Recitation 1: Syntax and Semantics, FOL, and Differential Equations
15-424/15-624/15-824 Logical Foundations of Cyber-Physical Systems

1 Announcements

- Please make sure that you have access to Diderot and that your notification settings are adjusted appropriately.
- Lab 0 will be released soon. This is a preparatory lab: install KeYmaera X and try it out ASAP so we can help diagnose any installation issues.

2 First-Order Logic: Syntax and Semantics

Throughout this course, we will pay special attention to the precision and formality with which we reason about Cyber-Physical System (CPS). The reason for this is simple: CPSs are complex and subtle beasts, which means (subtle) 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 a computer can help catch those mistakes. Specifically for this course, you will learn to do proofs about CPSs in a theorem prover called KeYmaera X.

For many of you this will be your first time working with logic, and especially with a logic for reasoning about programs. We will start this recitation by looking at the syntax and semantics of classical first-order logic over the real numbers. Later lectures will extend this logic with tools for reasoning about *hybrid programs*, our modeling language for CPSs for most of this course.

2.1 Syntax

A context-free grammar is a common notation for specifying the syntax, e.g., of programming languages and of terms and formulas. The syntax of terms is given as follows where $x \in \mathbb{V}$ is a variable and $c \in \mathbb{Q}$ is a rational constant:

$$e ::= x \mid c \mid e_1 + e_2 \mid e_1 \cdot e_2$$

Intuitively, e is a placeholder for a term in which you can replace using one of the constructors on the RHS. For example, the following chain of replacements (informally denoted $\Longrightarrow$) shows how the concrete term $x + (5 \cdot y)$ can be obtained by a sequence of replacements of the placeholders e:

$$e \Longrightarrow e_1 + e_2 \Longrightarrow x + e_2 \Longrightarrow x + (e_3 \cdot e_4) \Longrightarrow x + (5 \cdot e_4) \Longrightarrow x + (5 \cdot y)$$

The final concrete term $x + (5 \cdot y)$ is one where no placeholders are left. From now on, we will simply refer to terms as "e" with the understanding that it is a placeholder for a

concrete term like $x+(5\cdot y)$ that is obtained from the grammar. This grammar for terms lets you write down polynomials over the variables only.[1] Another intuitive way to understand this grammar is as a SML datatype:

```
datatype term =
  Var of variable_name
| Const of rat
| Plus of term * term
| Times of term * term
```

The concrete term $x + (5 \cdot y)$ is represented by `Plus(Var(x), Times(Const(5),Var(y))`.

Exercise 1:
Can you see why this syntax might be useful for implementation on a computer? Why might we not want to allow the `Const` constructor to take an arbitrary real number argument?

The syntax of formulas is given as follows, where $\sim$ is one of the comparison operators familiar from arithmetic: $\{\neq, =, \geq, >, <, \leq\}$:

$$P, Q ::= \overbrace{e_1 \sim e_2}^{\text{Atomic Comparisons}} \mid \overbrace{\neg P \mid P \wedge Q \mid P \vee Q \mid P \rightarrow Q \mid P \leftrightarrow Q}^{\text{Propositional Connectives}} \mid \underbrace{\forall x\, P \mid \exists x\, P}_{\text{First-order Connectives}}$$

Exercise 2:
Show how to obtain the formula $\forall x\,(x \leq 0 \vee x \geq 1)$ from the above grammar.

Exercise 3:
Is $x \leq 0 \wedge (x \geq 1 \vee)$ a concrete formula from the above grammar?
Answer: No, it is not well-formed! Indeed, it is impossible to arrive at this formula by a sequence of replacements.

2.2 Semantics

The syntactic terms that we have written down above are just that: strings that have been written down. This course takes the view that pieces of syntax do not have any *inherent* meaning. Instead, their meaning is given by the semantics.[2]

Exercise 4:
Based only on the syntax given above, is the formula $x \leq 0 \vee x \geq 1$ true?
Answer: This question does not really make sense until we have defined what $\vee$ means and what "true" means.

[1] For practical modeling purposes, KeYmaera X also includes convenient constructs, such as division and $\max(\cdot, \cdot)$, which make your models easier to express.

[2] If you have taken a course in constructive logic, this view might differ from what you have seen.

For example, we might have chosen $P \vee Q$ to mean "P and Q are both true". Such a definition of the operator $\vee$ could be completely self-consistent, but not very useful for anyone because it does not line up with our usual intuition about $\vee$. However, even if we used the usual definition of $\vee$, we still need to answer what "true" means e.g., what values do the variable x range over?

Exercise 5:
Suggest some domains for x where the formula would be true for all (resp. some) values of x in that domain. Are there domains where the formula is unsatisfiable (false for every possible value of x)?
Answer: Natural numbers (resp. real numbers). Unsat e.g., if domain of x is $\{0.5\}$.

We will assign precise meaning to terms and formulas with *denotation/interpretation* functions $[\![\cdot]\!]$. These functions take in the *syntax* of a term or formula and gives us its *meaning* compositionally in terms of mathematical constructs whose meaning is already well-understood.
Note: In recitation, we used $State$ for the set of all states rather than $\mathcal{S}$, 2^{State} for its powerset, 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* e, 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 e 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 $\omega_1[\![e]\!] = 3$, but in state $\omega_2 = \{x \mapsto 0, y \mapsto 1\}$ then $\omega_2[\![e]\!] = 1$.

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

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

  Some illustrative cases:

  - 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 $+$:

$$\omega[\![e_1 + e_2]\!] = \omega[\![e_1]\!] + \omega[\![e_2]\!]$$

It is important to distinguish the two different $+$ that occurs in this last case. On the left, $+$ is a syntactic operator that stitches together two terms e_1, e_2. On the right, $+$ is the familiar real-valued addition function that takes as input two real numbers and outputs their sum. Indeed, we could have defined the syntax of terms to be:

$$e ::= x \mid c \mid e_1 \oplus e_2 \mid e_1 \otimes e_2$$

Then this last case would be:

$$\omega[\![e_1 \oplus e_2]\!] = \omega[\![e_1]\!] + \omega[\![e_2]\!]$$

Exercise 6:
Suppose we added an extra term constructor $\times$ that stands for the "product". Define its semantics and clarify the type of each operator that appears in the definition:

$$e ::= \cdots \mid \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \times \begin{bmatrix} e_3 \\ e_4 \end{bmatrix}$$

Answer: We first have to decide what this product means. The $\times$ notation suggests that we might define it is as the 2D cross product (where $+, \cdot, -$ are the respective real valued functions):

$$\omega[\![\begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \times \begin{bmatrix} e_3 \\ e_4 \end{bmatrix}]\!] = \omega[\![e_1]\!] \cdot \omega[\![e_4]\!] - \omega[\![e_2]\!] \cdot \omega[\![e_3]\!]$$

Of course, we may have also chosen to define it like the dot product instead:

$$\omega[\![\begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \times \begin{bmatrix} e_3 \\ e_4 \end{bmatrix}]\!] = \omega[\![e_1]\!] \cdot \omega[\![e_3]\!] + \omega[\![e_2]\!] \cdot \omega[\![e_4]\!]$$

This emphasizes again that the semantics has the *final say* on what a piece of syntax means, rather than any preconceived notion we may have.

- Say we have a formula P, 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* (read as P is true in state ω):

$$\omega \models P$$

In this recitation, we shall give a *set-valued* semantics for formulas (Textbook Exercise 2.12). The meaning assigned to formula P will be the set of states ω where P is true. Thus, 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 $\omega \models P$ iff $\omega \in [\![P]\!]$.

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

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

$$\omega \in [\![e_1 \sim e_2]\!] \iff \omega[\![e_1]\!] \sim \omega[\![e_2]\!]$$

$$\omega \models e_1 \sim e_2 \iff \omega[\![e_1]\!] \sim \omega[\![e_2]\!]$$

- Example compositional cases (defined with set operations):

$$[\![P \wedge Q]\!] = [\![P]\!] \cap [\![Q]\!]$$

$$\omega \models P \wedge Q \iff \omega \models P \text{ and } \omega \models Q$$

Exercise 7:
Define the case for $P \vee Q$ (and $\neg P$, $P \rightarrow Q$).
Answer:

$$[\![P \vee Q]\!] = [\![P]\!] \cup [\![Q]\!]$$

$$\omega \models P \vee Q \iff \omega \models P \text{ or } \omega \models Q$$

- The case for quantifiers:

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

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

This definition for universal quantifiers is deceptively simple. What is interesting here other than the new notation? The interesting thing is that the semantics quantifies over *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 P has to talk about infinite constructs, like real numbers, but the syntax only gives us a finite way to write down those formulas.

Exercise 8:
Use the ω_x^d notation to write the semantics rule for $[\![\exists x\, P]\!]$.
Answer:

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

- Notice how the same formula can be true or false in different states. We say that a formula P is
  - *Valid* iff $[\![P]\!] = \mathcal{S}$ (the set of all possible states). For example, $x < 0 \lor x \geq 0$ is valid because it is true for all x.
  - *Falsifiable* iff it is not valid, i.e., $[\![P]\!] \neq \mathcal{S}$. For example, $x \leq 0$ is falsifiable because it is false in state ω_1.
  - *Satisfiable* iff $[\![P]\!] \neq \emptyset$. 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., $[\![P]\!] = \emptyset$. Unsatisfiable formulas are always falsifiable and never valid. For example, $x > 0 \land x < 0$ is unsatisfiable, because no value of x can satisfy both conjuncts at the same time.
- We will usually be interested in proving *validity* of formulas, i.e., that they are true no matter what states you start in.

Exercise 9:
Why might we be interested in proving validity (as opposed to satisfiability) of formulas involving CPS models?

Exercise 10:
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: **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.**

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

3 Demo: KeYmaera X

Note: We went through using KeYmaera X for Exercise 10. We found a counterexample for the first formula in Exercise 10, and then we proved the validity of the remaining two formulas (the second formula is unsatisfiable, so its negation is valid).

4 Differential Equations as Programs

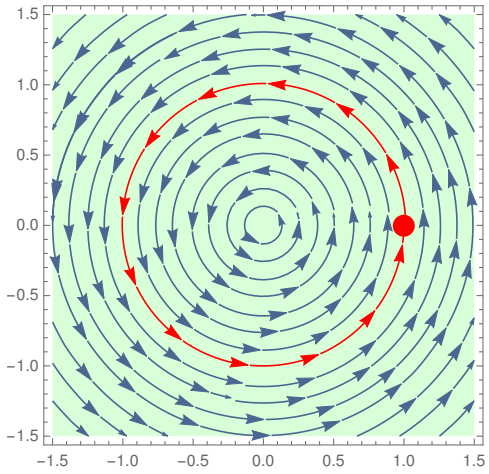
For this course, we shall view differential equations as programs. To run the program $\{x' = f(x)\,\&\,Q\}$ we evolve (nondeterministically) along the ODE $\{x' = f(x)\}$, but only so long as domain constraint Q holds true.

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. Let us imagine a robot that is driving on a circular track, modeled by the following ODE as a running example:

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

A simple way to visualize this ODE (and hence the robot motion) 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.



Exercise 11:
If the robot starts at the red point corresponding to initial state ω, which final states ν can the robot reach?
Answer: Its continuous evolution follows the red arrows of the direction field, tracing out the red circle shown above. We will take a nondeterminstic view of the robot's evolution so it can reach *any* state on the red circle.

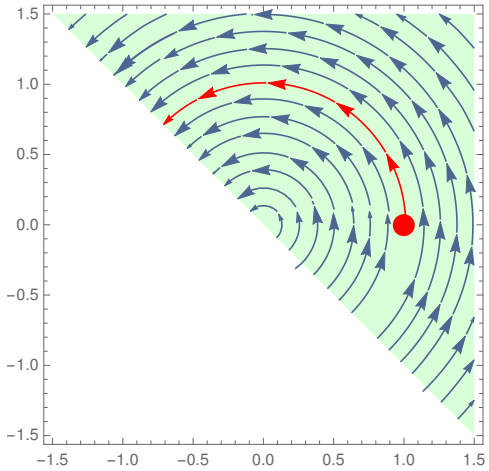
Exercise 12:
Suppose we want the robot to move clockwise instead. How might we modify the ODE to achieve this? What if we want the path to be elliptical?
Answer: To evolve clockwise, negate both equations $x' = y, y' = -x$. To evolve elliptically, one can "speed up" one of the axes e.g., $x' = -2y, y' = x$ will trace out an ellipse.

In ODE above, there is no domain constraint (entire plane shaded in green), so the robot is allowed to keep driving along the circle for as long as it likes!

Exercise 13:
Suppose there is a gate along $y = -x$ which the robot must unlock to cross. How could this be modeled?
Answer: With a domain constraint, that restricts the robot's continuous motion:

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

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:



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

Exercise 14:
Instead of restricting the ODE spatially as above, how might we restrict the robot to drive continuously for only a fixed time bound, say T seconds?
Answer: Remember that the ODEs we will work with are *autonomous* so if we need an explicit time dependency we will have to add a "clock" ODE $t' = 1$. We can then restrict t to be below the time bound so that the ODE can evolve for at most T seconds.

$$\{x' = -y, y' = x, t' = 1 \,\&\, t \leq T\}$$

Exercise 15:
Is there something that is not quite satisfactory about the above time-restricted ODE?
Answer: We have not quite set the clock correctly! What if it starts at $t = -10$? In later lectures, we will see how the clock can be set correctly.

4.2 Formal Semantics

To formalize the semantics, we shall specify precisely 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:[3]

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

For the class of ODEs in this course, such a solution φ always exists for some time $r \geq 0$ (but not always $r = \infty$).[4] 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 $\varphi : [0, r] \to \mathcal{S}$ with $r \geq 0$, we write $\varphi \models \{x' = f(x)\} \wedge Q$ iff for all $\tau \in [0, r]$:

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

[4] 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.

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 $z \notin \{x, x'\}$

Exercise 16:
Make sure you understand why the conditions listed above are equivalent to the 3 conditions given in the lecture.
Answer: The only change was to break up the conjunction $[\![x' = f(x) \wedge Q]\!]$ for ease of reading.

We say that state ν is reachable from initial state ω iff there is a function $\varphi : [0, r] \to \mathcal{S}$ where $\varphi \models \{x' = f(x)\} \wedge Q$ and $\varphi(0) = \omega$ and $\varphi(r) = \nu$.

Exercise 17:
Define $\psi : [0, \frac{r}{2}] \to \mathcal{S}$ to be identical to φ except it is defined on the smaller interval $[0, \frac{r}{2}]$. Does $\psi \models \{x' = f(x)\} \wedge Q$?
Answer: Yes. In particular, any truncation of φ to a smaller interval is also a solution. This means that not only can we reach final state $\varphi(r) = \nu$ from ω, we can also reach all of states along the continuous evolution i.e., the states $\varphi(\tau)$ for all $\tau \in [0, r]$! Notice how this lines up with the informal semantics given above. We will make use of this nondeterminism in subsequent lectures.