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 (yongkiat@cs.cmu.edu)

1 Announcements

- Proposal grades released on Autolab.
- (Preliminary) Grand Prix schedule:
  `http://symbolaris.com/course/lfcps18-projects.html`
- Project deadlines:
  - Final Project (your code/models/proofs/etc.), December 6, 11:59PM
  - Term Paper (*scientific* paper describing the work done), December 7, 11:59PM
  - Presentation Slides, December 11, before Grand Prix

  Watch on Piazza for further announcements.

2 Theory 5

In recitation we discussed common mistakes on the assignments, but we will not give out solutions in the recitation notes. If you have questions, come talk to us.

3 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.

In this recitation, we will first take a high level overview of QE techniques before recapping some tips and tricks for speeding up QE which you have seen throughout the semester.

4 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? Eliminate variables (and quantifiers) yourself instead of leaving it to QE.

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?

5 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.

- *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:

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

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 1:
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 2:
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 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:

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

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

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 3:
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 4:
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 5:
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]
```

Note: In recitation we saw more complicated breakages too. KeYmaera X already tries to protect you against strange QE behavior. However, nothing beats checking the arithmetic yourself, especially that you have not accidentally divided by 0 or taken non-integer powers of a negative term.

Exercise 6:
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.

6 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:

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

  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

[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.

[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.