifier to distribute into the operands, but one must commit to such a choice for all such operands. Hence we obtain scope for choice-of-variable strategy, of which we acknowledge but do not discuss further here.

Two important concepts must be drawn attention to at this point. The first is that quantifier alternations can be troublesome here, as in [DH88], as they will progressively form conjunctions within disjunctions within conjunctions and so on. The latter is that multivariate VTS forms a canonical tree structure, in a not dissimilar manner to a tree for CAD corresponding to construction of stacks over intermediate cells. Figure 1 demonstrates the idea of such a VTS tree. The root node of this tree represents the input formula (1). Edges represent structural VTS test points, and nodes are the results of virtual substitutions (in general multiple successive substitutions, via following a path to the root). Each node is an operand of the implicit disjunction/conjunction formed at each level for VTS. If one is to create an object-oriented implementation really implied by such a tree structure, then node objects should at the very least store the formula obtained via the substitution that exists somewhere in the implicit conjunction/disjunction formed by iterated VTS. One can associate various other properties to such a node as well. Intermediate leaves of the tree represent unevaluated QE problems, via unevaluated quantifiers from $Q_{n-m+1}x_{n-m+1} \ldots Q_n x_n$, while genuine leaves of the tree must be those that hold the formula $\top$ or $\bot$. Meaningful leaves are those that are immediately meaningful to the quantifier elimination they came from - for example $\top$ short circuits a disjunction, and so is meaningful in terms of the elimination of an existential quantifier. The test point for such a meaningful leaf *is* the witness to the equivalence of the existentially quantified formula with $\top$, and as such somewhat a proof of it. Such a test point may require processing to be an equation featuring only a real number.

1.2 A Poly-algorithmic Quantifier Elimination System

While the main purpose of this paper is discussion of evolutionary VTS, the author moves to explain further motivations behind engaging with evolutionary VTS. Development of a new package `QuantifierElimination` for Maple in collaboration with Maplesoft is intended to be a poly-algorithmic solution to QE, which has aims including mitigation of QE's headline doubly exponential complexity in the number of variables [DH88], comprehensive strategy options, user friendliness, and rich output for users. This package is intended to be included in a future version of Maple.

One sees that usage of VTS in general produces further intermediate QE problems. There may be some opportunities for variable strategy such that one can continue using VTS. With respect to a full QE system, one can consider using CAD (in particular partial CAD) to solve any intermediate problems that arise, that are entirely of excessive degree for VTS to handle. Hence one starts with VTS, and then moves to CAD only when strictly necessary. This has several implications, some of which are briefly detailed here, but at least for the purposes of brevity of this work, full coverage of which the author leaves with proper discussion of his implementation of the package `QuantifierElimination` in Maple. At the very least, usage of CAD on problems with fewer variables should be far preferable to taking one CAD for the entire input, due to the high sensitivity of CAD on number of variables.

- It would be concerning to take a potentially exponential number of CADs owing to the worst case where one obtains a set of intermediate VTS nodes, none of which are amenable to further VTS due to excessive degree. So far this approach suggests to calculate individual partial CADs for each. Instead, an approach using incremental CAD to build a "Master CAD" seems more intelligent. Indeed, the efficiency of this approach is commensurate with the degree to which VTS nodes "share" their polynomials. The author hopes that in the average case, this can be observed to some degree.
- In most cases, any equational constraints from Φ will manifest in further VTS nodes, and so we can potentially use a reduced projection operator in CAD [McC99, McC01].
- Further, the aforementioned VTS "guards" will often feature equational constraints (especially owing to cases where one substitutes a point from the reductum of a higher degree polynomial), which offers extra opportunity to use reduced projection operators.
- One may even be able to complete QE without using CAD at all, if one is able to find a satisfying meaningful VTS node without completing the whole "QE tree", as it were.

2 Evolutionary VTS

We have seen that iterated usage of VTS to (modulo degree limitations) perform full QE forms a canonical tree structure, where each node holds a formula obtained by virtual substitution(s) on Φ. For example, a node at level $1 \leq i \leq m$ has the formula $\Phi[x_n//t_{n_{j_n}}]...[x_{n-i+1}//t_{n-i+1_{j_{n-i+1}}}]$. One notes that this is an unquantified Tarski formula that in general one would simplify, at the very least to remove trivial terms - anything evaluating to $\top$ or $\bot$. At best one can even make better deductions. Importantly, any such simplification risks losing the original boolean structure that originated with Φ, assuming it had any. We make precise a notion of atomic position that is necessary to understand the suggested process for evolutionary VTS.

Definition 3 *The* Atomic Position *of some subformula of a formula Ψ is its index when viewed as an operand of Ψ when Ψ has outer operator $\wedge$ or $\vee$, or the concatenation of the index of the operand it is contained in, say i, with its atomic position within operand i. If we wish to speak of the atomic position of Ψ itself, we can refer to this as atomic position 0, or just the empty position.*

This is well defined and meaningful as long as one considers that commutativity of operands within boolean operators is meaningless for computation, and as such operands within any one formula should remain fixed relative to their position upon input. Insertion or deletion of a formula may change the atomic positions of other subformulae, hence may only be meaningful with respect to the formulae associated with the last set of computations. One should note that individual virtual substitutions may have relations within Φ expand as formulae with multiple atoms, but one is able to keep a one to one correspondence between

subformulae of the formulae of VTS nodes and relations in the original Φ if one is careful with simplification as not to lose boolean structure. If one is to take an object-oriented approach to implementation, then one can store both the simplified and unsimplified formulae associated to any one result of VTS with each VTS tree node.

As motivation for the concept of atomic position for evolutionary VTS, one can take heed of [MBD+18], where it was seen that the Tarski framework can be used in the context of proving economics theorems. Most notably, one can first check consistency of the assumptions ($\exists \mathbf{v}\ A(\mathbf{v})$, where $A(\mathbf{v})$ is usually a conjunction of constraints, but not necessarily). Having done this and received $\top$, one can extend this result by conjunction with the hypotheses to check for an example. To continue with the examples from the economics framework, presently the below frameworks for insertion and deletion do not allow us to adapt a QE result from consistency of assumptions to proving the overall theorem ($\forall \mathbf{v}\ A(\mathbf{v}) \rightarrow H(\mathbf{v}) \equiv \forall \mathbf{v}\ \neg A(\mathbf{v}) \vee H(\mathbf{v})$), as this would require a framework to negate an existing VTS tree. This may look similar to what is presented here, and is an interesting further question in itself. Past this, the more general concept of atomic position for evolutionary methods may be useful to contrast and compare QE results for addition or removal of constraints in the presence of an analogy to UNSAT cores from the nearby satisfiability framework. Upon receiving a quantifier free answer for a QE problem, one may wish to experiment with strengthening or weakening assumptions (or hypotheses), which can be enabled by the generic evolutionary approach suggested here. The author knows of no such analogous concept for UNSAT cores for the QE framework so far, but hopes that storage of structurally "unsimplified" formulae as detailed below could enable such a concept.

Algorithm 2 describes the process for incremental VTS, with Algorithm 1 being an important auxillary. Note that of course insertion (and indeed deletion) of a subformula may change the atomic position of other subformulae within the formula, and as such atomic position is meaningful as of the last full QE computation (evolutionary or otherwise). One may question lines 16 & 18 of Algorithm 1 (and later similar lines in Algorithm 3), where one compares a boolean formula to $\top$ or $\bot$. Note that S may be a genuine Tarski formula (that is, may or may not be a true/false value) at these points, and as such may not be applicable to be directly parsed as a boolean value, so semantics means a comparison may really be necessary. While we do not describe the process of "propagation of further VTS" in this work, it essentially revolves around iterated univariate VTS, as somewhat shown in Figure 1. We note scope for strategy involving test point selection, and VTS node selection.

Note that the algorithms presented require a node to at least be, for example an object (in the context of object-oriented) that stores at least a simplified formula, unsimplified formula, (both of which the results of virtual substitutions from above for this node only) associated test point, children nodes, and parent nodes, all of which are implied by Figure 1. The test point associated to any one node is the edge directly above that node, ie. the last substituted test point resulting in this node. Unfinished tree leaves in this context are nodes that are present leaves in the context of the tree, but do not hold the formula $\top$ or $\bot$, and hence there is precedent for future quantifier elimination on the formula for that node. Meaningful tree leaves are those that imply termination of VTS somehow, via obtaining $\top$ in a disjunction arising from an existential quantifier, or $\bot$ for the conjunction from a universal quantifier. The above framework for insertion requires no resubstitution, considering unsimplified formulae associated with every node are the past results of iterated substitution themselves. One must be careful of the treatment of guards - the unsimplified formulae should purely be the results of iterated substitution on Φ, without the inclusion of guards, while the full simplification of the formula for any node should include the conjunction of the guard.

Theorem 1 *Usage of VTSInsert achieves the goal of incremental VTS.*

Proof. Calling VTSNodeInsert inserts ψ at the correct location on every existing node in the VTS tree, assuming each node stores an unsimplified formula. Note that the recursive call in VTSNodeInsert uses ψ after virtual substitution of any test point used at the current level, and as such each node gets ψ with appropriate substitutions for its level. Note that each node should insert ψ modulo substitutions retaining boolean structure of ψ, as well, to retain boolean structure such that atomic position in such formulae remains well defined. After a call to this at the root, we traverse the entire tree, and so the tree is at least *semantically* correct, however may not be *complete* in the sense that the implicit disjunction/conjunction formed by VTS as of this point may not evaluate to $\top/\bot$. This is equivalent to L being empty, as L should be a set of (new) meaningful leaves, the existence of any of which implies termination of VTS. As such, propagate VTS on any nodes existing in U (a

Algorithm 1: VTSNodeInsert(N,α,ψ,U,L)

Data: A node of the VTS tree, N, an atomic position α, a Tarski formula ψ, a set of unfinished tree leaves U, and a set of meaningful VTS leaves L

Result: Modifies N by insertion of ψ at atomic position α into the formula associated with N.

if *N is the root VTS node* **then**
- insert ψ at atomic position α into the structurally unsimplified formula associated to N (which is Φ);
- add the set difference of test points in x_m for ψ with the set of test points used to obtain the children of N (if they exist) to the set of test points that can be used to propagate VTS on N further;
- if this set is nonempty, add N to U;
- **for** *all C, children nodes of N* **do**
  - VTSNodeInsert(C,α,ψ,U,L);

else
- let T be the VTS test point of this node. Let $i > 0$ be the level of N;
- **if** $Q_{m-i+1} = \exists$ **then**
  - $\tau \leftarrow \psi[x_{m-i+1}//T]$;
  - insert τ at atomic position α into the structurally unsimplified formula associated to N;
- **else**
  - $\tau \leftarrow \neg((\neg\psi)[x_{m-i+1}//T])$;
  - insert τ at atomic position α into the structurally unsimplified formula associated to N;
- let S be the full simplification of the formula associated with N after insertion;
- **if** $Q_{m-i+1} = \exists$ *and* $S = \top$ *or* $Q_{m-i+1} = \forall$ *and* $S = \bot$ **then**
  - N is a meaningful leaf - add N to L, delete its children nodes from the tree;
- **else if** $S = \top$ *or* $S = \bot$ **then**
  - N a non meaningful leaf - delete the children nodes of N from the VTS tree;
- **else**
  - add the set difference of test points in x_{m-i} for τ with the set of test points used to obtain the children of N (if they exist) to the set of test points that can be used to propagate VTS on N further;
  - if this set is nonempty, or undefined, add N to U;
  - **for** *all C, children nodes of N* **do**
    - VTSNodeInsert(C,α,τ,U,L);

Algorithm 2: VTSInsert(R,α,ψ)

Data: The VTS root node R (which is associated to the formula Φ), an atomic position α, and a Tarski formula ψ

Result: Produces a set of meaningful VTS leaves for the QE problem with ψ inserted at atomic position α in Φ

$U \leftarrow \emptyset$;
$L \leftarrow \emptyset$;
VTSNodeInsert(R,α,ψ,U,L);
while *L is empty* **do**
- propagate VTS as far as possible on any nodes in U, including using additional test points attributed to them from VTSNodeInsert, adding meaningful tree leaves to L;

return L

Algorithm 3: VTSNodeDelete(N,α,U,L,*prune*)

Data: A node of the VTS tree, N, an atomic position α, a set of unfinished tree leaves U, a set of meaningful VTS leaves L, and *prune*: a boolean flag dictating whether the user wishes to "prune" the VTS tree

Result: Modifies N by deletion of the subformula at atomic position α from the formula associated with N.

```
delete the subformula at atomic position α from the structurally unsimplified formula associated with N.
  Let D be the formula that is deleted;
let S be the full simplification of the formula for N after deletion;
let i > 0 be the level of N;
if Q_{m-i} = ∃ and S = ⊤ or Q_{m-i} = ∀ and S = ⊥ then
    N is a meaningful leaf - add N to L, delete its children nodes from the tree;
else if S = ⊤ or S = ⊥ then
    N a non meaningful leaf - delete the child nodes of N from the VTS tree;
else
    if prune then
        E_D ← set of test points in x_{m-i} for D;
        E_S ← set of test points in x_{m-i} for S;
        let E be the set of future test points for propagation of VTS on N (if it has been defined thus far).;
        E ← E \ (E_D \ E_S);
        for all C, child nodes of N do
            Let T be the test point of C;
            if T ∈ E_D \ E_S then
                discard C from the VTS tree;
    if the set of future test points for propagation of VTS on N is calculated already and nonempty add
      N to U;
    for all C, child nodes of N do
        VTSNodeDelete(C,α,U,L,prune);
```

set of unfinished VTS tree leaves) to find a "complete" tree for the QE problem with ψ inserted in Φ at atomic position α. □

The case for deletion is similar, albeit this corresponds to removal of information rather than addition, and as such there is somewhat less to do. Trivially, no resubstitution is required. Algorithms 3 and 4 describe the case for deletion.

Theorem 2 *Usage of VTSDelete achieves the goal of decremental VTS.*

Proof. Calling VTSNodeDelete deletes the subformula at α from a node in the VTS tree, assuming each node stores an unsimplified formula. After a call to this at the root node, the entire tree is traversed due to recursion on children nodes, and so the tree is semantically correct as per the input formula with deletion. However the treee may not be *complete* in the sense that the implicit disjunction/conjunction formed by VTS may not evaluate to $\top/\bot$. This is equivalent to L being empty, as L should be a set of (new) meaningful leaves, the existence of any of which implies termination of VTS. As such, propagate VTS on any nodes existing in U (a set of unfinished VTS tree leaves) to find a "complete" tree for the QE problem having done deletion. □

The notion of "pruning" the VTS tree refers to deleting subtrees of the VTS tree arising from test points (ie. tree edges) that can no longer be attributed to *any* constraint in the formula attributed to the above node. Note that pruning the tree is in fact a choice - usage of too many test points at any one level of VTS is never incorrect. In some sense one should see VTS as tackling existential quantifiers by looking for valid examples, and likewise universal quantifiers by looking for valid counterexamples. As such, otherwise superflous test points only correspond to a surplus of valid examples or counterexamples alike.

That being said, "a surplus" obviously implies that carrying such redundant nodes around is somewhat inefficient, depending on what happens further, such as further evolutionary operations, or the nodes being

Algorithm 4: VTSDelete(R,α,*prune*)

Data: The VTS root node R (which is associated to the formula Φ), an atomic position α, and a boolean flag *prune*

Result: Produces a set of meaningful VTS leaves for the QE problem with the subformula at atomic position α deleted from Φ

```
U ← ∅;
L ← ∅;
VTSNodeDelete(R,α,U,L,prune);
while L is empty do
    propagate VTS on an unfinished VTS node in U that has further test points to use, according to the
     test points attributed to them as of tree traversal, adding meaningful tree leaves to L, and
     unfinished VTS nodes to U;
return L;
```

processed to deduce QE output. On the other hand, as can be seen in Algorithm 3, this pruning requires some recalculation, which is never particularly in the spirit of evolutionary operations. In particular, calculation of VTS structural test point sets may incur some non-trivial costs in terms of attempts at factorization on the polynomials involved, and/or deductions about signs of coefficients of polynomials. As VTS test point sets are not comprehensively covered here, the author refers to [Ko16] for such explanations.

All algorithms presented here are for pure VTS, and make no reference to CAD as per the aforementioned poly-algorithmic approach that the author intends to take for the new implementation of QE. However one should note that the interfacing between VTS and CAD for a full evolutionary approach for QE appears to look canonical here, with CAD results following from VTS nodes of excessive degree receiving polynomials to potentially add/delete from the CAD for incremental and decremental QE respectively. Considering a CAD that is sign invariant for a set F of polynomials is also sign invariant for any subset of F, CAD incrementality is strictly the least that would be needed for both the "master CAD" approach of the poly-algorithmic system, or the continuation of evolutionary methods from VTS to CAD (where modification of high degree VTS nodes implies modification of the ensuing master CAD).

2.1 Comparison to Other Work

We acknowledge work for `SMT-RAT` from [CKJ+15] that covers what would be referred to here as incremental VTS. Superficially, these algorithms are intended to be included in an overall package for QE in Maple, and hence implemented in Maple and not directly open source. Otherwise, there are a few main differences. One is that the presentation in terms of insertion and deletion of subformulae here is more general, with the concept of atomic position allowing for insertion or deletion in nested subformulae. In particular the starting formula need not be a flat conjunction of constraints. In theory, insertion of subformulae via conjunction or disjunction with the entire formula would require no further storage of "unsimplified" formulae by virtue of no need of atomic position. Secondly, the above speaks of the multivariate case in greater detail, suggesting how an insertion or deletion should modify the entire VTS tree. Lastly, we present the new case for deletion, with the additional nuance of "pruning" the VTS tree.

For the purposes of simplifying this work, in terms of insertion we have merely suggested how to insert within existing boolean operators. In general, there is precedent to form a new disjunction or conjunction arbitrarily within the existing formula, which makes the concept of atomic position a little more complicated. The case of forming a disjunction or conjunction of the entire existing input with a new formula is a reasonably trivial extension to this framework, which almost coincides with the case from [CKJ+15].

3 Conclusion and Future Work

While the author's new QE package is in relatively early days, requiring extensive testing and benchmarking, the author hopes for it to be a fully comprehensive system. Many of the implications suggested by the system must be investigated in terms of meaningful QE examples, of which the author hopes to engage with further. This

includes bringing attention to further real world examples, which the author notes the community is especially fond of over artificially invented ones. The poly-algorithmic approach of the author's new package requires an implementation of incremental CAD, which the author is yet to traverse. Incremental CAD is also relevant towards a full evolutionary QE system extending upon the evolutionary approach for VTS presented here. Another problem that is of interest to the author for the purposes of this implementation is Tarski formula simplification, where an effective simplifier is likely to boost performance of VTS for the purposes of deducing termination.

The author hopes for the presented evolutionary techniques to be useful in the context of QE for SMT. These evolutionary techniques will be included as part of the `QuantifierElimination` package for Maple. A general framework that works for insertion or deletion of formulae at an arbitrary position has been suggested, requiring additional storage but providing scope for a user to investigate the nuances of a quantifier elimination problem further, else providing the usual requirements of addition of constraints for the multivariate case. Examples and benchmarking will be provided in the future using this framework, especially in the case of economics examples, where there is a canonical extension for past QE results.

References

[ACL+] P. Alvandi, C. Chen, F. Lemaire, M.M. Maza, and Y. Xie. The RegularChains Library.

[Bro03] Christopher W. Brown. QEPCAD B: A Program for Computing with Semi-algebraic Sets Using CADs. *SIGSAM Bull.*, 37(4):97–108, December 2003.

[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 Marijn Heule and Sean Weaver, editors, *Theory and Applications of Satisfiability Testing – SAT 2015*, pages 360–368, Cham, 2015. Springer International Publishing.

[DH88] J.H. Davenport and J. Heintz. Real Quantifier Elimination is Doubly Exponential. *Journal of Symbolic Computation*, 5(1):29–35, 1988.

[DSK] A. Dolzmann, T. Sturm, and M. Košta. Redlog - computing with logic. Accessed: 2019-04-19.

[EWBD14] M. England, D.J. Wilson, R. Bradford, and J.H. Davenport. Using the Regular Chains Library to build Cylindrical Algebraic Decompositions by projecting and lifting. In H. Hong and C. Yap, editors, *Mathematical Software – ICMS 2014*, pages 458–465. Springer Berlin Heidelberg, 2014.

[HC91] H. Hong and G.E. Collins. Partial Cylindrical Algebraic Decomposition for quantifier elimination. *Journal of Symbolic Computation*, pages 299–328, 1991.

[Ko16] M. Košta. *New Concepts for Real Quantifier Elimination by Virtual Substitution.* PhD thesis, Universität des Saarlandes, 2016.

[MBD+18] C.B. Mulligan, R. Bradford, J.H. Davenport, M. England, and Z. Tonks. Non-linear Real Arithmetic Benchmarks derived from Automated Reasoning in Economics. Proc. SC^2 workshop FLoC 2018, 2018.

[McC88] Scott McCallum. An improved projection operation for Cylindrical Algebraic Decomposition of three-dimensional space. *Journal of Symbolic Computation*, 5(1):141 – 161, 1988.

[McC99] S. McCallum. On projection in CAD-based quantifier elimination with equational constraint. In *Proceedings ISSAC 1999*, pages 145–149, 1999.

[McC01] S. McCallum. On propagation of equational constraints in CAD-based quantifier elimination. In *Proceedings ISSAC 2001*, pages 223–231, 2001.

[MPP19] Scott McCallum, Adam Parusiski, and Laurentiu Paunescu. Validity proof of Lazard's method for CAD construction. *Journal of Symbolic Computation*, 92:52 – 69, 2019.

[WDEB13] D. Wilson, J. H. Davenport, M. England, and R. Bradford. A "Piano Movers" Problem Reformulated. In *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 53–60, Sep. 2013.

[YA07] Hitoshi Yanami and Hirokazu Anai. SyNRAC: A Maple Toolbox for Solving Real Algebraic Constraints. *ACM Commun. Comput. Algebra*, 41(3):112–113, September 2007.

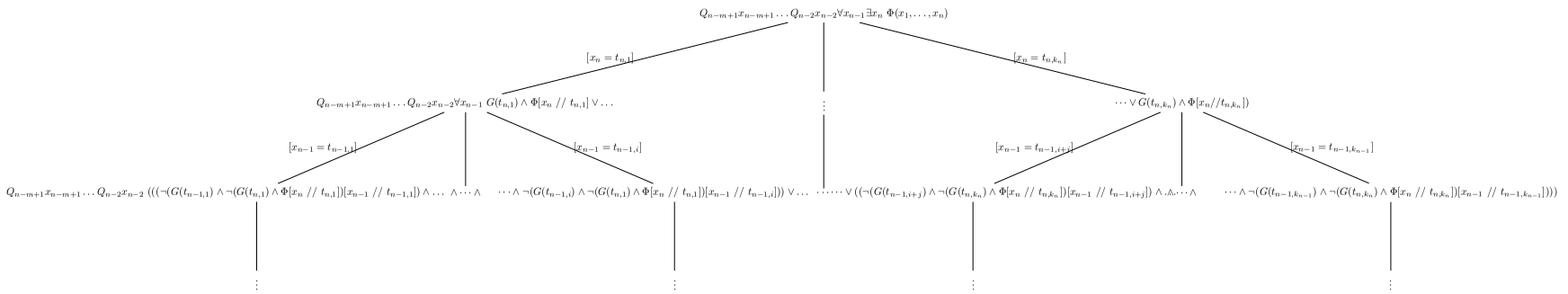

Figure 1: A demonstration of a VTS tree, where the last two quantifiers are $\forall$, $\exists$ for the purposes of demonstration of the conjunction nested within a disjunction formed as a result, that may be sufficient to deduce the quantifier free equivalent of (1).