15-424/15-624 Recitation 1
First-Order Logic, Syntax and Semantics, and Differential Equations
Notes by: Brandon Bohrer (bbohrer@cs.cmu.edu)

1 Agenda

- Admin – Everyone should have access to both Piazza and AutoLab.
- KeYmaera X – Please install and load the example file **today**!

2 Formal Logic: Syntax and Semantics

Throughout this course, we will pay special attention to the precision and formality with which we reason about cyber-physical systems. The reason for this is simple, but important: Cyber-physical systems (CPS) are complex and subtle, which means mistakes in our thinking are all but guaranteed. By working with formal logic, we can make our reasoning so precise that we can get a computer to catch those mistakes, specifically by doing proofs in KeYmaera X. However, for many of you this will be your first time working with formal logic. We will start getting in the formal mindset by looking at how to make first-order logic (a.k.a. "regular logic") formal, then work our way up toward the logic that we use in this course. Along the way, we will see that the logic we take for granted is less obvious than we thought.

2.1 History: Motivating Semantics

Our exploration starts with one of the great wars of the 1940's: Syntax vs. Semantics. Since the beginning of the 20th century, logicians had preoccupied themselves with using formal logic to place mathematics on a solid foundation. For decades, they followed what is known as the *sacred* approach, where syntax (i.e. the way we write things down) is considered the most fundamental part of logic. When we define the syntax, we also define *proof rules* for manipulating the syntax, and we prove theorems by manipulating the syntax according to our rules. Any step that follows the rules is considered *justified*. For example, if we know that P is true and Q is true, we say that $P \wedge Q$ is true simply because that's how we defined the connective $\wedge$ to work.

Starting in the 1940's, Alfred Tarski spearheaded what is known as the *blasphemous* approach. We too are blasphemers, and will follow Tarski's approach in this course. Like Tarski, we believe that a proof cannot be justified simply by saying it follows the rules, but rather the rules too must be justified:

1. What if the rules and syntax don't mean what we think they mean? For example, I could have chosen $\phi \vee \psi$ to mean "ϕ and ψ are both true". Such a definiton of logic would be completely self-consistent, but utterly useless because you won't understand what I mean when I write down $\phi \vee \psi$: it doesn't agree with your intuition about $\vee$.

   To drive this home, here's an exercise for everyone:
   Exercise 1:
   Is the following formula valid (true for all values of x)?: $x \leq 0 \vee x \geq 1$

   The answer depends not just on the value of x but on the meaning of the word *all*. If *all* means *all real numbers* (which it generally does in this class), then the statement is not valid (we can pick $x = 0.5$ to make it false), but if *all* means *all integers* it would be true! This is why, as mathemeticians, we must be extremely careful with our words: our answers depend on it!

2. How do know that all things we can prove true actually should be true? There are many propositions that clearly *should* be false, such as $5 < 0$. We need to convince ourselves that we didn't give ourselves a way to prove that these *obviously false* propositions are true. Otherwise our idea of "true" would be quite ... illogical.

Questions 1 and 2 are both real issues for CPS. If I tell you that your car is safe, I need to be able to tell you what that statement really means. If I write a proof that says your car is safe, that statement had better be true. For this reason we follow in the footsteps of Tarski. The question remains: How?

2.2 Solution: Denotational (Tarskian) Semantics

Tarski's solution to the above problems is *denotational semantics*. If we wish to understand what formulas mean, we define a *denotation* function which takes in the *syntax* of a formula and gives us its *meaning* in terms of simpler mathematics whose meaning is already well-understood. This answers Questions 1 and 2 from Section 2.1:

1. If you give me a formula, I can now explain it in terms of mathematics you already understand (or if you don't understand it yet, you can learn it from other mathematicians).

2. I now have a new way to define what "true" means, independent of the proof rules I picked. Instead of just justifying my proof by "it follows the rules", I can now justify the *rules* themselves by saying "Everything you can prove is really true". This property is called *soundness*, and we will cover it more later.

What do the denotational semantic functions look like for first-order logic? This is the notation $\omega[\![\phi]\!]$ introduced in yesterday's lecture:

- Say we have a *term* θ, like $x^3 + xy$. The meaning (value) of a term θ is a real number $r \in \mathbb{R}$ that depends on a state $\omega \in \mathcal{S}$. For example if $\omega = \{x \mapsto 2, y \mapsto 1\}$ then $[\![\theta]\!]_\nu = 10$ for the above term. And because the definition of S says states can assign any real number to a variable, $x \leq 0 \vee x \geq 1$ formula is not valid, because it is not true in this state ω.
  Type: $[\![\theta]\!]_\nu : \textbf{Term} \ \rightarrow \ \mathcal{S} \rightarrow \ \mathbb{R}$

- Say we have a *formula* ϕ, like $x \geq 0$. The meaning of ϕ is the set of states where it's true. In this example, the relevant part of the state is just the value of x, so we can simplify matters and just say $[\![\phi]\!] = [0..\infty)$. We say ϕ is

  - *Valid* if $[\![\phi]\!] = \mathcal{S}$ (the set of all possible states). For example, $x < 0 \vee x \geq 0$ is valid because it's true for all x.
  - *Falsifiable* if it's not valid, i.e. $u[\![\phi]\!] \neq \mathcal{S}$. For example, $x < 0$ is falsifiable because it's false when $x = 1$.
  - *Satisfiable* if $[\![\phi]\!] \neq \emptyset$ All valid formulas are satisfiable, but satisfiable formulas might be falsifiable instead of valid. For example, the falsifiable formula $x < 0$ is also satisfiable when $x = -1$.
  - *Unsatisfiable* if it's not satisfiable $u[\![\phi]\!] = \emptyset$. Unsatisfiable formulas are always falsifiable and never valid. For example, $x > 0 \wedge x < 0$ is unsatisfiable, because no value of x can satisfy both conjuncts at the same time.

  When we prove formulas, we are proving that they are valid.

  Type: $[\![\phi]\!] : \textbf{Formula} \ \rightarrow \ 2^S$
  where 2^S is the power set of S.

The interpretation functions are defined inductively. Some of the cases, especially for terms, are so simple as to seem unnecessary. Consider the case for addition:

$$[\![\theta_1 + \theta_2]\!]_\omega = [\![\theta_1]\!]_\omega + [\![\theta_2]\!]_\omega$$

What is this saying? It may help to look at the types: The interpretation turns some syntax (a **Term**) into a number. This just says if we see two terms θ_1, θ_2 joined together with the + symbol, we compute their meaning by first figuring out what θ_1 and θ_2 mean and adding together the resulting real numbers. This case is boring precisely because the meaning of + on real numbers is already well-understood, but this precison will pay off for more complicated constructs (like differential equations).

Let's look at another deceptively simple case: universal quantifiers. The formula $\forall x.\phi$ should be true when ϕ is always true, no matter what we change x to. To make this easier to write down, we use the notation ω_x^d to say "the state ω, except change x to d"

$$[\![\forall x.\phi]\!] = \{\omega. \text{ for all } d \in \mathbb{R}, \omega_x^d \in [\![\phi]\!]\}$$

What's interesting here other than the new notation? The interesting thing is that there are uncountably many real numbers d, most of which do not have a nice finite representation on a computer. Writing down an arbitrary real number like π, e, or the Golden section would be quite impossible to do when typing a theorem into KeYmaera X (these numbers have infinite decimal expansions, and the only numeric literals we can type into KeYmaera X are finite), but it is quite easy to include these number in our semantics by just saying "all $d \in \mathbb{R}$". One way to help clarify the difference between syntax and semantics is that the meaning of a formula ϕ has to talk about infinite constructs, like real numbers, but the syntax gives us a finite way to write down those formulas. Since semantics is about reducing syntax to math, quantifying over infinite sets is no problem: math has been dealing with infinity for a long time.

Exercise 2:
Use the ω_x^d notation to write the semantics rule for $[\![\exists x\ \phi]\!]_\omega$.

Exercise 3:
Ponder the interaction between validity and universal quantification. How does the interpration of $\forall x.x \leq 0 \lor x \geq 1$ compare to that of $x \leq 0 \lor x \geq 1$? There's a pretty good chance this will be on Assignment 1.

Exercise 4:
Are the following formulas valid? satisfiable? unsatisfiable?

- $0 > 1$
- $x > x^2$
- $\exists y\ y < x$

3 Differential equations as a program

In $\mathsf{d}\mathcal{L}$, differential equations are programs: to run the program $x' = f(x) \,\&\, Q$ we evolve according to $x' = f(x)$, but only so long as Q holds true. Let's start by describing the semantics of ODEs informally, then make them formal. This will be by far the most complicated semantics of the day.

The semantics of an ODE tell us when the ODE can take us from some state ω to another state ν. So, when is it that we can get from ω to ν by following the differential equation and evolution domain specified by $x' = f(x) \,\&\, Q$?

Notice how we know the initial state, ω, making this an initial value problem. Given the initial state, we can now talk about the solution φ of the initial-value problem (IVP) given by $\varphi(0) = \omega, \varphi'(r) = f(\varphi(r))$. For the class of ODEs we consider, a solution φ will always exist for some time $t \geq 0$ (but not always $t = \infty$). [1] But what does φ look like? It's a function that will tells us what the state is at each moment of time.

$$\varphi : [0, t] \to \mathbb{R}^n$$

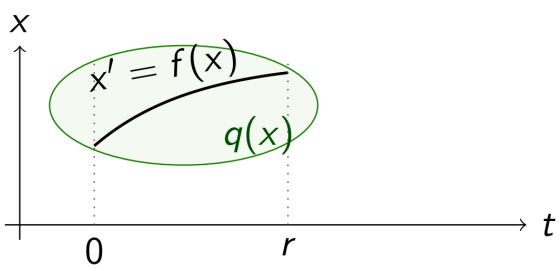
Since φ is the solution to our IVP, it must be a solution at time 0:

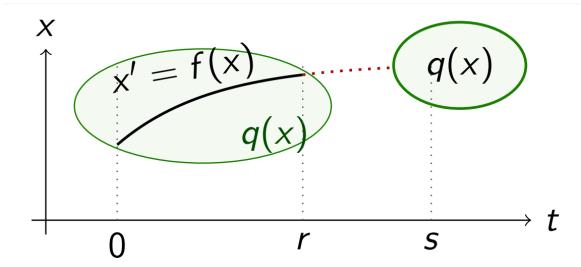
$$\varphi(0) = \omega$$

Since we transitioning from ω to ν, ν should be our final state. Since we chose t as the duration of the ODE, then the final state is also $\varphi(t)$, giving us:

$$\varphi(t) = \nu$$

It's not enough to evolve for time t, we also have a domain constraint Q, which we're not supposed to leave, not at time 0, time t, nor any time in between. That's what makes it a constraint. For example, the first image below is a valid execution trace of $x' = f(x)$ because it never leaves the domain constraint $q(x)$, but the second is not because it leaves the domain temporarily.





Not only that, but to be a solution to the IVP, the derivative of our solution φ must obey the ODE. We have special notation to say that φ obeys both the derivative constraint and the domain constraint. Specifically, we say that for all $r \in [0..t]$,

$$\varphi(r) \models (x' = f(x) \ \& \ Q)$$

As we will see for other programs in the next lecture, the formal semantics of an ODE is described as a set of all the pairs (ω, ν) where the ODE can take us from state ω to state ν. So, putting it all together, the formal definition is:

$$[\![(x' = f(x) \ \& \ Q)]\!] = \{(\varphi(0), \varphi(t)) \mid \exists \varphi \ \exists t \geq 0 \ \forall r \in [0, t].\ \varphi(r) \models (x' = f(x) \ \& \ Q)\}$$

[1] This is by the Picard-Lindeloeff theorem, because all terms we can write down in $\mathsf{d}\mathcal{L}$ are Lipschitz continuous.

where

$$\phi(r) \models (x' = f(x) \ \&\ Q) \iff (\phi'(r)(x) = f(\phi(r))) \wedge (\phi(r) \in [\![Q]\!])$$

Note that t is existentially quantified in this definition, which makes the semantics *nondeterministic*. Ignoring the edge cases, there are in general infinitely many values of t that we can pick. For example, once we have a duration $t > 0$, then any duration $r \in [0, t]$ is also a valid duration: An ODE can always choose to stop earlier than necessary.

Graphically, any time we have an execution trace (black line), we can take an initial segment and get another valid trace: