Recitation 12: Real Arithmetic
15-424/15-624/15-824 Logical Foundations of Cyber-Physical Systems

Notes by Brandon Bohrer. Edits by Yong Kiam Tan and, later, Katherine Cordwell (kcordwel@cs.cmu.edu).

1 Announcements

- Proof redos for Lab 3/4 due by Friday before midnight. Submit to me by email.
- Proposals to be graded soon, followed by Lab 4 and Assignment 5 (in that order).
- Project deadlines:
  - Final Project (your code/models/proofs/etc.), December 5, 11:59PM. **Submit a zip file.**
  - Term Paper (*scientific* paper describing the work done), December 6, 11:59PM. **Submit a pdf.**
  - Presentation Slides, email to me ideally Monday December 9 by midnight, or, less ideally, **very early on Tuesday i.e. before 2 am**—well before the Grand Prix.

  Watch on Piazza for further announcements.

2 Motivation and Learning Objectives

In class, we have started looking at how to rigorously prove real arithmetic properties. Until now, the real arithmetic proofs in the labs relied (mainly) on a call to QE in KeYmaera X. However, following the theme of the uniform substitution lectures, why should we trust that the QE tool is correctly implemented? What if it tells us something is true when it is really false? And how are they even implemented in the first place?

One technique for QE that we have started looking at in class is Virtual Substitution (VSubst). VSubst provides an intuitive (albeit restricted to low degree) way of quantifier elimination that is still relatively straightforward. So far we've only seen VSubst with equalities.

In this recitation, we will first take a detour and discuss *cylindrical algebraic decomposition (CAD)*. CAD was developed by George Collins in 1975, and it is the best known algorithm for QE. Then we'll discuss VSubst with inequalities and draw connections between it and CAD. We'll finish with a high level overview of QE techniques and some tips and tricks for speeding up QE which you have seen throughout the semester.

3 Cylindrical Algebraic Decomposition (CAD)

Here is some general intuition for quantifier elimination procedures: The continuous nature of the reals makes computation difficult. So the key idea is to reduce a continuous problem to something discrete (computable).

CAD does this by taking real space ($\mathbb{R}^n$, if there are n variables in the polynomials in the QE query) and partitioning it into finitely many *sign-invariant regions*. This means that the sign of each input polynomial (each polynomial involved in the QE query) is constant on the entire region. Intuitively, the reason why there are only finitely many such regions will be that each polynomial p changes sign at most $\deg(p)$ times.

Exercise 1:
How does such a decomposition help?

Hint: Remember that we're trying to reduce a continuous problem to a discrete problem.

Answer: If we take one point from each sign-invariant region, that will give us a finite collection of points where the behavior of the input polynomials at this collection of points is representative of the behavior of the input polynomials *on the entirety of* $\mathbb{R}^n$.

3.1 Decomposing CAD (or: How does CAD Actually Work?)

Cylindrical algebraic decomposition works in two phases: projection and lifting (see Figure 1). The projection phase successively generates sets of polynomials where after each projection, one variable is eliminated. In the lifting phase, a CAD is constructed for the last projection set. Because this projection set contains only univariate polynomials, the CAD construction only requires finding the roots of univariate polynomials. Then, variables are added back in one by one as we "lift" each CAD for $k-1$ variables to a CAD for k variables.

The projection sets satisfy the following property: each polynomial in proj^{i-1} is *delineable* over each cell in proj^{i}. We omit the formal definition of delineability and instead focus on what delineability ensures: To paraphrase [Bro01], the delineability property ensures that the projections of different cells are either disjoint or identical, so that if CAD A is lifted from CAD B, the cells in CAD A are cylindrically arranged over the cells in CAD B. We explain the significance of this cylindrical arrangement with an example in Section 3.2.

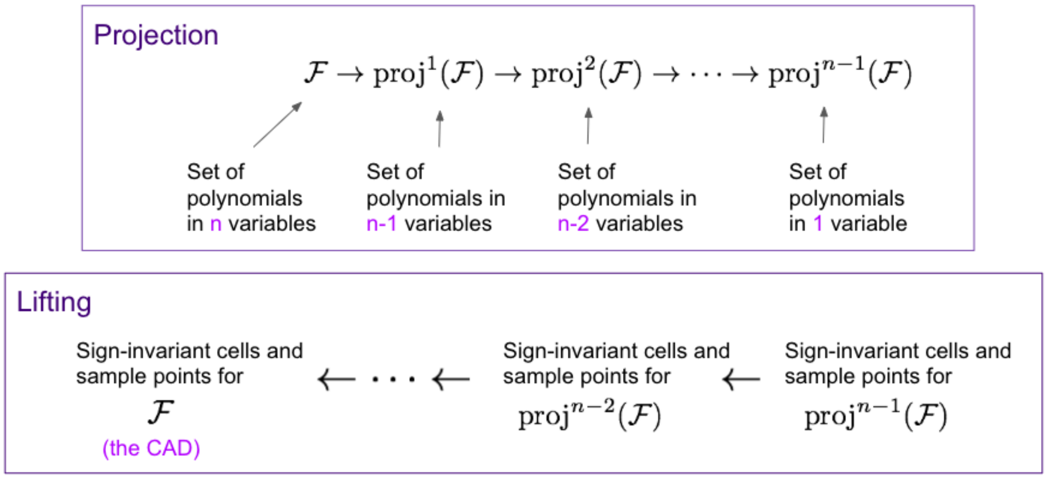

Projection
$\mathcal{F} \to \text{proj}^1(\mathcal{F}) \to \text{proj}^2(\mathcal{F}) \to \cdots \to \text{proj}^{n-1}(\mathcal{F})$
Set of polynomials in n variables
Set of polynomials in n-1 variables
Set of polynomials in n-2 variables
Set of polynomials in 1 variable
Lifting
Sign-invariant cells and sample points for $\mathcal{F}$
(the CAD)
$\leftarrow \cdots \leftarrow$
Sign-invariant cells and sample points for $\text{proj}^{n-2}(\mathcal{F})$
$\leftarrow$
Sign-invariant cells and sample points for $\text{proj}^{n-1}(\mathcal{F})$

Figure 1: Overview of CAD

The projection operator *proj* is carefully designed so that delinability holds. The number of polynomials in the projection set has a significant influence on the efficiency of CAD, and so many improved projection sets have been developed for certain situations. We largely blackbox *proj*, but we note that the set $proj(f_1, \ldots, f_k)$ includes discriminants of f_i for all i (taken with respect to the particular variable that is being eliminated) and pairwise resultants $res(f_i, f_j)$ for all $i \neq j$ (again taken with respect to the particular variable that is being eliminated)—and in fact, as we will discuss later, even improved projection sets involve calculating these discriminants and resultants (see [Bro01], [Str00]).

3.2 An Example (from [Jir98])

Jirstrand gives a beautiful illustration of the lifting phase and the cylindrical arrangement of cells in his thesis [Jir98]. We follow the example of CAD that he discusses in Section 5.4.2 of [Jir98]. This example is for CAD construction for $f = (x_1 - 2)^2 + (x_2 - 2)^2 + (x_3 - 2)^2 - 1$. In the projection phase, first x_3 is eliminated, and then x_2 is eliminated.

Figure 2 is a modification of Figure 5.7 from [Jir98]. It shows the CAD construction for this example geometrically. The sphere depicts the zeros of f, and the circle in the x_2x_1 plane depicts the zeros of $\text{proj}^1(f)$ (which is a set of bivariate polynomials in x_1 and x_2). The zeros for $\text{proj}^2(f)$, which is a set of univariate polynomials in x_1, are $x_1 = 1$ and $x_1 = 3$ (pictured in light red in the first and second of the three figures).

The first of the three figures shows the CAD for $\text{proj}^2(f)$. It contains the points 1 and 3 (shown in light red) and one arbitrary sample point from each of the intervals $(-\infty, 1)$, $(1, 3)$, and $(3, \infty)$. The sample points from these intervals are shown in dark red. The second of the three figures shows the lifting of this CAD to a CAD for $\mathbb{R}^2$, the sample points of which are shown in blue. For clarity, we show the sample points of the CAD that intersect the circle (the zero set of $\text{proj}(f)$) in light blue, and sample points of the CAD that do not intersect the zero set of $\text{proj}(f)$ in dark blue. As we can see, since the line $x_1 = .5$ does not intersect the zero-set of the polynomials in proj(f), $(.5, a)$ and $(.5, b)$ will have the same sign on proj(f) for any a and b. Thus we only need one sample point when we lift $x_1 = .5$ to a CAD for proj(f). However, the line $x_1 = 1$ intersects the zero-set of the polynomials in proj(f) at $x_2 = 2$. So, we need to have three sample points when we lift $x_1 = 1$ to a CAD for proj(f): $(1, a)$ (shown in dark blue), $(1, 2)$ (shown in light blue), and $(1, b)$ where $a < 2$ and $b > 2$ (shown in dark blue). This lifting works similarly for $2, 3$, and 3.5.

Note that we can set the first coordinates of the sample points for the CAD in $\mathbb{R}^2$ to be the values of the sample points for the CAD in $\mathbb{R}$. This is what the cylindrical arrangement and delineability ensure—effectively, the lifting phase is reduced to finding roots of univariate polynomials.

The last of the three figures shows the lifting of the CAD for proj(f) to the CAD for f. Here, the green points are the sample points for the CAD for f. For clarity, we show the sample points that intersect the sphere (the zero set of f) in light green and all other sample points in dark green.

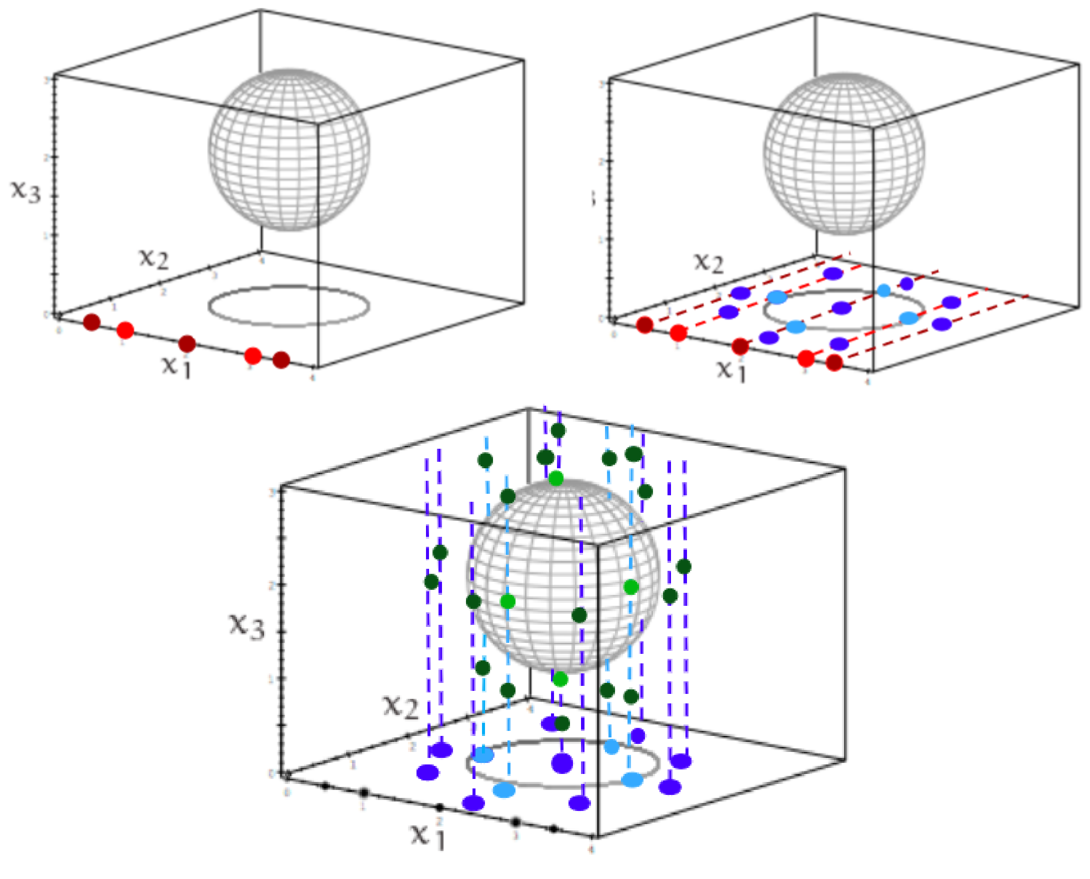


Figure 2: The Lifting Phase. Modified from Figure 5.7 in Jirstrand [Jir98] (color added)

When we lift over a given sample point (a, b), we are fixing $x_1 = a$ and $x_2 = b$. We then take the line through $(a, b, 0)$ and $(a, b, 1)$ and look at the points where it intersects the sphere. If this line never intersects the sphere, then we only need one sample point when we lift (a, b). This is the case for all of the dark blue points except the one in the middle of the circle. If this line intersects the sphere in one point, say (a, b, z), then we need three sample points—(a, b, p), (a, b, z), and (a, b, q), where $p < z$ and $z < q$. This is the case for each of the light blue points. Finally, if the line intersects the sphere in two points, say (a, b, w) and (a, b, z) with $w < z$, then we need five sample points—(a, b, p), (a, b, w), (a, b, q), (a, b, z), and (a, b, r) where $p < w$, $w < q < z$, and $z < r$. This is the case for the dark blue point in the middle of the circle (notice that (a, b, q) is inside the sphere and thus is not pictured). Again, we see visually how the CAD for f has been cylindrically lifted over the CAD for proj(f), and again we can see how delineability ensures that we can set the first two coordinates of the sample points for the CAD in $\mathbb{R}^3$ to be the coordinates of the sample points for the CAD in $\mathbb{R}^2$—so that the lifting phase really only requires finding roots of univariate polynomials.

Since CAD is a fundamental algorithm that is used extensively in QE procedures, improving CAD is an important and current area of research.

4 Virtual Substitution with Inequalities

The intuition behind Vsubst has a lot in common with the intuition behind CAD. Vsubst is good with *low-degree polynomials* when roots can be found explicitly.

Remember first the equational-case, where we have the following theorem:

Theorem (Virtual Substitution: Quadratic Equation $x \notin a, b, c$)

$$a \neq 0 \lor b \neq 0 \lor c \neq 0 \rightarrow$$
$$\Big(\exists x\,(ax^2 + bx + c = 0 \land F) \leftrightarrow$$
$$a = 0 \land b \neq 0 \land F_{\bar{x}}^{-c/b}$$
$$\lor\, a \neq 0 \land b^2 - 4ac \geq 0 \land \left(F_{\bar{x}}^{(-b+\sqrt{b^2-4ac})/(2a)} \lor F_{\bar{x}}^{(-b-\sqrt{b^2-4ac})/(2a)}\right)\Big)$$

What this theorem is saying is that since $ax^2 + bx + c = 0$ must be satisfied for our QE query to be true, we can solve for the roots of $ax^2 + bx + c$ and then virtually substitute those for x into the rest of the polynomials.

Exercise 2:
Do you see a connection to CAD?
Answer: In some sense, we are zooming in on some of the useful sample points. We know that the only relevant sample points that could possibly make this QE query true have x set to a solution of $ax^2 + bx + c = 0$. So we can focus in on these sample points and ignore points/regions in the CAD with any other value of x.

Exercise 3:

What would change if suddenly we had an inequality, i.e. $\exists x(ax^2 + bx + c \sim 0 \wedge F)$ where $\sim \in \{\geq, \leq, >, <\}$?
Answer: Well, suddenly we'll need to consider more sample points. We can't just consider the sample points r_1 and r_2 where $ax^2 + bc + c = 0$. We'll also care about sample points where $x < r_1, r_1 < x < r_2$, and $x > r_2$. **Furthermore**, it will also matter if x occurs in polynomials in F.

Exercise 4:
Why does it matter if x occurs in polynomials in F? Why did this not matter in the $ax^2 + bx + c = 0$ case?
Answer: Intuitively, with $ax^2 + bx + c = 0$, we were constrained to particular sample points. When we have inequalities, we are now picking relevant sample points from **ranges**. Whether x occurs in polynomials in F will influence what ranges we care about.

As an example of this, what if we have $x^2 - 25 \geq 0 \wedge x^2 - 9 < 0$? If we only looked at $x^2 - 25 \geq 0$, then we'd consider sample x's from ranges $(-\infty, -5), [-5, -5], (-5, 5), [5, 5], (5, \infty)$. We might as well choose $-6, -5, -4, 5, 6$.

Exercise 5:
This is a bad choice of sample points. Why?
Answer: At all of these sample points, we have$x^2 - 9 \geq 0$. So we'd return that $\exists x(x^2 - 25 \geq 0 \wedge x^2 - 9 < 0)$ is false. But of course this formula is true, and if we instead chose 0 as our sample point instead of -4, then we'd see $0^2 - 9 \leq 0$. So we haven't hit representative sample points.

So there are two challenges with inequalities: First, we need Vsubst to always pick representative sample points, and then we want to virtually substitute them in.

4.1 How Vsubst Actually Works

Now that we've seen some of the connections to CAD and some of the extra challenges with Vsusbt with inequalities, let's see how to resolve these challenges.

First of all, let's think about the case where x does not occur in F, and say that ax^2+bx+c is such that $a \neq 0$ (so that we do truly have a quadratic).

Exercise 6:
What sample points might we choose?
Answer: When x does not occur in F there are various answers. But we will focus on a choice of sample points that will generalize well to the case where x does occur in polynomials in F. If $\sim$ is a weak inequality, we'll use $-\infty$, $\frac{-b-\sqrt{b^2-4ac}}{2a}$, $\frac{-b-\sqrt{b^2-4ac}}{2a} + \epsilon$, $\frac{-b+\sqrt{b^2-4ac}}{2a}$, and $\frac{-b+\sqrt{b^2-4ac}}{2a} + \epsilon$. If $\sim$ is a strong inequality, we no longer care about the roots so we'll use $-\infty$, $\frac{-b-\sqrt{b^2-4ac}}{2a} + \epsilon$, and $\frac{-b+\sqrt{b^2-4ac}}{2a} + \epsilon$.

There's a lot going on here. In particular, what do we mean by $-\infty$ and ϵ? Well, $-\infty$ is the "rubber-band" number that is as negative as you want it to be. To quote from the textbook, we should think of $-\infty$ as being "built out of elastic rubber so that it always ends up being smaller when compared to any real number". Similarly, ϵ is another "rubber-band" number. It satisfies two properties: it's positive, and it's always as small as you want it to be—i.e. $\epsilon > 0$, $\epsilon < r$ for all $r \in \mathbb{R}$ with $r > 0$.

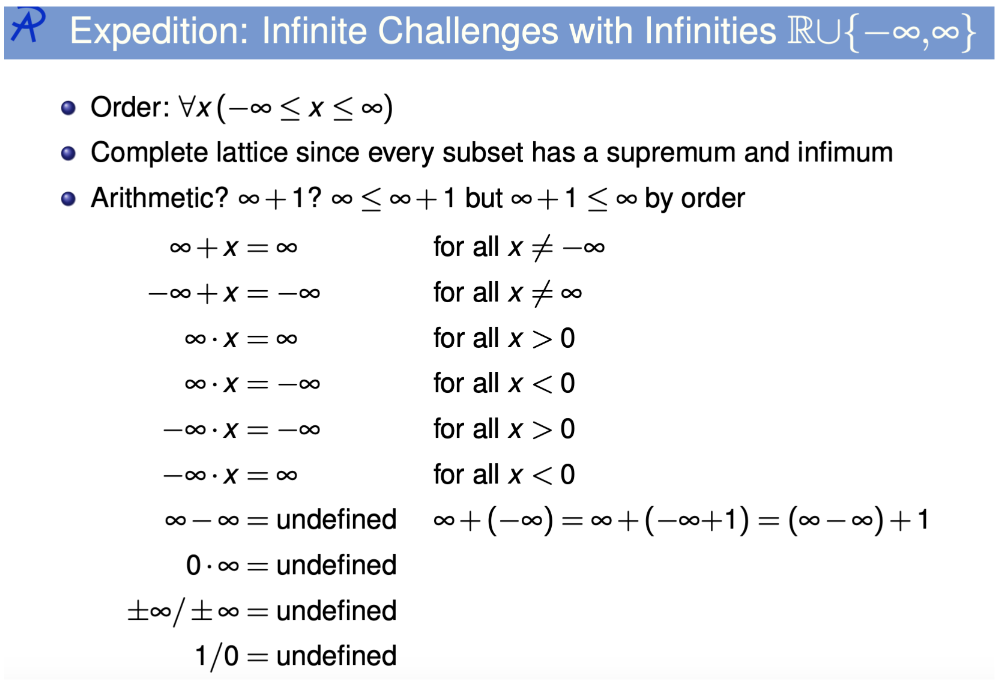
Exercise 7:
Convince yourselves that these sample points cover all relevant ranges.

WHAT HAVE WE DONE?

Well, we've delved into nonstandard analysis. By considering $-\infty$, we're thinking about the extended reals. We'll need to understand some computational properties of $-\infty$.

With ϵ, we're doing something that's arguably even more crazy! Suddenly we need to understand not only some computational properties of ϵ but also some ordering properties with ϵ. To fully understand infinitesimals, nonstandard analysts have developed structures like the surreals, superreals, or hyperreals. But fortunately Vsubst only needs tame ordering principles: $\epsilon > 0$ and $\forall x \in \mathbb{R}(x > 0 \to \epsilon < x)$ is enough. So while we're getting a little wild by adding ϵ, it's manageable.

In case you are curious, here are Prof. Platzer's relevant slides on computational properties of $-\infty$ and ϵ:



Expedition: Infinite Challenges with Infinities $\mathbb{R}\cup\{-\infty,\infty\}$

- Order: $\forall x\,(-\infty \le x \le \infty)$
- Complete lattice since every subset has a supremum and infimum
- Arithmetic? $\infty + 1$? $\infty \le \infty + 1$ but $\infty + 1 \le \infty$ by order

$\infty + x = \infty$	for all $x \neq -\infty$
$-\infty + x = -\infty$	for all $x \neq \infty$
$\infty \cdot x = \infty$	for all $x > 0$
$\infty \cdot x = -\infty$	for all $x < 0$
$-\infty \cdot x = -\infty$	for all $x > 0$
$-\infty \cdot x = \infty$	for all $x < 0$
$\infty - \infty =$ undefined	$\infty + (-\infty) = \infty + (-\infty + 1) = (\infty - \infty) + 1$
$0 \cdot \infty =$ undefined	
$\pm\infty / \pm\infty =$ undefined	
$1/0 =$ undefined	

Expedition: Infinitesimal Nonstandard Field Ext. $\mathbb{R}[\varepsilon]$

ε is "always as small as needed"

- Positive: $\varepsilon > 0$
- Smaller: $\forall x \in \mathbb{R}\,(x > 0 \to \varepsilon < x)$
- Standard $\mathbb{R}$ are Archimedean: $\forall x \in \mathbb{R} \setminus \{0\}\ \exists n \in \mathbb{N}\ |\underbrace{x + x + \cdots + x}_{n \text{ times}}| > 1$
- $\mathbb{R}[\varepsilon]$ are non-Archimedean: $\underbrace{\varepsilon + \varepsilon + \cdots + \varepsilon}_{\text{any } n \in \mathbb{N} \text{ times}} < 1$
- Infinitesimals as inverses of infinities?

$$\varepsilon \cdot \infty = 1? \qquad -\varepsilon \cdot -\infty = 1? \qquad (\varepsilon + 1) \cdot (\underbrace{\infty + 2}_{\infty?}) = \ldots$$

- How to order for $x \neq 0$?

$$\varepsilon^2 \;<\; \varepsilon \;<\; x^2 + \varepsilon \;<\; (x + \varepsilon)^2 \;<\; x^2 + 2\varepsilon x + 5\varepsilon + \varepsilon^2$$

Note: We didn't spend much time discussing computational properties of $-\infty$ and ϵ in recitation. More detail on this is available in Chapter 21 of the textbook.

Remember that we are doing **virtual substitution**. So ultimately we only need to handle $-\infty$ and ϵ virtually, just like we handled square roots virtually in the last lecture. Here's how we do that for $-\infty$

Virtual Substitution of $-\infty$ into Comparisons $\quad p = \sum_{i=0}^{n} a_i x^i$

$$(p = 0)_{\bar{x}}^{-\infty} \equiv \bigwedge_{i=0}^{n} a_i = 0$$

$$(p \leq 0)_{\bar{x}}^{-\infty} \equiv (p < 0)_{\bar{x}}^{-\infty} \vee (p = 0)_{\bar{x}}^{-\infty}$$

$$(p < 0)_{\bar{x}}^{-\infty} \equiv p(\text{-}\infty) < 0$$

$$(p \neq 0)_{\bar{x}}^{-\infty} \equiv \bigvee_{i=0}^{n} a_i \neq 0$$

Ultimately negative at $-\infty$ $\quad \lim_{x \to -\infty} p(x) < 0$

$$p(\text{-}\infty) < 0 \overset{\text{def}}{\equiv} \begin{cases} p < 0 & \text{if } \deg(p) \leq 0 \\ (-1)^n a_n < 0 \vee \left(a_n = 0 \wedge (\sum_{i=0}^{n-1} a_i x^i)(\text{-}\infty) < 0\right) & \text{if } \deg(p) > 0 \end{cases}$$

...and here's how we do it for $e + \epsilon$.

Virtual Substitution of $e+\varepsilon$ into Comparisons $\quad p = \sum_{i=0}^{n} a_i x^i$

$$(p=0)_{\bar{x}}^{e+\varepsilon} \equiv \bigwedge_{i=0}^{n} a_i = 0$$
$$(p \leq 0)_{\bar{x}}^{e+\varepsilon} \equiv (p<0)_{\bar{x}}^{e+\varepsilon} \vee (p=0)_{\bar{x}}^{e+\varepsilon}$$
$$(p<0)_{\bar{x}}^{e+\varepsilon} \equiv (p^{+}<0)_{\bar{x}}^{e}$$
$$(p \neq 0)_{\bar{x}}^{e+\varepsilon} \equiv \bigvee_{i=0}^{n} a_i \neq 0$$

Immediately negative at x $\quad \lim_{y \searrow x} p(y) < 0$

$$p^{+} < 0 \overset{\text{def}}{\equiv} \begin{cases} p<0 & \text{if } \deg(p) \leq 0 \\ p<0 \vee (p=0 \wedge (p')^{+} < 0) & \text{if } \deg(p) > 0 \end{cases}$$

Note: We broke down these definitions a bit in recitation. They are elaborated in Chapter 21 of the textbook.

Exercise 8:
Finally, how do we do Vsubst with inequalities when x occurs in F? What sample points can we pick?
Answer: This is a trick question because in general we cannot do Vsubst with inequalities when x occurs in F.

Exercise 9:
What condition might we add to be able to generalize?
Answer: If x only occurs with degree at most 2 in F, then we can generalize Vsubst.

Exercise 10:
Why is this condition useful?
Answer: Because now we can get sample points for all of the regions very explicitly.

Here is the theorem:

Theorem (Virtual Substitution: Quadratics) (Weispfenning'97)

Let all atomic formulas in F be of the form $ax^2 + bx + c \sim 0$ with $x \notin a, b, c$ and $\sim \in \{=, \leq, <, \neq\}$ and its discriminant $d \overset{\text{def}}{=} b^2 - 4ac$.

$$\begin{aligned}
&\exists x\, F \leftrightarrow \\
&F_{\bar{x}}^{-\infty} \\
\lor &\bigvee_{(ax^2+bx+c \{\substack{=\\ \leq}\} 0) \in F} \left(a{=}0 \land b{\neq}0 \land F_{\bar{x}}^{-c/b} \lor a{\neq}0 \land d{\geq}0 \land (F_{\bar{x}}^{(-b+\sqrt{d})/(2a)} \lor F_{\bar{x}}^{(-b-\sqrt{d})/(2a)})\right) \\
\lor &\bigvee_{(ax^2+bx+c \{\substack{\neq\\ <}\} 0) \in F} \left(a{=}0 \land b{\neq}0 \land F_{\bar{x}}^{-c/b+\varepsilon} \lor a{\neq}0 \land d{\geq}0 \land (F_{\bar{x}}^{(-b+\sqrt{d})/(2a)+\varepsilon} \lor F_{\bar{x}}^{(-b-\sqrt{d})/(2a)+\varepsilon})\right)
\end{aligned}$$

A Technicality

Note that there is a subtle optimization here: If we have a weak inequality, we only need to virtually substitute the roots and $-\infty$. If we have a strong inequality, we only have to virtually substitute $-\infty$ and the roots $+\ \epsilon$. To see this fully, you may wish to spend some time thinking about it or reading the soundness proof (Section 21.5 in the textbook). My best intuition for it is the following: If r is the root of a weak inequality, $r+\epsilon$ adds no significant information as a sample point because 1) it is not a root of any of the weak inequalities (since ϵ is a rubber band) and 2) every polynomial f that is positive at r will also be positive at $r + \epsilon$(and similarly every f that is negative at r will also be negative at $r + \epsilon$). (Admittedly this is a somewhat univariate intuition, but we can think of other variables as being previously instantiated.)

We care about $-\infty$ as a representative point in case our weak inequality has no roots and also for the strong inequalities to sample from every representative cell.

Previously we hinted that the choice of $-\infty$ and the roots $+\ \epsilon$ as the sample points is good because it generalizes well to many polynomials. To see this, forget about the optimization for a second and think about if we have just a single quadratic with two roots, $x = r_1$ and $x = r_2$. Then we may as well pick sample points $r_1 - 1, r_1, (r_1 + r_2)/2, r_2$, and $r_2 + 1$ instead of worrying about the funny $-\infty$ and ϵ. But what if we now add in a second quadratic with two roots $x = r_3$ and $x = r_4$? Suddenly we don't know how r_1, r_2, r_3, and r_4 compare to each other, and we no longer know which interval $(r_1 + r_2)/2$ is representing. We'd have to do a lot of comparisons to get some ordering on r_1, r_2, r_3 and r_4 to keep up this strategy. By contrast, we know that $-\infty$ is less than all of our sample points, so it covers the leftmost interval. And we know that $r_1 + \epsilon$ is covering the interval just to the right of $r_1, r_2 + \epsilon$ is covering the interval just to the right of r_2, and so on. So we can be sure we've covered all the intervals very easily. Of course the tradeoff is the extra fiddling with $-\infty$ and ϵ, but this doesn't matter so much because we're substituting them virtually anyways.

Note: Past this point we covered these notes in less detail. If you're really curious about real arithmetic, there's some cool material in the next few sections that we didn't get to in recitation!

5 QE Algorithms and Their Complexity

One of the most useful things we learn from studying QE over the reals is how slow it will be in different circumstances. This informs how we should try to simplify arithmetic in order to improve proof performance. There are some simple bounds fundamental to QE, regardless of which algorithm is used:

- All QE algoriths are at best doubly exponential in the number of *quantifier alternations* (i.e., $\exists x\ \exists y\ \forall z\ \phi$ and $\forall x\ \exists y\ \phi$ both count as one alternation).
- All *known practical algorithms* are doubly exponential in the *number of variables*, regardless of the number of alternations.
- If there are *no alternations*, then in theory the lower bound is *singly exponential* in the number of variables, but in practice still doubly exponential.

What does this teach us about QE in general? When possible, eliminate variables (and quantifiers) yourself instead of leaving it to QE. The more simplifications you can do, the faster your proofs will close.

Another way to develop good intuition is to look at the actual asymptotic time bound for a specific QE algorithm. For example, here is the time bound from Collins' original CAD (Cylindrical Algebraic Decomposition) paper [Col75], where:

- m is the number of polynomials in the formula,
- r is the number of variables,
- n is the largest degree of any variable in any polynomial,
- d is the length of the largest coefficient in any polynomial (which is generally constant), and
- a is the number of atomic propositions in the formula (also often constant)

$$CAD \in \mathcal{O}((2n)^{2^{2r+8}} m^{2^{r+6}} d^3 a)$$

What do you learn from this? You learn that n and r both have a *serious* impact on the running time. Getting rid of variables (reducing r) is usually easier than reducing n unless you get lucky, but reducing n can be helpful when possible. For example if you have some assumptions with x^3 or x^4 terms in them, you should see if you can get away with hiding them or even simplifying them to formulas with only x and x^2 terms, which are much faster. Reducing m is useful too, but it is only the base of a singly exponential, so it quickly gets dominated by the doubly exponential $(2n)^{2^{2r+8}}$. That being said, it is often the single easiest parameter to reduce (it goes down any time you hide any assumption at all), so why not decrease it if it is easy to do so?

6 Comparison of Algorithms

We went through a number of different algorithms in lecture, and you might be wondering why we have so many different algorithms. The algorithms have different strengths and weaknesses. Some of those that apply to smaller fragments of real arithmetic are much easier to understand than others.

- *Cylindrical Algebraic Decomposition* (CAD): CAD is the oldest (relatively) practical QE algorithm for the complete first-order theory of the reals.

- *Partial CAD* (PCAD): PCAD is an optimized followup algorithm to CAD which is even more efficient in practice and equally general, but even more complex. Mathematica (most likely) uses PCAD with extra tricks. The takeaway is that the CAD time bound and intuition above are mostly accurate for Mathematica, but the reality of their implementation is a bit more complicated.

- *Other modified/optimized versions of CAD*: Many people have worked to improve CAD over the years, so there are other variations on CAD, e.g. CADMD (CAD maximum dimension) which works when there are no weak inequalities in the QE query.

- *Virtual Substitution*: Virtual Substitution is conceptually simpler than the CAD family, but very incomplete. It only supports terms up to degree 2 (x^2) (or degree 3 with extreme effort), and can fail even on x^2 terms because VS can introduce higher degree terms internally. Why is it so useful, then? It turns that (a) it often works even when the theory does not give you a completeness guarantee, (b) optimizations can make it work even more often, (c) many models of interest to us only have x^2 terms, and (d) perhaps most importantly: it can make a useful pre-processing step for a more general method like CAD.

- *Satisfiability Modulo Theories Solving* (SMT): SMT solving is an extremely general purpose automated proving technology used in many automated theorem proving domains. In SMT solving, a general purpose SAT solver is given a *theory* (set of axioms/rules) that tell it how to prove a new class of problems. Unlike the CAD family, SMT is logic based and thus in principle has the potential to handle the combinatorial logical challenges of big formulas with big terms quickly as long as the arithmetic itself works out well enough. The completeness of SMT solving depends on which axioms the solver uses. Z3's QE procedure is SMT based. Takeaway: Z3's and Mathematica's QE differ on a very fundamental level, and thus each one might do well on problems where the other struggles.

- *Linear Real Arithmetic* (LRA): One powerful subclass of real arithmetic questions that Z3 works well with is LRA. Here, all of the terms are required to be linear, i.e., no products between variables. This theory is decidable by Fourier-Motzkin elimination. The idea is relatively straightforward so let us take a quick look at it. As we saw in class, quantifiers can be eliminated homomorphically across the logical connectives (and using logical equivalences to simplify away unnecessary operators) e.g.,:

  $$QE(P \vee Q) \equiv QE(P) \vee QE(Q)$$

  $$\forall x\, P \leftrightarrow \neg \exists x\, \neg P$$

  For LRA, we will use a similar idea to avoid dealing with disjunctions when eliminating quantifiers with the equivalence:

  $$\exists x\, (P \vee Q) \leftrightarrow \exists x\, P \vee \exists x\, Q$$

  The "base case" formula that we need to eliminate quantifiers from therefore looks like this (we will ignore strict inequalities), with p_i linear polynomials on the variables:

  $$\exists x \bigwedge_i p_i \leq 0$$

  Now, because each p_i is linear, we can rearrange (and divide) the inequalities to move x to one side, yielding the following formula:

  $$\exists x\, \big(\bigwedge_{i=1}^{N} a_i \leq x \wedge \bigwedge_{j=1}^{M} x \leq b_j \big)$$

  The resulting a_i, b_j are polynomials over the remaining variables (not including x). Finally, we may eliminate the existential quantifier on x with the following formula:

  $$\bigwedge_{i}^{N} \bigwedge_{j}^{M} a_i \leq b_j$$

  Exercise 11:
  Convince yourselves that the previous two formulas are equivalent.

  In the above algorithm, we would rewrite original input (linear) equalities $p = 0$ by transforming it into the conjunction $p \leq 0 \wedge p \geq 0$. This is obviously not the most optimal thing to do in practice, even though it simplifies the theory.

  Exercise 12:
  How could we handle equational constraints?

Answer: We have actually seen this in class with the virtual substitution of equations.

Suppose again that we were trying to eliminate an existential quantifier on x but we had an additional linear equational constraint $(p = 0)$:

$$\exists x \left(p = 0 \wedge \bigwedge_i p_i \leq 0\right)$$

Since p is linear, there are only two possibilities: 1) x appears with a non-zero, rational coefficient or 2) x does not appear (or has coefficient 0).

In the first case, we can eliminate x by rearranging the constraint $p = 0$ into $x = q$ for some linear polynomial q that does not mention x. Then, we can eliminate the x quantifier with the following formula, where $(p_i)_x^q$ means to replace x by q wherever it appears in p_i:[1]

$$\bigwedge_i (p_i)_x^q \leq 0$$

In the latter case, the equality $p = 0$ can be moved out of the quantification since it does not depend on the value of x. We then continue eliminating quantifiers on the remaining inequalities as we did before.

$$p = 0 \wedge \exists x \bigwedge_i p_i \leq 0$$

- *Cohen-Hörmander* (CH): Cohen-Hörmander is complete in theory and still reasonably straightforward, and somewhat practical for small problems. It is notable because there is a *proof-producing* implementation for it, meaning it can be added to a theorem prover without significantly increasing the amount of trusted code in the prover. Whereas the CAD's and SMT both require us to trust complex solvers, the proof-producing version of CH generates a proof that we can check on our own.

- *Witness Generating Procedures*: The idea of proof witnesses is especially attractive if it is easy to convince a theorem prover (or even just ourselves) that the witness is correct. Of course, finding such a witness may be difficult. As an analogy, think of the complexity class NP. It is much easier to verify a solution to an instance of a NP-complete problem (say, 3SAT) than it is to find the solution!

  Let us look at one particular approach from [PQR09] that works for the universal fragment of real arithmetic (i.e., with all universal quantifiers when written in prenex normal form). These kinds of real arithmetic formulas are, in fact, those that you have been using throughout the course. To simplify matters, let us suppose that you were

[1]Observe that linearity is important here to make sure that the resulting $(p_i)_x^q$ are also linear.

trying to prove a sequent that looks like the following ($\sim$ stands for one of the usual comparison operators):

$$\Gamma \vdash q \sim 0$$

How could you proceed? Well, one way is to move q onto the LHS, and attempt to prove *false*, i.e., derive a contradiction from the resulting assumptions:

$$\Gamma, \neg q \sim 0 \vdash \mathit{false}$$

So in general, after normalizing appropriately, the question we are interested in looks like this (p_i are polynomials, and each $\sim$ is a comparison operator):

$$p_0 \sim 0, p_1 \sim 0, \ldots, p_n \sim 0 \vdash \mathit{false}$$

It is possible to derive a witness of the contradiction directly from here, but let us take it a step further by turning all the $\sim$ into equalities. After all, equalities seem a little tamer from our experience with LRA and VSubst. We can use the following real arithmetic equivalence on non-strict inequalities:

$$p \geq 0 \leftrightarrow \exists z \, p - z^2 = 0$$

Exercise 13:
What about $p > 0$ and $p \neq 0$? (Think about how we used differential ghosts!)

Using these equivalence transformations, we would end up with a sequent that looks like this (here, q_i are again polynomials, possibly mentioning z_i):

$$\exists z_0 \, q_0 = 0, \exists z_1 \, q_1 = 0 \ldots, \exists z_n \, q_n = 0 \vdash \mathit{false}$$

Exercise 14:
What do we do now?

We simply apply $\exists$L to get rid of the existential quantifiers:

$$q_0 = 0, q_1 = 0, \ldots, q_n = 0 \vdash \mathit{false}$$

Now that we are left with just equational assumptions, it is fairly easy to see what a counterexample witness could look like.

For example, if we could find polynomial cofactors g_i such that the following polynomial identity holds:

$$\sum_i g_i q_i = 1$$

These these cofactors g_i can be our proof witness for validity of the sequent (why?). At first glance, this procedure seems highly incomplete. Surprisingly, Hilbert's Nullstellensatz guarantees that these cofactors g_i always exist if the sequent is valid over the complex numbers.

Over the real numbers (which is our usual setting) we will need the real Nullstellensatz instead. The witness here is a little bit more involved and we will need both the polynomial cofactors g_i from before, as well as an additional sum-of-squares term ($\sum_j s_j^2$) on the RHS of the polynomial identity:

$$\sum_i g_i q_i = 1 + \sum_j s_j^2$$

Exercise 15:
Convince yourselves that the g_i and s_j satisfying this identity also witnesses the fact that the aforementioned sequent is valid.

Why do we bother with these witness procedures? The reason, as hinted before, is that QE algorithms are complicated beasts. For your entertainment, try out the following calls to QE (Resolve and Reduce are two possible choices for performing QE with Mathematica that KeYmaera X lets you use):

```
Resolve[0.333333333*3 == 1]
Resolve[0.33333333333333333*3 == 1]
Resolve[1 - 0.33333333333333333*3 > 0]

Reduce[(Sqrt[x])^2 == x]
Reduce[x^2 >= 0, {}, Reals]

Reduce[x/x == 1, {}, Reals]
Reduce[0/0 == 1, {}, Reals]
```

Exercise 16:
Can you think of how to use these to break KeYmaera X? It goes without saying that you should NOT exploit these in your final projects.

7 Practical Arithmetic Proving Advice

Here is a collection of useful QE advice, some of it possibly new. In addition to reading the below, look at the accompanying examples in recitation12.kya which prove quite slowly by `master` but quickly with a clever proof.

- **Hide Formulas.** This is the easiest way to speed of QE. If some assumptions are not relevant, then hide/weaken them away before calling QE.

  Note: Look at rec12assum for an example of how this could be done.

- **Tell KeYmaera X the argument.** Sometimes we need to prove simple facts about complicated terms. Say we have defined some complicated terms:

  $$\begin{aligned}\theta_1 &\equiv 100000000x^{200}y^{127}z^2w + 200x + 122wy^2\\ \theta_2 &\equiv 100000000x^{200}y^{127}z^2w + 200x + 122wy^2 + 1\\ \theta_3 &\equiv 100000000x^{200}y^{127}z^2w + 200x + 122wy^2 + 2\end{aligned}$$

  Clearly $\theta_1 < \theta_2 < \theta_3$. Due to the high degree polynomials this question is awful for QE and takes a while to prove. Yet, there is a simple argument: Regardless of the value of θ_1 we have $\theta_1 < \theta_1 + 1 < \theta_2 + 2$. We can make arithmetic *much faster* by turning $100000000x^{200}y^{127}z^2w+200x+122wy^2$ into a variable θ_1 and then asking QE. This is our way of telling KeYmaera X that the details of $100000000x^{200}y^{127}z^2w + 200x + 122wy^2$ are actually irrelevant to the proof, and the theorem is true no matter what θ_1 was.

  This is sometimes easier said than done. One way to "rename" a variable is to use the cut rule to introduce a universally quantified fact which will hopefully prove quickly by QE, and can then be used to recover our original conclusion.

  Note: Look at rec12transitivity for an example of how this could be done.

- **Expand special functions.** It is unclear what Mathematica does with the abs,min and max functions. Sometimes, it can get much faster if you first remove these in KeYmaera X before handing the real arithmetic question to Mathematica.

- **Instantiate Quantifiers.** In general, applying the rules $\forall L$ and $\exists R$ also greatly speeds up QE by removing a quantifier and/or quantifier alternation. Applying these rules takes creativity, though, because you have to pick an explicit value for the quantifier. If you pick the wrong one, you might end up with an unprovable subgoal.

  For example, if you are working with the solution to an ODE that has a domain constraint, you will have the domain constraint as an assumption, which includes a quantifier ($\forall x\ \phi$). For most problems that occur in practice, we only need to assume the constraint is true at the end (and maybe beginning) of an ODE.

- **Try Different Solvers.** As mentioned above, different solvers take fundamentally different approaches, so you might get better results by switching to a different one.

- **Try Lazy vs. Eager QE.** There are different ways that KeYmaera X can use a QE solver. The main two ways are *eager* QE vs. *lazy* QE. In *eager* QE we take whatever formula we have right now and (after hiding useless assumptions) hand it right to the QE solver. In *lazy* QE we wait as long as possible before calling the QE solver, meaning we apply all possible propositional reasoning steps first. As with the "which solver" question, there's not a universal winner here. Lazy QE will often produce a large number of proof branches, but each branch will be simpler, possibly much simpler if there are less variables in each branch. Because increasing the number of branches slows down proving, but simplifying each branch speeds up QE, either one can be faster. To do eager QE, use the `QE` tactic directly. To do lazy QE, expand out e.g., disjunctions in the antecedents before calling the tactic.

- **Search for Counterexamples.** When proving theorems, it is important to consider unpleasant possibilities such as the possibility that your conjecture is false. If QE is taking a really long time, you should re-evaluate whether your theorem is true. One way to do that (other than thinking hard) is to use KeYmaera X's counterexample search tool. Even if you are convinced the theorem is true, searching for counterexamples could help you discover that it is actually false, saving your proof effort.

- **Check Falsehood Manually.** Sometimes there is no substitute for "think about whether this is true". When you get to this point, keep in mind that there are some very commonly recurring mistakes that people make with arithmetic. You should be a little cautious any time you divide or take a square root: can you prove the divisor is non-zero or that you are always taking the root of a non-negative number? If not, QE will get stuck because the property is not even well-defined, let alone true. Note this is the case if you take quotients or roots *anywhere* in a sequent, not just in the parts that you think are interesting.

  Another common mistake is to forget an important assumption. When we reason informally about arithmetic, we often forget to mention basic assumptions such as the signs of different variables. These assumptions are essential when doing a computer proof, though. If you think a theorem is true and QE disagrees, double check all your assumptions and add some more assumptions to your model if necessary.

References

[Bro01] Christopher Brown. Improved projection for cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 32(5):447–465, 2001.

[Col75] George E. Collins. Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In H. Barkhage, editor, *Automata Theory and Formal Languages, 2nd GI Conference, Kaiserslautern, May 20-23, 1975*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183. Springer, 1975.

[Jir98] Mats Jirstrand. *Algebraic methods for inequality constraints in control.* PhD thesis, Linköping University, Department of Electrical Engineering, 1998.

[PQR09] André Platzer, Jan-David Quesel, and Philipp Rümmer. Real world verification. In Renate A. Schmidt, editor, *Automated Deduction - CADE-22, 22nd International*

Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings, volume 5663 of *Lecture Notes in Computer Science*, pages 485–501. Springer, 2009.

[Str00] Adam Strzeboński. Solving systems of strict polynomial inequalities. *J. Symb. Comput.*, 29(3):471–480, 2000.