Recitation 2: Logic and Transition Relations
15-424/15-624/15-824 Logical Foundations of Cyber-Physical Systems

Notes by: Brandon Bohrer
Edits by: Yong Kiam Tan and (later) Katherine Cordwell (kcordwel@cs.cmu.edu)

1 Announcements

- Course schedule reminder: Betabot labs are due on **Fridays before recitation** and **you may not use late days for the Betabot submission!** Theory assignments are due on Thursdays.

2 Motivation and Learning Objectives

Safety properties of cyber-physical systems can often be stated in terms of *reachability*: "starting from an initial state, every state that the system can *reach* satisfies a given safety property". This informal prose translates into a dL formula of the form:

$$\psi \rightarrow [\alpha]\phi$$

where

- ψ describes the set of initial state(s) of the system,
- α is a hybrid program modelling the behavior of the system, and
- ϕ is the desired safety property.

In order to give a formal and unambiguous meaning to formulas of this form, we need to describe in a mathematically rigorous way what it means for a final state ν to be *reachable* from an initial state ω by running hybrid program α.

This recitation presents the semantics of hybrid programs and a corresponding diagrammatic representation to serve as a mnemonic device.

By the end of this recitation, you should know how to:

1. Visualize hybrid programs using transition diagrams.
2. Identify sources of non-determinism in hybrid programs.
3. Understand the transition relation for hybrid programs.
4. Avoid common modeling mistakes that occur when a hybrid program has an empty transition relation.

We will also cover some basic facts about KeYmaera X for Lab 1.

3 Quick Reminder: Syntax

As we did for $\text{FOL}_{\mathbb{R}}$, we shall start by defining the syntax of hybrid programs (and formulas) before looking at the informal and formal semantics.

The syntax of hybrid programs is given by the following grammar:

$$\alpha, \beta ::= x := e \mid ?Q \mid \{x' = f(x) \,\&\, Q\} \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^*$$

We shall extend the syntax of formulas from $\text{FOL}_{\mathbb{R}}$to include the two new *modal* operators: $[\cdot]P$ (pronounced "box") and $\langle\cdot\rangle P$ (pronounced "diamond"):

$$P, Q ::= \overbrace{\ldots}^{\text{Usual connectives of FOL}_{\mathbb{R}}} \mid [\alpha]P \mid \langle\alpha\rangle Q$$

4 Semantics of Hybrid Programs

We now need to give a formal semantics to hybrid programs, similar to what we did for the terms and formulas of $\text{FOL}_{\mathbb{R}}$. Recall that the meaning of a term (in a given state) $\omega[\![e]\!]$ is a real number, while the meaning of a formula $[\![P]\!]$ is the set of states in which that formula is true.

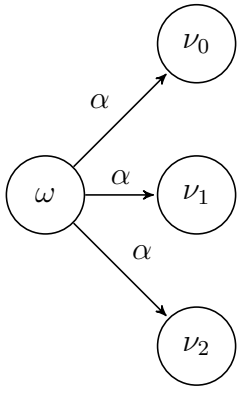
We give a transition semantics to hybrid programs:

$$[\![\cdot]\!] : \mathbf{HP} \to \wp(\mathcal{S} \times \mathcal{S})$$

Here, $\mathcal{S} \times \mathcal{S}$ is the set of all pairs of states, while $\wp(\mathcal{S} \times \mathcal{S})$ is its powerset.

In other words, for a given hybrid program α, $[\![\alpha]\!]$ is a set of pairs (ω, ν), where ω should be read intuitively as the "current" or "initial" state, while ν is a "final" state that can be reached by running hybrid program α from ω.

One useful way to understand the semantics is to use a transition diagram:



For example, the diagram above illustrates the transition relation:

$$[\![\alpha]\!] = \{(\omega, \nu_0), (\omega, \nu_1), (\omega, \nu_2)\}$$

In such diagrams, we use **solid** arrows for an actual transition of the program that we are considering, while **dashed** arrows will be used for the "intermediate" transitions that are used in the definition (we'll see this for ; and *).

We shall develop the definition of $[\![\alpha]\!]$ case-by-case, drawing informal transition diagrams to help us along the way.

4.1 Discrete Assignments and Tests

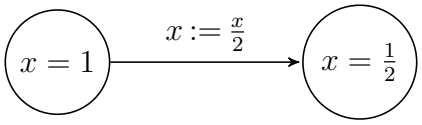
Let us start by dealing with the atomic cases of hybrid programs. Differential equations are also atomic hybrid programs, but their transition diagrams are a bit more complex so we will leave them for later.

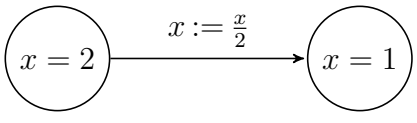
Assignments The formal semantics of an assignment is:

$$[\![x := e]\!] = \{(\omega, \nu) | \nu = \omega \text{ except } \nu(x) = \omega[\![e]\!]\}$$

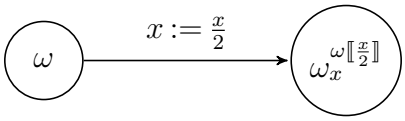
Exercise 1:
Write the above definition using the ω_x^d notation.
Answer: $[\![x := e]\!] = \{(\omega, \omega_x^{\omega[\![e]\!]})\}$

Consider the program $x := \frac{x}{2}$. In order to draw its complete transition relation we would need to draw (uncountably) infinitely many transitions:

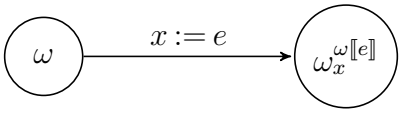




Fortunately, we can just use a representative initial state ω, and draw the final states that are reachable from it. In this case, the assignment program relates each initial state ω to exactly one final state:



More generally:



Tests The formal semantics of a test is:

$$[\![?Q]\!] = \{(\omega, \omega) | \omega \in [\![Q]\!]\}$$

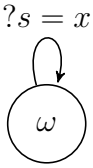
Now, if $\omega \in [\![Q]\!]$, then we can easily represent this transition with a self-loop:

Notice that the state is left unchanged. The test program merely checks that Q is true in state ω.

Exercise 2:
What if $\omega \notin [\![Q]\!]$?
Answer: Draw just the state itself with no transitions.

Note that when a test fails, the transition relation for $?Q$ from state ω will contain *no* transitions, not even to itself. We will return to this in a later section.

Concrete example for the test program $?s = x$:



if $\omega(s) = \omega(x)$

Exercise 3:
When do we want to use tests in our hybrid programs? Answer: There are various reasons tests are useful. They can be useful for directing control flow. Although they should not be used to vacuously ensure safety in controllers, they can be used to constrain away cases like division by zero. (Note: While division is not directly in the term syntax of dL, you can write a hybrid program to capture division by using nondeterministic assignment, which will be covered later in the course. Also, you will be allowed to use division directly in KeYmaera X, since it is so useful in modeling.)

4.2 Composing Hybrid Programs

Having just one hybrid program is rather boring: so far, we could only go from initial state ω to exactly one final state via assignment, or to itself via a test program.

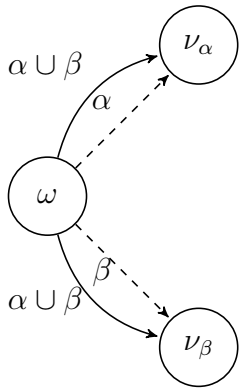
Now, let us add the hybrid program connectives: choice, sequential composition and looping.

Choice The formal semantics of a choice is:

$$[\![\alpha \cup \beta]\!] = [\![\alpha]\!] \cup [\![\beta]\!]$$

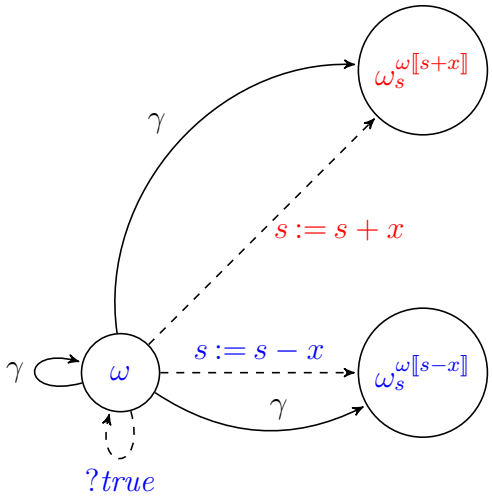
The choice operator allows us to pick between hybrid programs α and β, so its semantics is simply the union of transitions for either choice.

We now have our first source of non-determinism: from initial state ω, we can either choose to run α and reach final states ν_α or run β and reach final states ν_β.



Exercise 4:
Is there something inaccurate or perhaps unsatisfactory about the above picture?
Answer: It is now possible to reach many different states (or possibly 0 states) by running α, β, so we ought to represent the final states as sets of states rather than one state.
Note: We extended the diagram above to make ν_α, ν_β represent sets on the board in recitation.

Concrete example for the choice program $\underbrace{\overbrace{s := s + x}^{\alpha} \cup \overbrace{(s := s - x \cup ?true)}^{\beta}}_{\gamma}$:



Sequential composition The formal semantics of a sequential composition of two programs is:

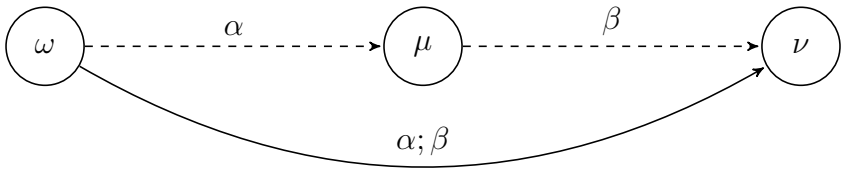
$$[\![\alpha;\beta]\!] = [\![\alpha]\!] \circ [\![\beta]\!]$$

Note: The $\circ$ operator is relational composition, whose order of operations is often confused with that for function composition.

When in doubt, we may just expand its definition:

$$[\![\alpha;\beta]\!] = \{(\omega,\nu) | (\omega,\mu) \in [\![\alpha]\!], (\mu,\nu) \in [\![\beta]\!]\}$$

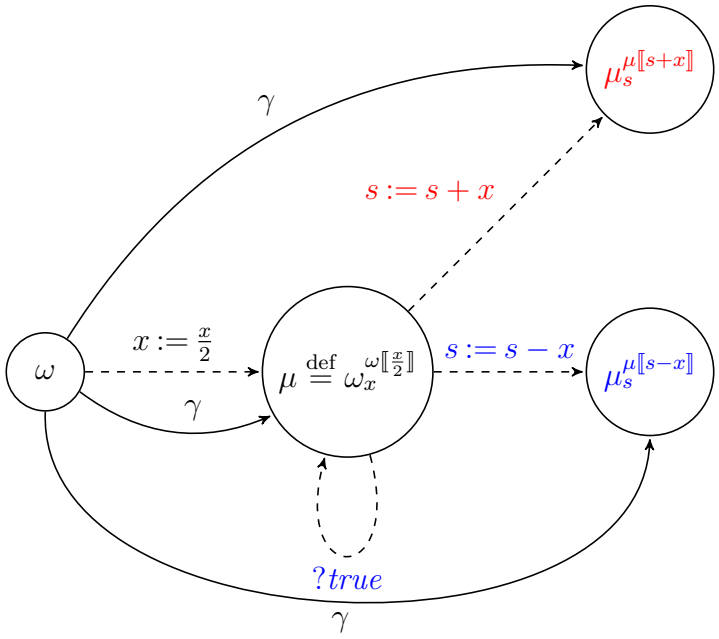
Thus, we can reach ν from an initial state ω, if we can first run program α to some intermediate state μ before running program β from μ to reach ν.



Notice the dashed transitions for α and β. It is not necessary that $(\omega,\mu) \in [\![\alpha;\beta]\!]$. Informally, the sequential composition "forgets" about the intermediate state μ once it has reached ν.

For a concrete example, let us extend the previous program with a sequence:

$$\underbrace{x := \frac{x}{2}; (s := s + x \cup (s := s - x \cup ?true))}_{\gamma}$$

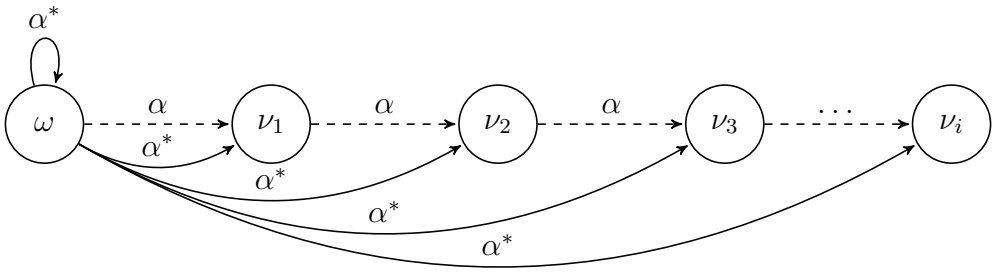


Loops The formal semantics of a loop is:

$$[\![\alpha^*]\!] = \bigcup_{n \in \mathbb{N}} [\![\alpha^n]\!]$$

where $\alpha^0 = ?true, \alpha^{i+1} = \alpha^i; \alpha$.

This is a (countably) infinite union: $[\![\alpha^*]\!]$ includes the transitions from $[\![\alpha]\!], [\![\alpha; \alpha]\!], [\![\alpha; \alpha; \alpha]\!], \ldots$.



Solid arrows corresponds to possible transitions from α^*. Observe that ω is *always* reachable from itself by following the loop for 0 iterations.

We now have a second source of non-determinism: whereas $\alpha \cup \beta$ presented (finite) choice between two options, we now have a (countably) infinite choice arising from the number of times the loop is followed.

Exercise 5:

Describe (informally) the values of x, s in states that can be reached by running the program γ^* (simplified to remove one of its branches).

$$\Big(\underbrace{x := \frac{x}{2}; (s := s + x \cup ?true)}_{\gamma}\Big)^*$$

Answer: The value of x is halved on each iteration, so if x_0 is the initial value, we can reach $x_0, \frac{x_0}{2}, \ldots,$ more generally $\frac{x_0}{2^i}$ for $i \in \mathbb{N}$.

On each iteration, we could choose either to add x to s or do nothing. Thus, at iteration i, the value that s can take is of the form $s_0 + \sum_{j=1}^{i} \delta_j \frac{x_0}{2^j}$, where δ is any indicator sequence e.g., $\delta = 0, 1, 0, 1, \ldots$.

Note: This means that s has the form of a (finite) binary expansion, i.e., of the form: $1.xxx \ldots xxx$.

4.3 Differential Equations

Finally, let us return to the differential equations $\{x' = f(x) \,\&\, Q\}$. Its formal transition semantics is:

$$[\![\{x' = f(x) \,\&\, Q\}]\!] = \{(\omega, \nu) \mid \varphi(0) = \omega, \varphi(T) = \nu \text{ for a solution } \varphi : [0, T] \to \mathcal{S} \text{ satisfying } \varphi \models \{x' = f(x)\} \wedge Q\}$$

This definition uses the notation introduced last week: $\varphi \models \{x' = f(x)\} \wedge Q$, which specifies that φ is indeed a solution of the ODE $x' = f(x)$ and that it stays within the domain constraint Q for its entire duration $[0, T]$. In addition, it requires that φ starts at the initial state ω and ends at the final state ν.

Why might this transition relation be harder to draw? This is due to the fact that there is a choice between *uncountably* many alternatives. **Note: This introduces non-determinism.** Observe that, by definition, if $\varphi : [0, T] \to \mathcal{S}$ is a solution to the differential equations, then we can truncate its domain of definition to obtain a shorter solution $\varphi_t : [0, t] \to \mathcal{S}$ for any $t \in [0, T]$. Thus, if we could reach the final state $\varphi(T)$ from initial state ω, then we could also have reached every state in between: $\varphi(t)$ for any $t \in [0, T]$.

Exercise 6:
Are the cases where the hybrid program $\{x' = f(x) \,\&\, Q\}$ does not have uncountable behavior?

Answer: If Q is not initially satisfied, then the set of transition relations of the hybrid program is empty. If Q is initially satisfied, but only in the initial state (and Q is immediately falsified when the program begins to run), then the hybrid program only has one transition relation (corresponding to running the hybrid program for 0 time): For example, consider running $\{x' = x' = -1 \,\&\, x \geq 0\}$ from an initial state ω where $\omega[\![x]\!] = 0$. Finally, as was brought up in class, if we have a differential equation that does not change the values of its variables, then there is finite behavior: For example, for the hybrid program $x' = 0$, the only

possible transition relation is from the initial state to itself.

Let us consider the simplest case of a time-bounded clock: $\{x' = 1 \,\&\, x \leq 5\}$

Exercise 7:
Suppose that the initial value of x in initial state ω was $\omega(x) = x_0$. According to the semantics of differential equations, what values can x take in states reachable from ω? What happens if $x_0 > 5$?
Answer: In any reachable state, x must take values in the interval $[x_0, 5]$. In particular, if $x_0 > 5$ then there are no reachable states, not even ω. Notice that the semantics of domain constraints (and also tests) can be slightly subtle.

Note: We skipped Exercise 7 in recitation.
Exercise 8:
Consider the differential equation $\{x' = 1 \,\&\, x \leq 1 \lor x \geq 2\}$. For example, this might model a car driving on a road where there is a chasm on the interval $(1, 2)$. Now, consider an initial state where $\omega(x) = 0$. How could we make sure that ω has a transition to final states where the value of x is greater than 2?
Answer: This is not possible with just the differential equation alone, because solutions to differential equations must be continuous and they must stay in the domain constraint at all times. One possibility suggested in class was to add a "teleporter" at $x = 1$ to $x = 2$, for example with the following loop:

$$\big(\{x' = 1 \,\&\, x \leq 1 \lor x \geq 2\}; (?x = 1; x := 2 \cup ?\mathit{true})\big)^*$$

5 Semantics of Box and Diamond

Now that we know the semantics of programs, we can formally define the meaning of the new modal connectives.

$$[\![[\alpha]P]\!] = \{\omega \mid \nu \in [\![P]\!] \text{ for all } (\omega, \nu) \in [\![\alpha]\!]\}$$
$$[\![\langle\alpha\rangle P]\!] = \{\omega \mid \nu \in [\![P]\!] \text{ for some } (\omega, \nu) \in [\![\alpha]\!]\}$$

Before we further explore the subtleties with these formulas, let us do a quick recap. A formula P is:

- *Valid* iff $[\![P]\!] = \mathcal{S}$ (the set of all possible states).
- *Satisfiable* iff $[\![P]\!] \neq \emptyset$
- *Unsatisfiable* iff it is not satisfiable, i.e., $[\![P]\!] = \emptyset$.
- *Falsifiable* iff it is not valid, i.e., $[\![P]\!] \neq \mathcal{S}$.

Note: Don't describe formulas as "invalid", as it is somewhat ambiguous. Instead, use the 4 descriptors above.

Let us recap these definitions in the context of some problems on Assignment 0. **Note: We skipped this recap in recitation.**

First, consider the formula $\forall y\, x < y$. For what states ω is the formula true? Let us unfold the definition:

$$\begin{aligned}\omega \in [\![\forall y\, x < y]\!] &\iff \omega_y^d \in [\![x < y]\!] \text{ for all } d \in \mathbb{R} \\ &\iff \omega(x) < d \text{ for all } d \in \mathbb{R}\end{aligned}$$

To make things more concrete, suppose that: $\omega(x) = 0, \omega(y) = 1$.

Unfolding definitions again: $\omega_y^d(x) = 0, \omega_y^d(y) = d$

So when does $\omega_y^d \in [\![x < y]\!]$? By definition, this is iff $\omega_y^d(x) < \omega_y^d(y)$, i.e., $0 < d$.

Thus, in order for the RHS to be true, we would require $0 < d$ **for all** $d \in \mathbb{R}$, which is certainly false and therefore, $\omega \notin [\![\forall y\, x < y]\!]$ for this particular concrete ω.

More generally, for any ω, a very similar argument would work to show that $\omega \notin [\![\forall y\, x < y]\!]$, because we would be requiring that $\omega(x) < d$ for all $d \in \mathbb{R}$, but no such number exists over the reals. Therefore, the formula $\forall y\, x < y$ is unsatisfiable because there is no ω such that $\omega \in [\![\forall y\, x < y]\!]$.

Next, let us add an existential quantifier: $\exists x\, \forall y\, x < y$. A very common mistake was to flip the order of quantification and read the formula as *for any y there exists an x such that $x < y$*. That is not what the formula means, as we can see by unfolding definitions:

$$\begin{aligned}\omega \in [\![\exists x\, \forall y\, x < y]\!] &\iff \omega_x^{d_x} \in [\![\forall y\, x < y]\!] \text{ for some } d_x \in \mathbb{R} \\ &\iff \big((\omega_x^{d_x})_y^{d_y} \in [\![x < y]\!] \text{ for all } d_y \in \mathbb{R}\big) \text{ for some } d_x \in \mathbb{R} \\ &\iff \big(d_x < d_y \text{ for all } d_y \in \mathbb{R}\big) \text{ for some } d_x \in \mathbb{R}\end{aligned}$$

Notice that the choice of d_x must be made **before** the choice of d_y. Moreover, we already saw earlier that there is no d_x such that $d_x < d_y$ for all $d_y \in \mathbb{R}$. Therefore, this formula is also unsatisfiable.

Now, let us think about validity and unsatisfiability in the context of the modal operators. **Note: Exercise 9 is meant to highlight the subtleties with empty transition relations and their interaction with the modal operators.**

For a program α with an empty transition relation, the box modality formula $[\alpha]P$ is vacuously valid because the box modality quantifies over *all* of the *no* runs of α.

Note: In class we highlighted the significance of the semantics of loops including α^0—that is, α^* never has an empty transition relation.

On the other hand, for a program β with an empty transition relation, the diamond modality formula $\langle\beta\rangle P$ is unsatisfiable because the diamond modality quantifies over *some* of the *no* runs of β. Since there are no runs, there cannot possibly be some run!

For a more familiar example, consider the following formula:

$$\forall r \left(r^2 = -1 \to 1 + 1 = 3\right)$$

Informally, this formula says that for all real numbers r such that $r^2 = -1$, $1+1 = 3$. Of course, $1+1$ is never equal to 3. Yet, the formula is valid because there are no real numbers r such that $r^2 = -1$.

Exercise 9:
Are the following formulas valid, satisfiable or unsatisfiable? For the formulas that are satisfiable, which states satisfy the formula?

1. $[?x > 0]1 = 0$
2. $[\{x' = 1 \,\&\, x \leq 5\}]x \leq 5$
3. $\langle\{x' = 1 \,\&\, x \leq 5\}\rangle 0 = 0$

Answer:

1. The formula is satisfiable (not valid). The postcondition $1 = 0$ is false in all states, which means we can only make the box modality formula true in states which have no transition. This is the case in initial states where $\omega(x) < 0$.
2. This formula is valid, and corresponds to a reasoning principle for differential equations called *differential weakening* that we will see later in the course. Intuitively, solutions of differential equations are required to stay in the domain constraint $x \leq 5$ at all times, and so it must be in $x \leq 5$ at the final state.
3. This formula is satisfiable (not valid). The reason is similar to the previous part: if $x \leq 5$ initially, then we can follow the differential equations for any duration and the postcondition $0 = 0$ will be true afterwards. However, if $x > 5$ initially, then there are no runs of the differential equation at all, **not even one of zero duration**.

Note: We did not go through the next exercise in class. They are meant as fun exercises for students who have a good grasp of the material. You are not required to understand these examples at this point in the course.

Exercise 10:
Same question but for the following formulas, where γ is the program defined in Exercise 4:

1. $[x := 1; ?x = s; \gamma^*]s < 2$
2. $\langle\{x' = x^2, t' = 1\}\rangle t \geq 5$

Answer:

1. The formula is valid. As we discussed earlier, the binary expansion of s is finite, and has the form $1.xxx\ldots xxx$, which implies that s is less than 2. In the theory of hybrid systems, this is known as Zeno behavior.

2. This formula is satisfiable (not valid). This is due to the fact that solutions of the differential equation can blow up in finite time. Concretely, consider an initial state where $\omega(x) = 1, \omega(t) = 0$, then the solution is $x(t) = \frac{1}{1-t}$, which has an asymptote when $t = 1$.

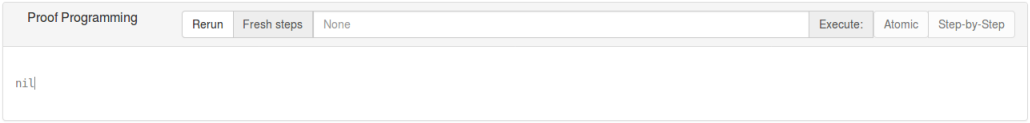
6 Demo: KeYmaera X

6.1 Tactic Proofs

Note: We rushed through tactic proofs in this recitation, but we will see more of them in class soon.

There are various ways of interacting with KeYmaera X. In the earlier labs, you will mainly be able to get by using its built-in proof automation.

Nevertheless, it is useful to start learning about more advanced ways of interacting with KeYmaera X. The main proof scripting language for KeYmaera X is called Bellerophon, and it can be accessed via the tactic editor:



When you click on various UI components in KeYmaera X, you will occasionally see new text appear in the Proof Programming box. The text that appears in this box is a textual representation of programs called *tactics*, which are used to automate proofs.

KeYmaera X automatically inserts tactics corresponding to buttons that you click on the UI. As the name suggests, it is also possible to script these tactics yourself by changing the text in the Proof Programming box. Tactics are useful for several reasons:

- They make repetitive proofs a lot easier. If your proof has lots of repetitive cases, you can prove the first case by clicking, then copy-paste the tactic on the rest of the cases. If you're lucky, the same tactic will prove them all. If you're not lucky, you can still often do the proof by making small changes to the previous tactic.

- They allow you to write customized proofs that might be faster than the built-in defaults.

- Tactics enable you to write fancy programs to prove things automatically by building them up from smaller tactics. For example, ODE solving and even the play button are just tactics. You probably don't want to write an ODE solver on the labs, but you may find some reusable snippets that make your proofs nicer.

Demo: replace `nil` with `master` and press "Execute: Atomic". This runs the "master" tactic, which is the main automation workhorse in KeYmaera X.

Tactics are great, but how do we learn how to write them? We will get a more thorough introduction in next week's recitation, but you can also learn them by watching the Proof Programming area to see what KeYmaera X fills in when you click on the UI. There are a lot of different tactics in KeYmaera X so this can help you learn all their names.

6.2 Debugging Models

Note: We demoed using counterexample search, but this was very rushed. You may wish to try out the debugging techniques yourself in Lab 1.

Let us try and use KeYmaera X to prove a property of a simple model of a ball being dropped from some height:

$$[\{x' = v, v' = -g \,\&\, x \geq 0\}]x \leq H$$

Recall that when we prove formulas, we are proving *validity* of that formula, but the above formula is clearly not true in all states.

Exercise 11:
The model is missing various assumptions on the initial state, suggest initial assumptions for the model.

- What if the ball was dropped from a height higher than H? (Add precondition: $x = H$ or $x \leq H$)
- What if the ball was thrown up in the air? (Precondition $v \leq 0$)
- What if gravity goes up? (Precondition $g \geq 0$). Remember that g is just a variable, not anything special, so we need to say if we want it to have some special property.