Lecture notes for: Introduction to decision procedures, part 2

Lindsey Kuper

October 4, 2019

These are lecture notes to accompany sections 1.4-1.7 (pp. 14-23) of Kroening and Strichman's *Decision Procedures: An Algorithmic Point of View*, second edition. See http://composition.al/CSE290Q-2019-09/readings.html for the full collection of notes.

Agenda

- First-order logic
- What's a theory (formally)?
- Plans for the rest of the quarter

First-order logic

Q: Questions left over from last time?

So last time we talked about propositional logic, in which formulas just consist of Boolean variables, Boolean connectives ($\wedge$, $\vee$, $\neg$), and parentheses.

(By the way, as mentioned last time, "propositional calculus" is a synonym for "propositional logic", and there are other synonyms you could use. Maybe a logician or a philosopher or a historian of science would have more to say about what a calculus is versus what a logic is, but I'm a computer scientist, and to me “propositional calculus” and “propositional logic” mean exactly the same thing, and you should use whichever one you like better. I’m going to stick with “logic” because it’s one fewer syllable.)

So, we talked about propositional logic, and we talked informally about what a “theory” was, and we gave a couple of examples of theories. But today we’re going to make precise what we mean by “theory”, and to do that we’re going to use the framework of first-order logic.

(By the way, first-order logic is also called “predicate logic”, or “predicate calculus”, or “first-order predicate calculus”. More synonyms.)

First-order logic formulas consist of:

- variables
- **logical symbols**:
  - the standard Boolean connectives that we talked about before ($\wedge$, $\vee$, $\neg$)
  - quantifiers ($\exists$ and $\forall$) — these are new, but you shouldn’t worry too much about them, because we’re going to do away with them shortly
  - parentheses
- **nonlogical symbols**:
  - function symbols
  - predicate symbols
  - constant symbols

For instance, say we have an formula: $y = z \vee (\neg(x = z) \wedge x = 2)$.

Q: So what are the nonlogical symbols in this formula?

A: well, “=” is a nonlogical symbol. Which kind of nonlogical symbol is it? It’s a binary predicate symbol. So it’s a function that takes two arguments and returns true or false. When we write $y = z$, that can be thought of as syntactic sugar for *equals*(y, z).

And then "2" is a nonlogical symbol. Which kind of nonlogical symbol is 2? It's a constant. So there must be some domain of numbers that constants come from.

So, we've been talking about how a formula is satisfiable if there exists an *assignment* of values to its variables that makes the formula evaluate to true. But in order to evaluate a formula that has nonlogical symbols, you need more than just an assignment; you also need to know what the nonlogical symbols in the formula mean. In other words, you need an **interpretation** of the nonlogical symbols. Furthermore, in order to come up with an assignment, you need to know the domain that your variables come from, because they might not just be Boolean variables.

So, you really need three things in order to evaluate a formula:

- a domain ($\mathbb{N}$ or $\mathbb{Z}$, for instance)
- an interpretation of the nonlogical symbols in the formula
  - every function symbol maps to a function
  - every predicate symbol maps to a predicate
  - every constant symbol maps to a domain element
- an assignment of a domain element to each (free) variable in the formula

These three things together are what logicians call a **structure**.

So now, with our logician hats on, we can refine our definition of "satisfiable". Up until now we've been saying that a formula is satisfiable if there's some assignment that makes it true. But now we can say this:

- a formula is **satisfiable** if there exists a structure under which the formula is true.

In some sources, you won't see the assignment being included as part of the structure — they define a structure to just be a domain and an interpretation. In that case, you would say that a formula is satisfiable if there exists a structure and an assignment under which it's true.

Q: Questions about this so far?

A: So, I see one hitch in this so far, which is: maybe instead of asking if there's *any* structure that will make a formula true, what if you want to require that certain symbols mean certain things? For instance, in that formula that I wrote down earlier:

$$y = z \vee (\neg(x = z) \wedge x = 2)$$

Presumably I picked this particular "2" symbol because I want it to mean the number two. And presumably I picked this particular "=" symbol because I want it to mean "equals", or some sort of equivalence relation. Sure, someone could come up with their own interpretation of the symbol "=" and the symbol "2" that would make this formula be true under lots of different assignments, but that's not helpful to me in the real world where I need to verify that my software works. So "is this formula with all these symbols in it satisfiable?" is actually not all that interesting of a question per se. The more interesting question is, is the formula satisfiable given the constraint that particular symbols have particular meanings?

And this is where the idea of a theory comes in! Another way to ask the question that I just asked ("is the formula satisfiable given the constraint that particular symbols have particular meanings?") is, "is the formula satisfiable *in a given theory*?"

What's a theory (formally)?

So now that we have this concept of first-order logic and of structure, we can talk more precisely than we did yesterday about what it means to be a theory. In particular, a **theory** can be thought of as the particular first-order logic that you get when you specify:

- a particular set of nonlogical symbols (which is called a **signature**),
- a particular domain for variables and constants to come from,
- a particular interpretation for the nonlogical symbols that you somehow specify

So, for instance, maybe we define a theory as follows:

- we put the "=" symbol in our signature
- we pick a domain (say, the integers)
- and, importantly, we specify an interpretation for "=" that makes it mean "equals"

How do we do that? It turns out we can write down a set of axioms that constrain the interpretation of "=". In particular, there are three properties that are true of every equivalence relation. An equivalence relation has to be:

- reflexive: $\forall x. x = x$
- symmetric: $\forall x, y. x = y$
- transitive: $\forall x, y, z. x = y \wedge y = z \implies x = z$

Specifying a set of axioms is not the only way to define a theory, but it's a common way.

So now, finally, if we want to know if a formula is satisfiable *in a given theory*, we have to ask if there exists a structure that makes the formula true *and* *also* satisfies the axioms of the theory.

If a formula is satisfiable in a theory T, we say that the formula is T-**satisfiable**.

That is, we say that a formula is T-satisfiable for a given theory T if there exists a structure that makes the formula true and also satisfies the axioms of T.

Theories that are defined using this framework of first-order logic are called **first-order theories**. All the theories that we're going to be concerned with are first-order theories.

Q: So, when we started using the framework of first-order logic, there's one thing that came along for the ride that, for the most part, we don't want. What is it?

A: The quantifiers! The $\forall$ and $\exists$ quantifiers are part of the set of logical symbols that you can have in first-order logic, but for the most part, we don't want formulas that have these quantifiers in them. So, by moving to the framework of first-order logic, we gained the flexibility of being able to add our own new nonlogical symbols, but we also got more than we wanted, because we got the quantifiers too.

Unfortunately, we can't do away with the quantifiers just by restricting things to a particular theory, because a theory only restricts the nonlogical symbols. So, if we want a particular first-order theory but we don't want the quantifiers that came along for the ride, we have to say that we want only the **quantifier-free fragment** of that theory.

For the most part, we're going to be dealing with quantifier-free fragments of theories. For instance, last time we mentioned equality logic. It turns out that equality logic is just a name for the quantifier-free fragment of the theory that we just defined a minute ago.

Q: Questions about anything so far?

So, for everything we've just discussed, we've had our logician hats on, but for most of the rest of this course we're going to put our computer scientist hats back on. Which is to say, from now on, when we talk about theories, even though we *could* use the framework of first-order logic to talk about them, for the most part we're not going to, because for our purposes, doing that would be kind of awkward and pedantic and it wouldn't help us do what we actually want to do, which is to figure out how to design decision procedures for the theories that interest us — or how to best make use of existing decision procedures for theories that interest us. (By "theories that interest us", I mean the ones that let us naturally encode properties that we want to prove.)

Plans for the rest of the quarter

Yay! You made it through week one! Only nine more to go!

(look at course web page and discuss)