Recitation 1: Syntax and Semantics, FOL, and Differential Equations
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 Agenda

- Admin – Assignment 0 due today!
- Admin – Everyone should have access to both Piazza and AutoLab.
- Admin – Lab 0 and Assignment 1 have been released on course webpage. Material for assignment 1 will be covered in next week's lecture.
- KeYmaera X– Please install and try to load the examples from Lab 0 **today**!

Note: After each recitation, we will add some additional points that are relevant to the material covered in the actual recitation. These notes will appear in bold, much like this one.

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 (CPSs). The reason for this is simple, but important: CPSs are complex and subtle beasts, which means mistakes in our thinking are all but guaranteed.

By working with formal logic, we can make our operational requirements (such as safety and efficiency) for the CPSs that we have designed so precise that we can get a computer to catch those mistakes. Specifically for this course, you will learn to do proofs about CPSs in a theorem prover called 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 classical first-order logic over the real numbers ($\mathrm{FOL}_{\mathbb{R}}$) before working our way towards 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 Quick Reminder: Syntax

The syntax of terms is given as follows ($x \in \mathbb{V}$ is a variable, while $c \in \mathbb{Q}$ is a rational constant):

$$\theta ::= x \mid c \mid \theta_1 + \theta_2 \mid \theta_1 \cdot \theta_2$$

Notice that this syntax lets you write down polynomials over the variables only.[1]

[1]For modelling purposes, KeYmaera X also includes additional constructs, such as division and $\max(\cdot,\cdot)$.

The syntax of formulas in FOL$_\mathbb{R}$is given as follows, where $\sim$ is a comparison operator $\neq, =, \geq, >, <, \leq$.

$$\phi ::= \theta_1 \sim \theta_2 \mid \overbrace{\phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \rightarrow \phi_2 \mid \phi_1 \leftrightarrow \phi_2 \mid \neg\phi}^{\text{Propositional Connectives}} \mid \overbrace{\forall x\, \phi \mid \exists x\, \phi}^{\text{First-order Connectives}}$$

2.2 History: Motivating Semantics

Note: We did not cover the historical stuff in recitation. 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:
   Poll: Is the following formula true? $x \leq 0 \vee x \geq 1$

   Answer: This question does not really make sense.

   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?

2.3 Solution: Denotational (Tarskian) Semantics

Tarski's solution to the above problems is *denotational semantics*. If we wish to understand what terms and formulas mean, we define *denotation/interpretation* functions which takes in the *syntax* of a term/formula and gives us its *meaning* compositionally in terms of mathematical constructs whose meaning is already well-understood. This answers Questions 1 and 2 from Section 2.2:

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? Let us first motivate the types of these functions:
Note: In recitation, we used $State$ for the set of all states rather than $\mathcal{S}$, and Var for the set of all variables rather than $\mathbb{V}$. This just makes it easier to write on the board.

- A state $\omega \in \mathcal{S}$ is a function assigning real values to each variable:

  $$\omega : \mathbb{V} \to \mathbb{R}$$

  Here, $\mathcal{S}$ is the set of all states.

- Say we have a *term* θ, like $xy + 1$. We want the meaning of a term to be a real number $r \in \mathbb{R}$. It is easy to give a real number meaning to the sub-term 1, but how could we assign a real value to xy without even knowing what values x, y should take?

  Thus, the meaning (value) of a term θ depends on a given state $\omega \in \mathcal{S}$, which assigns values to the variables.

  For example if $\omega_1 = \{x \mapsto 2, y \mapsto 1\}$ then $\omega_1[\![\theta]\!] = 3$, but in state $\omega_2 = \{x \mapsto 0, y \mapsto 1\}$ then $\omega_2[\![\theta]\!] = 1$.

  Formally, the denotation function for a term takes a term and a state and returns a real, i.e., it has the following type:

  $$[\![\cdot]\!] : \textbf{Term} \to \mathcal{S} \to \mathbb{R}$$

Note: In recitation, we developed the semantics of terms with the notation $[\![e]\!]_\omega$. For uniformity and better alignment with the book and course material, we should have written it with ω on the left, i.e., as $\omega[\![e]\!]$ instead. The rest of these recitation notes have been updated to use this notation.

Some illustrative cases (developed on board, together with further notes below):

- The case for constants c is simple: $\omega[\![c]\!] = c$
- For variables x, we need to lookup the state: $\omega[\![x]\!] = \omega(x)$
- The remaining cases are defined *compositionally*, e.g., for $+$:

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

- Say we have a formula ϕ, like $x \geq 0$. There are many (equivalent) ways of defining the semantics of such formulas. One such approach, which you already saw in the lectures, is to define a *satisfaction relation*:

$$\omega \models \phi$$

Read intuitively, $\omega \models \phi$ if and only if formula ϕ is true in state ω.

In this recitation, we shall instead give a *set-valued* semantics for formulas. The meaning of ϕ shall be the set of states where it is true, so formally the denotation function for a formula will have the following type:

$$[\![\cdot]\!] : \textbf{Formula} \to \wp(\mathcal{S})$$

Here, $\wp(\mathcal{S})$ denotes the power set of $\mathcal{S}$, or in other words, the set of all subsets of $\mathcal{S}$. The two approaches are equivalent in the sense that $\omega \models \phi$ if and only if $\omega \in \phi$.

Some illustrative cases (developed on board, together with further notes below, compare to $\omega \models \phi$ definitions from class):

- The base cases for comparison operators $\sim$:

$$\omega \in [\![\theta_1 \sim \theta_2]\!] \iff \omega[\![\theta_1]\!] \sim \omega[\![\theta_2]\!]$$

$$\omega \models \theta_1 \sim \theta_2 \iff \omega[\![\theta_1]\!] \sim \omega[\![\theta_2]\!]$$

- Example compositional cases (defined with set operations):

$$[\![\phi_1 \land \phi_2]\!] = [\![\theta_1]\!] \cap [\![\theta_2]\!]$$

$$\omega \models \phi_1 \land \phi_2 \iff \omega \models \phi_1 \text{ and } \omega \models \phi_2$$

$$[\![\phi_1 \lor \phi_2]\!] = [\![\theta_1]\!] \cup [\![\theta_2]\!]$$

$$\omega \models \phi_1 \lor \phi_2 \iff \omega \models \phi_1 \text{ or } \omega \models \phi_2$$

  - The case for quantifiers:

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

$$\omega \models \forall x\, \phi \iff \omega_x^d \models \phi \text{ for all } d \in \mathbb{R}$$

- Notice how the same formula can be true or false in different states. We say that a formula ϕ is
  - *Valid* iff $[\![\phi]\!] = \mathcal{S}$ (the set of all possible states). For example, $x < 0 \vee x \geq 0$ is valid because it is true for all x.
  - *Falsifiable* iff it is not valid, i.e., $[\![\phi]\!] \neq \mathcal{S}$. For example, $x \leq 0$ is falsifiable because it is false in state ω_1.
  - *Satisfiable* iff $[\![\phi]\!] \neq \emptyset$ All valid formulas are satisfiable, but satisfiable formulas might be falsifiable instead of valid. For example, the formula $x \leq 0$ is satisfiable (in state ω_2) but not valid.
  - *Unsatisfiable* iff it is not satisfiable, i.e., $[\![\phi]\!] = \emptyset$. 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 *validity* of that formula.

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:

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

What is this saying? It may help to look at the types: the interpretation function for terms should take a piece of syntax (and state) and return a real number.

So this equation says that whenever we see two terms θ_1, θ_2 joined together with the $+$ symbol, we compute its 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 precision will pay off for more complicated constructs (like differential equations).

Most notably, the $+$ symbol on the left is a piece of syntax, while the $+$ symbol on the right is the usual addition operator over the reals.

Let's look at another deceptively simple case: universal quantifiers. The formula $\forall x\, \phi$ should be true when ϕ is true "for all" values of x. To make this easier to write down, we use the notation ω_x^d to say "the state ω, except it maps x to d".

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

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.

In fact, writing down a transcendental like π, e would be quite impossible 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 numbers 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 only gives us a finite way to write down those formulas. Since semantics is about reducing syntax to mathematics, quantifying over infinite sets is no problem: mathematics has been dealing with infinity for a long time.

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

$$[\![\exists x\, \phi]\!] = \{\omega \mid \text{for some } d \in \mathbb{R},\, \omega_x^d \in [\![\phi]\!]\}$$

Exercise 3:
Are the following formulas valid/satisfiable/unsatisfiable?

- $x \leq 0 \lor x \geq 1$
- $\forall x\, (x \leq 0 \lor x \geq 1)$
- $\exists y\, y < x$

Answer:

- $x \leq 0 \lor x \geq 1$ is satisfiable but not valid.
- $\forall x\, (x \leq 0 \lor x \geq 1)$ is unsatisfiable.
- $\exists y\, y < x$ is valid (and satisfiable).

Note: Make sure that you understand these answers, and especially the unsatisfiability of the second formula! Carefully study the definition of validity, satisfiability, and unsatisfiability if you have trouble with them.

3 Demo: KeYmaera X

Note: We went through using KeYmaera X to search for a counterexample for the first question of Exercise 3, and used it to prove the validity of the formulas in the remaining two (the second formula is unsatisfiable, so its negation is valid).

4 Differential Equations as Programs

For this course, we shall view differential equations as programs. To run the program $\{x' = f(x) \,\&\, Q\}$ we evolve (nondeterministically) along the ODE $\{x' = f(x)\}$, but only so long as domain constraint Q holds true.

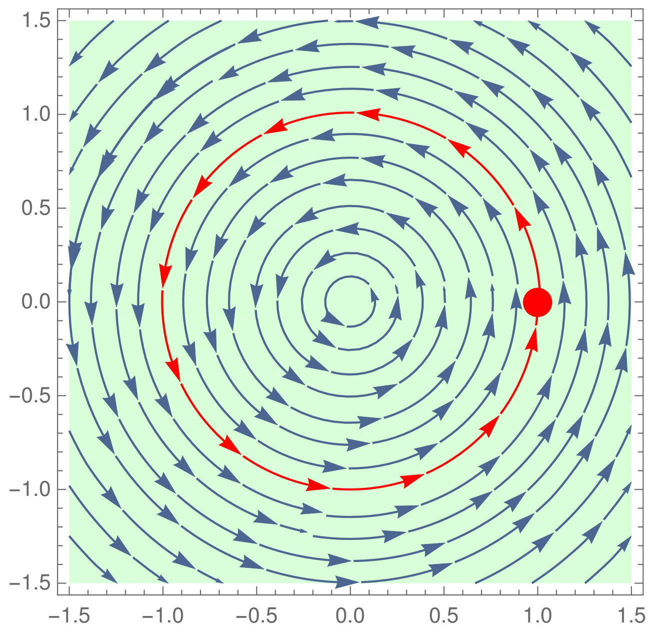
Let's start by describing the semantics of ODEs informally, then make them formal. This will be by far the most complicated semantics of the day.

4.1 Informal Semantics

The semantics of an ODE should tell us when the ODE can take us from an initial state ω to a final state ν. Consider the following ODE as a running example:

$$\{x' = -y, y' = x\}$$

A simple way to visualize this ODE is to draw its direction field: at each point on the xy-plane, draw a vector corresponding to the RHS of the ODEs evaluated at that point. Here, there is no domain constraint, so the ODEs are allowed to evolve within all of the xy-plane (shaded in green).



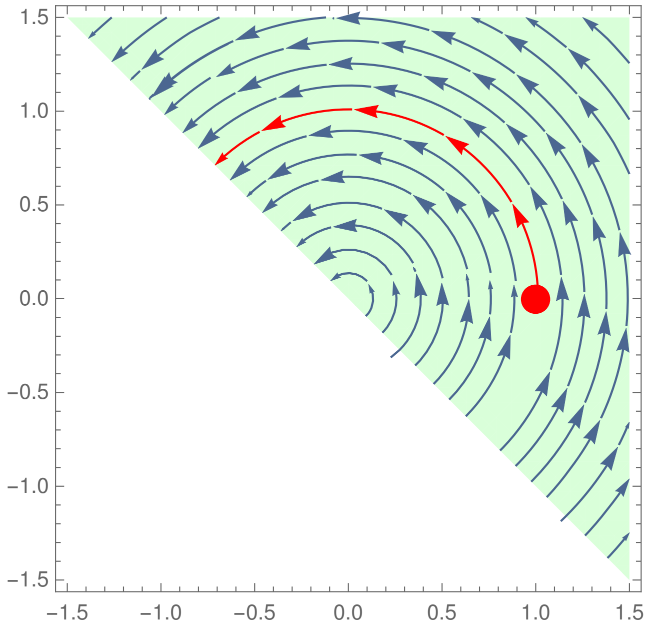
Now, if the initial state ω was the red point $(x = 1, y = 0)$, then the ODE program runs by following the red arrows of the direction field (tracing out the red circle shown above).

But when do we *stop* running the program? This is where nondeterminism comes in: the program can follow the red arrows for **as long as it likes**. In the running example, this means we can run the program from ω to any final state ν which lies on the unit circle.

The other way to *stop* the program is to add a domain constraint, e.g.:

$$\{x' = -y, y' = x \,\&\, y \geq -x\}$$

This can again be visualized on the xy-plane, but this time, we shall cut off the region $y < -x$, leaving us only with the shaded green region:



We can still run the ODE program for as long as we like, **but only while satisfying the domain constraint**. This means that we can now only reach final states ν can are somewhere on the red trajectory shown above.

4.2 Formal Semantics

Let us put the informal semantics down more formally. Specifically, we shall specify when we can get from an initial state ω to final state ν by following the program $\{x' = f(x) \,\&\, Q\}$.

Notice that we have the initial state ω and a system of differential equations $\{x' = f(x)\}$ which makes this an initial value problem (IVP). We can talk about a solution φ of this IVP obeying:[2]

$$\underbrace{\varphi(0) = \omega}_{\text{Initial state}}, \qquad \overbrace{\varphi'(r) = f(\varphi(r))}^{\varphi \text{ obeys the derivative constraints}}$$

For the class of ODEs in this course, such a solution φ always exists for some time $r \geq 0$ (but not always $r = \infty$).[3]

[2]See Lecture 2 slides, and Definition 2.6 of the textbook for a precise definition of when φ obeys the derivative constraints.

[3]In fact, the solution is also unique, and can be extended to its maximal interval of existence. This is by the Picard-Lindelöf theorem, because all terms we can write down in dL are locally Lipschitz continuous.

But what does φ look like? It is a function that will tells us what the state is at each moment in its domain of definition (the time interval $[0, r]$):

$$\varphi : [0, r] \to \mathcal{S}$$

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

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

Similarly, the state reached at the end of the solution at time r shall be our final state:

$$\varphi(r) = \nu$$

For φ to be the solution, it must satisfy the derivative constraints:

$$\varphi(t)(x') = \frac{d\varphi(t)(x)}{dt}(t) \text{ exists for all } t \in [0, r]$$

$$\varphi(t) \in [\![x' = f(x)]\!] \text{ for all } t \in [0, r]$$

Moreover, φ must stay inside the domain constraint Q for its entire duration.

$$\varphi(t) \in [\![Q]\!] \text{ for all } t \in [0, r]$$

Notice that the previous two clauses just expand the following from the lecture:

$$\varphi(t) \models x' = f(x) \wedge Q \text{ for all } t \in [0, r]$$

Finally, for all variables not mentioned in the ODE $\{x' = f(x)\}$, φ should hold their values constant.

$$\varphi(t)(z) = \varphi(0)(z) = \omega(z) \text{ for all } z \notin \{x, x'\} \text{ for all } t \in [0, r]$$

Writing all of the above down each time is a bit of a mouthful. We have special notation to say that φ obeys both the derivative constraint and the domain constraint. Specifically, for a function $\varphi : [0, T] \to \mathcal{S}$ with $T \geq 0$, we write $\varphi \models \{x' = f(x)\} \wedge Q$ iff for all $\tau \in [0, r]$:

1. $\varphi(\tau)(x') = \frac{d\varphi(t)(x)}{dt}(\tau)$ exists
2. $\varphi(\tau) \in [\![x' = f(x)]\!]$
3. $\varphi(\tau) \in [\![Q]\!]$
4. $\varphi(\tau)(z) = \varphi(0)(z)$ for all $z \notin \{x, x'\}$

Note: We skipped the remaining notes in recitation since we have not yet looked at the semantics of programs. The main takeaway should be the definition of $\varphi \models \{x' = f(x)\} \wedge Q$. Compare this definition with the one presented in lecture, and make sure you understand why they are equivalent.

We now have all the ingredients to give the formal semantics of an ODE. As we will see for other programs in the next lecture, the formal semantics of an ODE is described as a set of pairs (ω, ν) where the ODE can take us from initial state ω to final state ν.

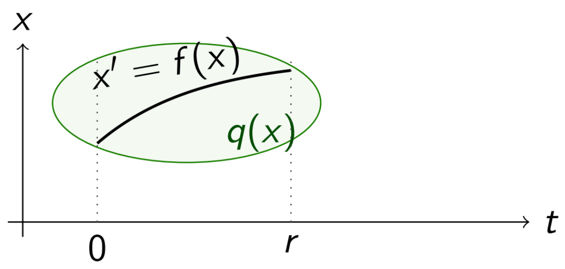
The formal definition is:

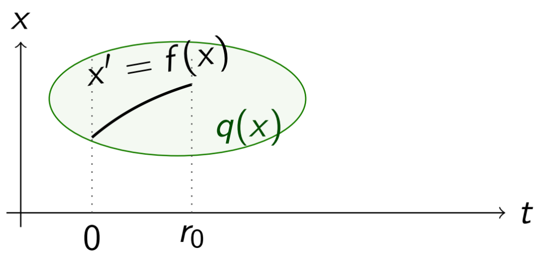
$$[\![\{x' = f(x) \,\&\, Q\}]\!] = \{(\omega, \nu) \mid \varphi : [0, r] \to \mathcal{S}, \varphi(0) = \omega, \varphi(r) = \nu, \varphi \models \{x' = f(x)\} \wedge Q\}$$

Note that the solution $\varphi : [0, r] \to \mathcal{S}$ is existentially quantified in this definition, which makes the semantics *nondeterministic*.

Indeed, by our definition, if $\varphi : [0, r] \to \mathcal{S}$ is a solution to the IVP where $\varphi \models \{x' = f(x)\} \wedge Q$, then we can truncate its domain of definition for any $0 \leq r_0 \leq r$ to obtain $\widehat{\varphi} : [0, r_0] \to \mathcal{S}$. The truncation $\widehat{\varphi}$ is also a solution to the IVP, and $\widehat{\varphi} \models \{x' = f(x)\} \wedge Q$.

Graphically, any time we have an execution (black line), we can take a truncation to obtain another valid execution:




x
x' = f(x)
q(x)
t
0
r0