Lecture notes for: Theories: Linear arithmetic

Lindsey Kuper

October 16, 2019

These are lecture notes to accompany sections sections 5.1-5.2 (pp. 97-106) of Kroening and Strichman's *Decision Procedures: An Algorithmic Point of View*, second edition. See https://decomposition.al/CSE290Q-2019-09/readings.html for the full collection of notes.

Agenda

- Flavors of linear arithmetic
- The simplex algorithm for solving linear real arithmetic formulas

Flavors of linear arithmetic

Last time we looked at the theory of equality and uninterpreted functions, or EUF, and we talked about how one strategy for solving formulas with functions in it, like, say, addition or multiplication, is to turn those functions into uninterpreted functions and then use an EUF solver. This works well for many applications. But as we saw, we lose meaning when we do this, and sometimes we do actually want to be able to reason about arithmetic.

A linear arithmetic formula might be something like this:

$$2z_1 + 3z_2 \leq 5 \wedge z_2 + 5z_2 - 10z_3 \geq 6 \wedge z_1 + z_3 = 3$$

(I don't actually know if this is satisfiable; I'm just making something up.)

In linear arithmetic formulas, we can multiply variables by constants, but we can't multiply variables by each other. Hence the name "linear arithmetic".

If you want to be able to multiply variables, that's called nonlinear arithmetic. Decision procedures do exist for various nonlinear arithmetic theories, but those are beyond the scope of this lecture.

There are various linear arithmetic theories, depending on the domain that the constants and variables come from. If the domain in question is real numbers, then the theory is called *linear real arithmetic*.

In practice, though, when we speak of linear "real" arithmetic solvers, the numbers involved are *rational* numbers. And I'll follow that convention too: whenever I say "linear real arithmetic" you can assume that the numbers are actually rational unless otherwise specified.

So you can assume that that the constants in the input formula are all rational numbers. (In my example above, they're actually all integral numbers.) And you can assume that the domain of variables is rational numbers, too. If a linear arithmetic formula with rational constants is satisfiable, there will exist a satisfying assignment of variables to rational numbers. So "linear real arithmetic" is a misnomer, and we should really say "linear rational arithmetic" — but "linear real arithmetic" is what people say, so we'll follow that convention too.

(There are solvers that handle irrational numbers, but they're beyond the scope of this lecture, and for the software and hardware verification applications that we're concerned with, for the most part we don't need irrational numbers.)

What if the domain is *integers*? Then the theory is called linear integer arithmetic, sometimes also called Presburger arithmetic. (To be pedantic, it's the *quantifier-free fragment* of Presburger arithmetic.)

Q: By the way, which do you think is computationally harder to solve, linear real arithmetic formulas or linear integer arithmetic formulas?

A: Linear integer arithmetic formulas are harder to solve. I found this somewhat mind-bending when I learned about it, because I'm used to thinking that integers are easier to deal with than real numbers. I'm a programming languages person! I'm allergic to real numbers! I like my numbers to be integral (and ideally non-negative)!

But on further reflection, it makes sense that coming up with satisfying assignments that are integral would be harder than coming up with satisfying assignments that are rational. For instance, if the formula is

$$5z_1 + 3z_2 = 4$$

then it's pretty straightforward to come up with a satisfying assignment to z_1 and z_2 that's rational. We can assign them both 0.5, for instance. But if we need to come up with a satisfying assignment that's integral, we can't do it. So yeah, solving linear integer arithmetic formulas is an NP-complete problem, while solving linear real arithmetic formulas can be done, it turns out, in polynomial time.

The simplex algorithm for solving linear real arithmetic formulas

The algorithm that we're going to look at for solving linear real arithmetic formulas is called the *simplex algorithm*. It is, in fact, *not* a polynomial-time algorithm for solving this problem, although polynomial-time algorithms do exist. It's worst-case exponential. But it nevertheless turns out to be the algorithm that's most efficient in practice on real inputs.

So you may have heard of the simplex algorithm before, not necessarily in the context of SMT solving, but in the context of solving what are known as *linear programming* problems, or *LP* problems for short.

A standard introduction to LP problems is the “diet problem”. The idea is that you want to minimize the cost of the food you eat in a day, subject to constraints on how many calories you need to eat and what your nutritional requirements are.

To make it really simple, let’s assume there are three foods: corn, milk, and bread. Each food has a certain cost per serving and a certain number of calories per serving, and it provides a certain amount of, say, vitamin A:

- A serving of corn costs 18 cents, has 107 units of vitamin A, and is 72 calories.
- A serving of milk costs 23 cents, has 500 units of vitamin A, and is 121 calories.
- A serving of bread costs 5 cents, has 0 units of vitamin A, and is 65 calories.

This example is clearly a bit contrived, but let’s go with it.

So let’s say you want to eat between 2000 and 2250 calories in a day, and furthermore, you want to get between 5000 and 50,000 units of vitamin A per day.

So the question is, how many servings of each food should you eat to minimize the cost?

OK, so what do the constraints look like?

So let’s set up three variables, x_1, x_2, and x_3, which are the numbers of servings we eat of corn, milk, and bread, respectively. So the cost of a day’s food would be

$$18x_1 + 23x_2 + 5x_3$$

and that's the number we want to minimize. But we need to make sure we get at least 2000 calories and no more than 2250:

$$2000 \leq 72x_1 + 121x_2 + 65x_3 \leq 2250$$

and betwen 5000 and 50,000 units of vitamin A:

$$5000 \leq 107x_1 + 500x_2 \leq 50,000$$

So in general, a linear programming problem is the problem of minimizing or maximizing a linear function called the *objective function* — in this case the function is $f(x_1, x_2, x_3) = 18x_1+23x_2+5x_3$ — subject to some number of *linear constraints*. Here, the linear constraints happen to be inequalities, but they could also be equalities. For example, we could have the constraint that you eat *precisely* 2250 calories a day:

$$72x_1 + 121x_2 + 65x_3 = 2250$$

Or we could have the constraint that you eat precisely five servings of corn:

$$x_1 = 5$$

So, this is an optimization problem, right? It's a problem of minimizing or maximizing a function.

Here's another example. So say you're a farmer and you have L square kilometers of land that you could plant with either wheat or barley or some combination. You have F kilograms of fertilizer and P kilograms of pesticide.

Every square kilometer of wheat requires F_1 kilograms of fertilizer and P_1 kilograms of pesticide, while every square kilometer of barley requires F_2 kilograms of fertilizer and P_2 kilograms of pesticide.

Let S_1 be the selling price of wheat per square kilometer, and S_2 be the selling price of barley. If we denote the area of land planted with wheat and barley by x_1 and x_2 respectively, then profit can be maximized by choosing optimal values for x_1 and x_2.

So in other words, the function we want to maximize is

$$S_1 x_1 + S_2 x_2$$

subject to the constraints on how much land we have:

$$x_1 + x_2 \leq L$$

and on how much fertilizer and pesticide we have:

$$F_1 x_1 + F_2 x_2 \leq F$$

$$P_1 x_1 + P_2 x_2 \leq P$$

Q: Any other constraints we need?

A: We can't plant a negative area with a crop, so we have some constraints that x_1 and x_2 have to be nonnegative:

$$x_1 \geq 0$$

$$x_2 \geq 0$$

This is another optimization problem, right? A problem of maximizing a function. And the simplex algorithm is a recipe for doing that.

It's an algorithm that you can carry out by hand if you want to, or you can use an off-the-shelf LP solver that uses some version of the simplex algorithm to find an optimal solution. An open-source LP solver is GLPK, for example; a closed-source commercial one is Gurobi.

Q: But wait a second. This course isn't about optimization problems, it's about decision problems, right? We just want to know if formulas are satisfiable or not. We don't care if they're a little bit satisfiable or a lot satisfiable; we just want to know if they're satisfiable. So why is the simplex algorithm of interest to us?

As it's normally presented, the simplex algorithm has two phases. The first phase involves finding what's called a *feasible solution* to the problem, in which all the constraints are satisfied. For instance, finding a feasible solution to our diet problem would mean coming up with values for x_1, x_2, and x_3 such that all these linear constraints are true.

Once that's done, you can move onto the second phase of the algorithm, which involves iterating toward an *optimal* solution that minimizes the objective function.

Many presentations of the simplex algorithm assume that you already have a feasible solution, and they *only* deal with optimizing the objective function. Because I don't actually have time to teach it today, I'm going to ask you to trust me that once you know that a feasible solution exists, then it's possible to optimize it. So just assume, for today, that there exists an algorithm for turning feasible solutions into optimal ones.

For many real-world LP problems, assuming you have a feasible solution to begin with is a realistic assumption to make. For instance, suppose you're the farmer from the example problem above. Whatever you're already doing is a feasible solution: assuming you've set up the problem correctly, the amounts of fertilizer, pesticide, and land you're using already fall within the constraints, because it would be impossible for you to do otherwise (for instance, you can't use more land than exists, or put a negative amount of fertilizer on it). Of course, what you're currently doing is probably not *optimal*, which is why you need the simplex algorithm! But at least you have an initial feasible solution from which to start iterating toward an optimal one.

So, many presentations of the simplex algorithm focus on the second phase only. But we want to do the opposite: we only care about coming up with the feasible solution in the first place. In other words, we only care about phase one of the algorithm. And *that* problem really is a decision problem. It's a yes-or-no question: is there a feasible solution, or isn't there?

And by this point you might have realized that the problem of finding an initial feasible solution to a linear programming problem coincides exactly with the problem of finding a satisfying assignment to a linear real arithmetic formula that consists of a *conjunction of linear constraints*. For instance, for our diet problem, the formula would be:

$$(72x_1 + 121x_2 + 65x_3 \geq 2000) \wedge (72x_1 + 121x_2 + 65x_3 \leq 2250) \wedge (107x_1 + 500x_2 \geq 5000) \wedge (107x_1 + 500x_2 \leq 5000)$$

Determining whether this formula is satisfiable is exactly the same problem as determining if there is a feasible solution to the linear programming problem. (Notice that we left the objective function out of the formula, because we don't need it.)

Strangely enough, it turns out that you can use essentially the same algorithm that you use for phase two of the simplex algorithm, which is the optimization part, to carry out phase one, which is the decision part. You just have to turn the decision problem into an optimization problem.

OK. So in particular we're going to be talking about LP problems in a particular form.

maximize $c_1x_1 + c_2x_2 + \cdots + c_nx_n$

subject to some number of constraints that are all of the form

$a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b$

plus some special constraints called *nonnegativity* constraints on all the variables:

$x_1 \geq 0, x_2 \geq 0, dots, x_n \geq 0$

(The nonnegativity constraints are like the ones that ensure that we don't plant a negative area with wheat or barley.)

Now, *all* we care about is whether this problem has a feasible solution. We don't care about the objective function. So we're going to *throw away* that objective function, and instead we're going to make a slight tweak to our constraints and throw in a new variable called x_0.

$a_1x_1 + a_2x_2 + \cdots + a_nx_n - x_0 \leq b$

And we'll throw in a new nonnegativity constraint on x_0 also:

$x_0 \geq 0$

This new problem is called an *auxiliary problem*.

So x_0 can be zero, but it can't be nonnegative. Now the problem becomes: *minimize* x_0. We *want* x_0 to be zero, because we can see that the original problem has a feasible solution *if and only if* this new problem has a solution where $x_0 = 0$.

So we turned the decision problem of finding a feasible solution into the optimization problem of minimizing x_0.

Q: Now, the key question: Does the *auxiliary* problem have a feasible solution?

A: Yes, it does! We just set all the x_1 through x_n to 0 and then we make x_0

something sufficiently large. So we have a feasible solution to the *auxiliary* problem, and now we just need to iterate toward an optimal one where x_0 is zero. But we have a way to get that! We just use the algorithm that I mentioned earlier that I asked you to trust me exists!

By the way, my way of explaining this is different from what Kroening and Strichman do. They formulate the problem differently, and they show what they call the "general simplex" algorithm, where you don't use an objective function at all. But I want to show it in a way that I understand better, and is also closer to what's presented in the Reluplex paper which we'll read later in the course. Your mileage may vary.