Lecture notes for: Introduction to decision procedures, part 1

Lindsey Kuper

October 2, 2019

These are lecture notes to accompany sections 1.0-1.3 (pp. 1-14) 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

- What's a theory (informally)?
- More terminology
- Soundness and completeness; decidability
- Normal forms

What's a theory (informally)?

So, this course is about SMT solving, and the T in SMT stands for "theories". What is a theory?

We haven't defined this formally yet, and we aren't going to right now. But, informally, it's **the language that formulas can be written in.**

So, last time we looked at SAT formulas, like $x_1 \vee (x_2 \vee \neg x_3)$. When you're just talking about SAT solving, which is all we talked about last time, you're only concerned with formulas that look like this.

These SAT formulas are also called **propositional logic** formulas (I'll in-

terchangeably say “SAT formula” and “propositional logic formula” in this course). Propositional logic is the simplest theory. The formulas are built out of Boolean variables, conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$), and parentheses. So the inductive definition of a formula is: - a Boolean variable by itself is a formula (we call this an “atomic formula” or just an “atom”) - if ϕ is a formula, then $\neg\phi$ is a formula - if ϕ_1 and ϕ_2 are formulas, then $(\phi_1 \wedge \phi_2)$ is a formula - if ϕ_1 and ϕ_2 are formulas, then $(\phi_1 \vee \phi_2)$ is a formula

(If we followed these rules to the letter, then we would need to put more parentheses in, but the usual convention is to omit a lot of the parentheses and assume a certain order of operations.)

Also, this doesn’t say anything about being in CNF, but every propositional logic formula can be rewritten in CNF, so we usually just assume CNF. Last time, when we talked about SAT formulas we assumed they were in CNF.

So, propositional logic is one theory. But what if we want to have formulas with more *stuff* in them? Here’s one example from the book:

$$y_1 = y_2 \wedge \neg(y_1 = y_3) \implies \neg(y_2 = y_3)$$

What did we add to the language that formulas can be written in? Well, we have a couple of new symbols: implication and equals. We haven’t said anything about the domain that the variables come from; are they still Boolean variables like in propositional logic, or could they be something else, like numbers? We’ll talk more about that later.

So this formula is from the “theory of equality”, or “equality logic”. It’s perhaps not all that exciting of a theory, since it turns out that it’s not any more expressive than the theory of propositional logic. But it might allow for a more natural way to encode some problems.

Here's another formula:

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

So now there's a lot more stuff. We have *in*equalities; we have numbers; we've got addition and subtraction. It looks like our variables definitely come from some sort of domain of numbers now. This is what's called a *linear arithmetic* formula.

The "linear" in linear arithmetic refers to the fact that you're not allowed to multiply variables with each other. You can only add or subtract them.

Again, we haven't said what domain the variables come from. But if they come from the domain of real numbers, then we have what's often called the theory of linear real arithmetic. If they come from the domain of integers, that's often called the theory of linear integer arithmetic. Sometimes it's also called the theory of Presburger arithmetic.

Believe it or not, these kinds of formulas can be converted to plain old boring SAT formulas and then solved with a SAT solver. (A citation for this is "On Solving Presburger and Linear Arithmetic with SAT" (Ofer Strichman, 2002). Even though it makes sense that one decidable theory can be converted to another, simpler decidable theory — in the same way that any programming language can be compiled to machine language — it's still kind of mind-blowing to me that it's possible.) But that is not, in fact, what most solvers do. We'll talk more about this more as we get further along in the book.

So, I'm listing all these off not because I want to talk about any of the details of these theories today, because I don't want to do that today, but simply because I want to give you some kind of intuition for what a theory is, and introduce you to the idea that when I talk about a "formula", it may not necessarily be a propositional logic formula, like the formulas we talked about on

Monday, but it may in fact be a formula that comes from some other theory.

So we're going to define some more terms, and they'll overlap with terms that we defined last time, but some of the definitions will be more general now, because now we're talking about arbitrary formulas and not necessarily just propositional logic formulas.

More terminology

I mentioned before that a Boolean variable by itself is an "atom" in propositional logic. In general, regardless of the theory, an "atom" is a formula with no propositional structure. That is, it doesn't contain any $\wedge$, $\vee$, or $\neg$ connectives.

So, in the propositional logic formula $x \vee \neg y$, x and y are atoms. But in an equality logic formula, like $y_1 = y_2 \vee \neg(y_1 = y_3)$, the atoms are the equalities $y_1 = y_2$ and $y_1 = y_3$. So, the atoms of a formula depend on the theory.

This means we can also generalize another of our definitions from last time. Last time we said a "literal" is either a Boolean variable or its negation. That's the case for propositional logic, but in general, a literal is either an atom or its negation. So, $y_1 = y_2$ and $\neg(y_1 = y_3)$ are the literals in the aforementioned formula.

Last time we said that an "assignment" is a mapping of each Boolean variable in a SAT formula to either true or false. We can generalize that one, too. Say we have a formula, call it ϕ, from some theory – doesn't matter which one. We can now say that:

- an "assignment" of ϕ *from a domain* D is a mapping of ϕ's variables to elements of D. So, if D is the integers, maybe an assignment

would be like x = 1, y = -3.

- If the assignment gives mappings for all of the variables in the formula, it's a "full" assignment of that formula, and otherwise it's a "partial" assignment of the formula.
- And just like yesterday, a formula is "satisfiable" if there is some assignment that makes the formula evaluate to true, and "unsatisfiable" otherwise.

So this is all the same as yesterday, it's just that now the domain can be something other than just true and false.

Q: What do we call it when the formula is satisfied by *every* assignment?

A: Then we say that the formula is "valid" or a "tautology".

Q: What's the relationship between satisfiability and validity?

A formula ϕ is valid if and only if $\neg\phi$ is unsatisfiable. Which is convenient, because you can check the validity of a formula by checking if its negation is unsatisfiable.

So, if I have a validity checker, and I want to know the satisfiability of a formula, I just negate the formula and run the validity checker on the negated formula. If the answer I get back is "valid", then the formula was unsatisfiable, and if the answer I get back is "not valid", then the formula was satisfiable.

Conversely, if I have a satisfiability checker, and I want to know the validity of the formula, again, I just negate the formula and run the satisfiability checker on it. If the answer I get back is "unsatisfiable", then the formula is valid. If the answer I get back is "satisfiable", then the formula is not valid.

So, validity checking and satisfiability checking are interchangeable, in the sense that if you can do one then you can trivially do the other.

If we have an assignment, call it α, that satisfies a formula ϕ, then we write $\alpha \vDash \phi$. You can pronounce this "α satisifies ϕ". If "ϕ" is satisfied by every assignment — in other words, if ϕ is valid — then we can just write $\vDash \phi$ and leave off the α part.

Q: Should anything that I said just now give you pause, now that we're talking about arbitrary formulas and not just plain old SAT formulas?

A: Yes. The word "evaluate" should give you pause! We already know how to "evaluate" a propositional logic formula, because we can write down a truth table that tells us what the propositional logic connectives mean. And we said last time that "solving" a SAT formula is determining its satisfiability or unsatisfiability, and a "SAT solver" is a computer program that solves SAT formulas. SAT solvers use CDCL and fancy heuristics, but we could also do it on paper with a truth table, if we had a million years.

But do we know how to evaluate an arbitrary formula that comes from some theory? We can do it only if we have a definition of what formulas in that theory "mean", also known as a "semantics".

So "SMT solving" is determining the satisfiability or unsatisfiability of formulas for some arbitrary theory or combination of theories that you plug in a semantics for, and an "SMT solver" is a computer program that does SMT solving. And, just as with SAT, the semantics for a theory on paper might be pretty simple, in the way that truth tables for propositional logic are simple, but the computer program can be a lot more complicated, in the way that CDCL and fancy heuristics for computerized solving of propositional logic formulas can be. In this course, we'll be trying to understand some of the complicated parts.

Soundness and completeness; decidability

We talked a little about soundness and completeness last time with regard to paradigms of SAT solving, and now we can generalize that to SMT solving, too.

So, we're going to say that the "decision problem" for a given formula is determining whether or not it is valid. I'm following the book's definition here. But we could just as easily say that the decision problem is determining whether or not the formula is satisfiable, because they amount to the same problem.

So then we can define "decision procedure":

A "decision procedure for a theory T" is an algorithm that, given any formula of the theory T: - always terminates - when it returns 'Valid', the input formula is valid (**soundness:** the algorithm never lies and says the formula is valid when it is not) - when the input formula is valid, it returns 'Valid' (**completeness:** the algorithm never lies and says that the formula is *not* valid when it *is*)

And we say that a theory is "decidable" if and only if there is a decision procedure for it.

I think all the theories that we're going to talk about in this course are decidable theories. Once you know a theory is decidable, then the question becomes how hard the decision problem is for that theory. Most of the ones we're going to talk about have NP-complete decision problems.

Normal forms

It turns out that the notion of conjunctive normal form or CNF that we had from before also generalizes beyond propositional logic. Before, we just

said that a clause was a disjunction of literals, and a formula was in CNF if it was a conjunction of clauses. All that still applies, except now a literal can be any atom or its negation. So a formula is in CNF if it's a conjunction of disjunctions of literals, regardless of what the atoms might be.

There are some other useful normal forms to know about:

- a formula is in disjunctive normal form (DNF) if it's a disjunction of conjunctions of literals.
- a formula is in negation normal form (NNF) if negation is only allowed over atoms, and $\wedge$, $\vee$, and $\neg$ are the only allowed Boolean connectives.

And the DNF and CNF formulas are both subsets of the NNF formulas. The book also covers an algorithm for converting formulas to CNF, which is kind of interesting to learn about, but we won't go over it now.