IMPLEMENTING VIRTUAL SUBSTITUTION IN KEYMAERAX

MANUEL FERNÁNDEZ

Abstract. Virtual Substitution is an existential quantifier elimination method for statements of First Order Logic over the Reals ($\mathrm{FOL}_{\mathbb{R}}$) satisfying certain constraints. Although not a comprehensive QE tool, it's relative simplicity and efficiency compared to other QE techniques make it a useful method. Moreover, Virtual Substitution can be very useful for proving results about CPS models that involve polynomials with low degree. In this paper, we discuss a partial implementation of Virtual Substitution in KeymaeraX based on certain assumptions we make about the input formula. Although not a complete implementation of Virtual Substitution, the implementation is designed to apply virtual substitution to $\mathrm{FOL}_{\mathbb{R}}$ statements made up of existential quantifiers and comparison formulas involving linear polynomials. In addition, the implementation can apply virtual substitution to certain $\mathrm{FOL}_{\mathbb{R}}$ statements containing quadratics.

1. Introduction

A Cyber-Physical System (CPS) is any electrical-mechanical system which utilizes discrete and continuous dynamics. From modern airplanes to personal computers, CPSs can be found in almost all aspects of every day life. Because of their importance, being able to prove safety and efficiency conditions about a CPS is of paramount importance. Because it's impossible to test a CPS under every scenario for which you'd hope for it to be safe and efficient, the standard approach is construct a model, usually simpler, of the CPS and prove results about the model. KeymaeraX, a theorem prover for Hybrid Systems (HS) using Differential Dynamic Logic ($d\mathcal{L}$), allows you to model a CPS as a Hybrid Program (HP) using the language of differential dynamic logic and prove safety and efficiency properties of the HP. What's important to note is that many statement of $d\mathcal{L}$ which are provable are reducible to statements of Real arithmetic over the Reals. The reduced statement quantifies a set of free variables appearing inside the statement and depicts some relation between the the free variables and Real numbers. To evaluate the model, you have to further reduce the statement so as to make it quantifier-free, a process known as Quantifier Elimination (QE). The resulting statement (when normalized) is now made of comparison formula of the form $a \sim b$, where $a, b \in \mathbb{R}$ and $\sim \in \{<, >, =\}$, each of which can be evaluated quickly. Although it's an open problem to determine the class of all statements of Real arithmetic which are decidable, Alfred Tarski proved in 1951 that the language of First Order Logic over the Reals ($\mathrm{FOL}_{\mathbb{R}}$) was decidable [3] . In this

manuelf@andrew.cmu.edu School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, 15213, USA.

language, all statements can contain universal or existential quantifiers, and all comparison formula are between polynomials with Rational Coefficients. As one can imagine, this is a powerful result in verifying models of CPSs. If a statement about the efficiency or safety of a model reduces to an $\mathrm{FOL}_{\mathbb{R}}$ statement, then we can either reduce that statement to a quantifier free statement. Surprisingly, KeymaeraX does not have its QE implementation, instead outsourcing the task to Mathematica. Since KeymaeraX assumes the QE result from Mathematica to be valid, KeymaeraX's proving power is only as strong as Mathematica's. This motivated us to implement some form of QE in KeymaeraX.

However, practical Quantifier Elimination can be tricky. The original algorithm due to Tarski for QE had unbounded complexity,(so for an $\mathrm{FOL}_{\mathbb{R}}$ with polynomials in $\mathbb{R}^n$, the runtime is atleast $2^{2^{\cdot^{\cdot^{n}}}}$ for any arbitrary long tower of 2s) [3] . More modern approaches to QE include using Cylindrical Algebraic Decomposition (CAD) and Semi-Definite Programming, but involve a significant amount of math and are difficult to implement. Instead, we consider a QE technique known as Virtual Substitution (VS). VS is a QE technique designed to remove existential quantifiers from statements of $\mathrm{FOL}_{\mathbb{R}}$ where the variable being quantified over appear only in linear polynomials or in certain types of quadratic polynomials. We present an implementation of VS for KeymaeraX which can apply QE elimination for statement involving linear equations.

2. Virtual Substitution Basics

The following is a description of certain result found in [1] and [2]

2.1. **Motivation.** To motivate the idea of VS, we'll consider the following scenario: Suppose we are given an $\mathrm{FOL}_{\mathbb{R}}$ statement F of the form $\exists x\ S$, where $S := F_1 \oplus \cdots \oplus F_m$ where $\oplus \in \{\wedge, \vee\}$ and each F_i is a comparison formula of the form $ax^2 + bx + c \sim dx^2 + ex + f$, where a, b, c, d, e, f are terms not including the variable x and $\sim \in \{>, <, =\}$. Since there are no quantifiers in S, we'll assume that each a, b, c, d, e, f reduces to either a rational number or an $_{\mathbb{R}}$ term not containing x. Then the F_i is equivalent to the following statement: $(a-d)x^2+(b-e)x+(c-f) \sim 0$. Since one can determine the existence of roots for such a polynomial in x and compute then using the linear formula and quadratic formula, one can quickly determine the domain of X_i for which F_i is potentially true. Thus assuming we have the domains $X_1, X_2, \ldots\ldots, X_m$, the domain X' for which S is valid on is a subset of $Y = X_1 \cup \cdots \cup X_m$ (based on the and-or connectors of the statement). Now computing X' is equivalent to determining whether or not S is satisfiable, so instead one can simply test every value y in Y and solve the resulting statement S^x_y, returning true if it's valid or false otherwise. Of course it's possible that X' is infinite, so it's not possible for a program to test each element in Y. However note that

$$X_i \equiv \{\emptyset, \{a\}, [a, b], (a, b), (a, \infty), (-\infty, b), \mathbb{R} \setminus [a, b], \mathbb{R} \setminus (a, b), \mathbb{R}\}$$

where $\equiv$ represents the form of X_i. Therefore one need only test ∂X_i for each X_i, where we now view $X_i \subseteq \bar{\mathbb{R}}_\epsilon$. Where $\bar{\mathbb{R}}_\epsilon$ represents $\mathbb{R}$ equipped with positive and negative infinity and infinitesimals. One can see this by noting that $X_i \cap X_j$

contains at least 1 boundary point from either X_i or X_j. Since there there are at most 4 boundary points per X_i, we only need to check at most 4 values for x per comparison formula, and so at most $4m$ values for x. Of course, one should note that the resulting substitutions would no longer be statements of $\text{FOL}_{\mathbb{R}}$, as the statement might include square roots, infinities, and infinitesimals.

2.2. **Base cases.** To see how the ∂X_is looks, consider a sub-formula F_i. Then upon normalization F_i has the following form:

$$F_i \equiv ax^2 + bx + c = 0 \mid ax^2 + bx + c < 0 \mid ax^2 + bx + c > 0$$

where a, b, c are x-free. Thus we have that

$$\partial X_i \subset$$

$$\{0, \pm\epsilon, \frac{-c}{b}, \frac{-c}{b}\pm\epsilon, \frac{-b+\sqrt{b^2-4ac}}{2a}, \frac{-b-\sqrt{b^2-4ac}}{2a}, \frac{-b+\sqrt{b^2-4ac}}{2a}\pm\epsilon, \frac{-b-\sqrt{b^2-4ac}}{2a}\pm\epsilon, \pm\infty\}$$

Note that some of the elements in the set are not well defined on the normalized polynomial, although this isn't an issue as such elements aren't boundary points for that particular polynomial.

Now suppose we compute ∂X. Note that some of the elements in ∂X cannot be simply substituted into F because some of them contain square roots, epsilons, and infinities. Let's first tackle the square root case. Suppose we have a well-defined formula of the form

$$\frac{a + b\sqrt{c}}{d} \sim 0. \tag{1}$$

where a, b, c, d are square root free. We assume $d > 0$ (as otherwise we can pass the negative sign to a and b). It follows that this formula is equivalent to

$$a^2 - b^2c \hat{\sim} 0 \tag{2}$$

where $\hat{\sim}$ is determined by the number of times we multiply the formula by -1. This in conjunction with the following facts:

$$\frac{a + b\sqrt{c}}{d} \cdot \frac{a' + b'\sqrt{c}}{d'} = \frac{aa' + bb'c + (ab' + a'b)\sqrt{c}}{d'd'}, \tag{3}$$

$$\frac{a + b\sqrt{c}}{d} + \frac{a' + b'\sqrt{c}}{d'} = \frac{ad' + a'd + (bd' + b'd)\sqrt{c}}{d'd} \tag{4}$$

means that a root of a quadratic equation can be plugged into any of the F_i's, which using simplification facts (3),(4) can reduce the formula to the form specified in (1), which is equivalent to the square-root free form in (2).
Next we tackle the epsilon case. Suppose an element $p \in \partial X$ is of the form $s \pm \epsilon$. If we plug it into one of the F_i, the substitution is equivalent to

$$M\epsilon^2 + N\epsilon + P \sim 0$$

where M, N, P are ϵ - free. If $\sim := =$, then the formula reduces to $P = 0 \wedge M = 0 \wedge N = 0$, where all 3 equations are equivalent to square-root free comparison formulas by the previous case. If $\sim := <$, the formula is equivalent to $P < 0 \vee (P = 0 \wedge N < 0) \vee (P = 0 \wedge N = 0 \wedge M < 0)$, where again all the comparison formula are equivalent to square-root free ones. Here we rely on the fact that if a, b are epsilon-free, infinity-free and $a, b \neq 0$, $m < n \implies sign(a\epsilon^m + b\epsilon^n) = sign(a\epsilon^m)$. For $\sim := >$ the formula is equivalent to $P > 0 \vee (P = 0 \wedge N > 0) \vee (P = 0 \wedge N = 0 \wedge M > 0)$.
Finally we have the infinity case. If we substitute $\pm\infty$ into one of the F_i, the substitution is equivalent to

$$M(\pm\infty)^2 + N(\pm\infty) + P \sim 0.$$

If $\sim := =$, the the formula reduces to $(P = 0 \wedge M = 0 \wedge N = 0)$. If $\sim := <$, then the formula reduces to $M < 0 \vee (M = 0 \wedge sign(\pm\infty) < 0) \vee (M = 0 \wedge N = 0 \wedge P < 0)$. If $\sim := >$, Then the formula reduces to $M > 0 \vee (M = 0 \wedge sign(\pm\infty)N > 0) \vee (M = 0 \wedge N = 0 \wedge P < 0)$ Note that $sign(\pm\infty)$ is simply 1 or -1.

So indeed, if given an $\text{FOL}_{\mathbb{R}}$ statement of the form $\exists x F_1 \oplus \cdots \oplus F_m$, where F_i are comparisons formula which are at most quadratic in x, we can compute $T = \partial X_1 \cup \cdots\cdots \partial X_m$, where $|T| \leq 4m$ and reduce the $\text{FOL}_{\mathbb{R}}$ statement to

$$(F_1 \oplus \cdots \oplus F_m)^x_{T_1} \vee \cdots \vee (F_1 \oplus \cdots \oplus F_m)^x_{T_{|T|}} \equiv (F_{1,T_1} \oplus \cdots \oplus F_{m,T_1}) \vee \cdots \vee (F_{1,T_{|T|}} \oplus \cdots \oplus F_{m,T_{|T|}})$$

where each $F_{i,j}$ is a comparison formula of $\text{FOL}_{\mathbb{R}}$.

2.3. **Extending the base cases.**

Of course, statements of the form $\exists x F_1 \oplus \cdots \oplus F_m$ are rather limited. We would like to be able to consider cases beyond this form. To that effect, we note the following:

(1) Any statement of $\text{FOL}_{\mathbb{R}}$ is equivalent to a statement of the form
$$(\exists x_1 \cdots \exists x_k)(F_1 \oplus \cdots \oplus F_m)$$
where F_i is a comparison formula with comparer $\sim \in \{<, >, =\}$.

(2) Polynomials over $\mathbb{R}$ with degree at least 5 can possibly have roots which are not representable with nth roots and basic arithmetic ops.

This tells us that 1) It suffices for us to consider statements of the specified in (1) and 2) it's impossible to give an elementary closed form expression for the boundary of most polynomials of degree ≥ 5. In addition, trying to compute explicit roots for degree 3 and 4 involve grotesque algebra. Therefore we'll restrict ourselves to trying to apply Virtual Substitution to $\text{FOL}_{\mathbb{R}}$ statements as specified in (1) whose comparison formula are at most quadratic.

2.4. **The Linear case.**

Let's now consider the case where our $\text{FOL}_{\mathbb{R}}$ statement F is of the form

$$(\exists x_1 \cdots \exists x_k)(F_1 \oplus \cdots \oplus F_m)$$

where as before each F_i is a comparison formula with comparison $\sim \in \{=, >, <\}$. However we now assume that when normalized, we have

$$F_i \equiv a_{1,i}x_1 + a_{2,i}x_2 + \cdots a_{k,i}x_k + b_k \ \sim \ 0.$$

where $a_{j,i}$ and b_k are $x_1, x_2, \cdots, x_k$- free. Because the LHS is a sum of linear polynomials. We have the following phenomenon. Suppose we only consider the subformula $\exists x_k(F_1 \oplus \cdots \oplus m)$. Let $\partial X_{i,k}$ denote the boundary for which $\exists x_k F_i$ is true. Then

$$\partial X_{i,k} \subset \{-\frac{a_{1,i}x_1 + \cdots + a_{k-1,i}x_{k-1} + b_k}{a_{k,i}}, -\frac{a_{1,i}x_1 + \cdots + a_{k-1,i}x_{k-1} + b_k}{a_{k,i}} \pm \epsilon, 0, \pm\infty\}$$

Now take $T_k = \partial X_{1,k} \cup \cdots \cup X_{m,k}$ and let $p \in T_k$. If $y = 0, \pm\infty$ then we can simply virtual substitute p and convert $F_p^{x_k}$ into an $\text{FOL}_{\mathbb{R}}$ statement as we did in the base case. However, if p is of the other two forms specified earlier, then substituting it into a given F_j gives us:

$$F_j := M_1x_1 + \cdots + M_{k-1}x_{k-1} \sim 0 \mid M_1x_1 + \cdots + M_{k-1}x_{k-1} + b_j\epsilon \sim 0$$

where all the M_is are $x_1, \cdots x_k, \epsilon$ -free. But the first form is exactly an $\text{FOL}_{\mathbb{R}}$ formula, and the second form is reducible to an $\text{FOL}_{\mathbb{R}}$ formula by the bases cases! If we apply VS for the first $k-1$ quantifiers, we're left with a reduced formula F' which is made up of $\wedge, \vee$ connectors and comparison formula F_i's, where F_i' is of the form

$$F_i' \equiv A_ix_1 + B_i + C_i(\pm\infty) + D_i\epsilon$$

where A_i is $x_2, \cdots, x_k, \epsilon, \infty$ free, B_i is $x_1, \cdots, x_k, \epsilon, \infty$ free, C_i is $x_1, \cdots, x_k, \epsilon$ free and D_i is $x_1, \cdots, x_k, \infty$ free, and either C_i or D_i is equivalent to 0. Again applying VS on more time produces a reduced statement which is quantifier free and x_1 free and each F_i' yields a quantifier-free $\text{FOLR}_{\mathbb{R}}$ formula F''. One can easily evaluate such statements.

2.5. **Problems with General Quadratic Cases.** Beyond the linear case, cases where VS works are limited. The problem largely comes from the form of the elements in ∂X. Aside from $0, \pm\epsilon, \pm\infty$, all other types of elements in ∂X are fractions, where the denominator may or may not be a quantified variable, $\pm\epsilon$ or $\pm\infty$. Due to the immense casework and formula processing that would have been necessary to consider more general cases. We did not consider additional techniques for other cases.

3. Implementing VS for KeymaeraX

3.1. **Approach.** For our implementation of VS in KeymaeraX, we chose a model which wasn't based on efficiency.

The substitution process is outlined below

(1) Take in a $\text{FLO}_{\mathbb{R}}$ statement with a existential quantifier $\exists$ in some variable x.
(2) Recurse through through each portion of the statement until we reach a statement comparison formula of the form $A \sim B$
(3) Convert A and B into A' and B' where A', B' do not contain Minus constructors. Then reduce the comparison formula to the form $A' + (-(B')) \sim 0$
(4) Normalize the term $A' + (-(-B'))$ into the canonical form G, where

$$G \equiv ((a_n \cdot x^n) + ((a_{n-1} \cdot x^{n-1}) + (\cdots + (a_0 \cdot x^0))) \cdots)$$

(5) Once all comparison formulas are normalized, we pass the normalize statement into a function that gathers replacement candidates and restriction from all the comparison. It traverses the formula recursively as well. Suppose that the function reaches a comparison formula:
  (a) First we check if the degree of the polynomial w.r.t x is > 3. If so we return the restriction-substitute pair $(False, Nil)$ If not we case on the polynomial and the comparer:
  (b) $((a_2 \cdot x^2) + ((a_1 \cdot x^1) + (a_0 \cdot x^0)))$
    (i) $< \implies [([a_2 = 0 \wedge a_1 = 0 \wedge a_0 < 0], [-\infty, \infty]), ([a_2 = 0 \wedge (a_1 < 0 \vee a_1 > 0)],), [-\infty, -(\frac{a_0}{a_1}) + \epsilon, -(\frac{a_0}{a_1}) + (-\epsilon), \infty]), ([a_2 > 0 \vee a_2 < 0, b^2 \geq 4ac], [-\infty, \frac{-b-\sqrt{b^2-4ac}}{2a} - \epsilon, \frac{-b-\sqrt{b^2-4ac}}{2a}, \frac{-b+\sqrt{b^2-4ac}}{2a}, \frac{-b-\sqrt{b^2-4ac}}{2a} + \epsilon, \infty]$
    (ii) $= \implies [([a_2 = 0 \wedge a_1 = 0 \wedge a_0 = 0], [-\infty, \infty]), ([a_2 = 0 \wedge (a_1 < 0 \vee a_1 > 0)],), [-\infty, -(\frac{a_0}{a_1}), \infty]), ([a_2 > 0 \vee a_2 < 0, b^2 \geq 4ac], [\frac{-b-\sqrt{b^2-4ac}}{2a}, \frac{-b+\sqrt{b^2-4ac}}{2a}]$
    (iii) $> \implies [([a_2 = 0 \wedge a_1 = 0 \wedge a_0 > 0], [-\infty, \infty]), ([a_2 = 0 \wedge (a_1 < 0 \vee a_1 > 0)],), [-\infty, -(\frac{a_0}{a_1}) + \epsilon, -(\frac{a_0}{a_1}) + (-\epsilon), \infty]), ([a_2 > 0 \vee a_2 < 0, b^2 \geq 4ac], [-\infty, \frac{-b-\sqrt{b^2-4ac}}{2a} - \epsilon, \frac{-b-\sqrt{b^2-4ac}}{2a}, \frac{-b+\sqrt{b^2-4ac}}{2a}, \frac{-b-\sqrt{b^2-4ac}}{2a} + \epsilon, \infty]$
  (c) $(a_1 \cdot x^1) + (a_0 \cdot x^0)$
    (i) $< \implies [([a_1 = 0 \wedge a_0 < 0], [-\infty, \infty]), ([a_1 > 0 \vee a_1 < 0], [-(\frac{a_0}{a_1}) + \epsilon, -(\frac{a_0}{a_1}) + (-\epsilon)])]$
    (ii) $= \implies [([a_1 = 0 \wedge a_0 = 0], [-\infty, \infty]), ([a_1 > 0 \vee a_1 < 0], [-(\frac{a_0}{a_1})]$
    (iii) $> \implies [([a_1 = 0 \wedge a_0 < 0], [-\infty, \infty]), ([a_1 > 0 \vee a_1 < 0], [-(\frac{a_0}{a_1}) + \epsilon, -(\frac{a_0}{a_1}) + (-\epsilon)])]$
  (d) $(a_0 \cdot x^0)$
    (i) $< \implies [(a_0 < 0, [-\infty, \infty])]$
    (ii) $= \implies [(a_0 = 0, [-\infty, \infty])]$
    (iii) $> \implies [(a_0 > 0, [-\infty, \infty])]$

  All restriction-substitute pairs are almalgamated into a list of restriction-substitute pairs P.
(6) We then take in P and the normalized statement F' and expand it to $F'_{P_1} \vee \cdots \vee F'_{P_{|P|}}$ where F'_{P_i} is of the form

$$P_i(R) \wedge (F'^{\,x}_{s_1} \vee \cdots \vee F'^{\,x}_{s_{|P_i(S)|}})$$

(7) Finally, we apply base case reduction to each $F'^{\,x}_{s_{i,j}}$ to remove $\pm\infty, \pm\epsilon$, and square root terms.

3.2. **Assumptions Made.** In KeymaeraX, a formula has type [[AtomicFormula]], [[ComparisonFormula]], [[ApplicationOf]],[[UnaryComposition]], [[BinaryComposition]],[[Quantified]], or [[Modal]]. As Virtual Substitution is used on an input of $\mathrm{FOL}_{\mathbb{R}}$, we assumed the input has type [[AtomicFormula]], [[Quantified]], [[ComparisonFormula]], [[UnaryComposition]] and all of its subformula are of the above types. In addition, we assumed that the input formula did not contain universal

quantifiers, as a $\mathrm{FOL}_{\mathbb{R}}$ statement containing a universal quantifier over x is equivalent, in a trivial way, to an $\mathrm{FOL}_{\mathbb{R}}$ statement containing an existential quantifier over x.
Another assumption made was to the construction of terms in $Formula$. In KeymaeraX, the constructors used to create Real terms are $Neg, Plus, Minus, Times$, $Divide$, and $Power$, where all but Neg, which takes in a Real term and subsequently represents of the Real term, take in a pair of Real terms and subsequently represent the application of the corresponding binary operator between the two terms. For example $Div(Number(3), number(2))$ represents $\frac{3}{2}$ and $Power(x, Number(3))$ represents x^3. We assumed for any Real term of the form $Divide(a, b)$, b does not contain any variables which are being quantified over in the Formula. We also assumed that for any Real term of the form $Power(a, b)$, b is strictly of the form $Number(s)$, where since an exponent in an $\mathrm{FOL}_{\mathbb{R}}$ statement should be a natural number. Finally, we assumed that every term in the $\mathrm{FOL}_{\mathbb{R}}$ statement was well defined. For instance, we assumed that $Div(Number(1), Number(0))$ was never an input to our program. Similarly, any convoluted input should still only have well-defined terms.

4. Work and Results

4.1. **Framework.** In order to create our implementation of Virtual Substitution in KeymaeraX, we built a slew of procedures for interpreting Formulas in KeymaeraX. Their name and their description are listed below:

- RemoveMinus : Takes in a term T, and converts it into T', where T' is semantically equivalent to T but no longer Minus constructors.
- NormalizePoly : Takes in term T, and the variable $x$and normalizes T into a polynomial P w.r.t. x
- AddNormalizedPolys : Takes in two normalized polynomials P_1, P_2 w.r.t. to x and the variable x and returns a normalized polynomial P' in x which corresponds to $P_1 + P_2$
- MultNormalizedPolys : Takes in two normalized polynomials P_1, P_2 w.r.t. to x and the variable x and returns a normalized polynomial P' in x which corresponds to $P_1 \cdot P_2$
- NormalizeFormula : Takes in a formula F and a free variable x and converts F into F', where each term in F' is a normalized polynomial of F.
- Degree : Takes in a normalized polynomial P w.r.t. x, and the variable x and returns the degree of P
- Derivative : Takes in a normalized polynomial P w.r.t. x and the variable x and returns $\frac{dP}{dx}$
- Replacement : Takes in a comparison Formula and F whose terms are normalized polynomials P_1, P_2 w.r.t. x and returns a list of restriction-substitution pairs for the comparison formula.
- ComputeReplacements : Takes in a Formula F and variable x and returns a list of all restriction-substitution pairs for the entire formula.
- SubstituteTerm : Takes in a Term P, a replacement term S, and a variable x, and replacemes all appearances of x in P with S

- SubstituteForm : Takes in a Formula F, a replacement term S and a variable x and replacements all appearances of X in F with S.

4.2. **Normalizing Polynomials.** We provide a further elaboration on the Normalization Procedure. In KeymaeraX, a polynomial is represented using constructors and terms. However, almost every non-trivial polynomial can be represented in a large number of ways in KeymaeraX. For example, the following are all valid ways to represent $x + 1 + 2 + y = 0$ in KeymaeraX:

$$\begin{aligned} Equal(Plus(Plus(Plus(x, 1), 2), y), 0) &\equiv Equal(Plus(Plus(x, Plus(1, 2)), y), 0) \\ &\equiv Equal(Plus(x, Plus(1, Plus(2, y))), 0) \equiv \cdots \end{aligned}$$

In particular, if a comparison formula's left hand side (LHS) has no parenthesis and all binary operators are associative, then necessarily there are 2^{f-1} ways to write the LHS in keymaeraX, where f is the number of binary operators on the LHS. Thus, it's practically impossible to simply pattern match out expressions. To deal with this, we identified a canonical form to represent polynomials in KeymaeraX and wrote functions to normalize a given polynomial into this form. In particular, we defined the form as follows: Let $\alpha(x)$ be a polynomial in x built from the constructors $Plus, Minus, Times, Divide, Power$ and Neg, where exponents are strictly of the form $Number(s)$, and if the representation of $\alpha(x)$ contains the form $Div(a, b)$, then b is x-free. Then **NormalizePoly** takes $\alpha(x)$ and converts it to the form $P(x)$, where the structure of $P(x)$ is defined as follows:

$$P(x) := Times(a, Power(x, Number(s))) \mid Plus(Times(a', Power(x, Number(s'))), P'(x))$$

where P, a, s, P', a', s' have no meaning other than the fact that s,s' are numbers, a, a' are x-free and $P(x), P'(x)$ both have polynomial form. We achieved this by identifying a base case for a Normalized Polynomial, $Times(a, Power(x, Numbers(S)))$, recursively pattern matching through a term until we were left with x, $Power($x$, Number(s))$ or a non-x symbol y, converting non-x symbols y into $Times(y, Power(x, Number(0)))$, x into $Times(Number(1), Power(x, Number(1)))$ and then using **AddNormalizedPolys** and **MultNormalizedPolys** to add and multiply Normalized Polynomials together while maintaining the Normalized Polynomial form.

4.3. **Results.** Unfortunately, due to time pressures and deadlines, we were unable to write the last necessary component of the VS implementation, specifically we did not write a function that converted a virtually substituted formula into $\text{FOL}_{\mathbb{R}}$ formula using base cases described in section 2.

However, we believe the current state of the implementation provides a foundation for VS in KeymaeraX. If given an adequate daily time allotment, we believe the implementation can be completed within at most a week.

References

[1] Weispfenning, Volker, Appl. Algebra Eng. Commun. Comput., 2, 85-101, Quantifier Elimination for Real Algebra — the Quadratic Case and Beyond, 8, 1997 2

[2] Loos, Rüdiger and Weispfenning, Volker, Comput. J., 5, 450-462, Applying Linear Quantifier Elimination., 36, 1993 2

[3] Alfred Tarski, A decision method for elementary algebra and geometry, 1951, University of California Press 1