15-424/15-624 Recitation 3
Did you prove what you meant to prove?

@Yesterday you saw how to do proofs about hybrid programs. Now you know how to do a proof and even better, KeYmaera X will make you sure you only prove true things[1], so life is great, right? If you want to make sure your CPS is safe, you can just type it into KeYmaera X and as soon as it tells you you've finished your proof, you know your CPS is safe, right?

Wrong!

How to do sound proofs is only one part of theorem proving! There is another problem, which is at least as important and often overlooked: Did you prove what you meant to prove? That will be the subject of today's recitation. Remember: Software can tell us whether we did a valid proof, but it can't tell us whether the theorem we proved was useful or not. That's a problem for humans to solve, so whenever we do a KeYmaera X proof we need to be very careful that we picked a good theorem and a good hybrid program to model our CPS.

1. **Example Hybrid Programs:**

   At the end of the day, "did you prove what you meant to prove" is asking: Do you understand what your hybrid programs and formulas really mean? Let's test our understanding by looking at a few similar, but different hybrid programs.

   What's the difference between the following hybrid programs α, β, and γ?

   $$\alpha \equiv \{x' = v, v' = a \;\&\; v \geq 0\}$$

   $$\beta \equiv \{x' = v, v' = a \;\&\; v \geq 0\}; ?(v = 0) \quad \text{(Bad idea! See below.)}$$

   $$\gamma \equiv t := 0; \{x' = v, v' = a, t' = 1 \;\&\; t \leq T\}; ?(t = T)$$

   $$\delta \equiv t := 0; \{x' = v, v' = a, t' = 1 \;\&\; t \leq T\}$$

   $\phi(x, v)$:

   Now let $\phi(x, v)$ be a property that holds after (or in some cases during) the execution of each of these hybrid programs. In the remainder of these notes, I will use ϕ and $\phi(x, v)$ interchangeably. We write ϕ so that it explicitly depends on state variables x, v because these are the only variables that represent continuously-changing physical state. Most often the properties we want to prove are focused on the continuous state variables of the system. That doesn't mean other variables won't also be used, but the state variables are crucial.

   $[\alpha]\phi(x, v)$:

   Suppose you have found a proof of property $[\alpha]\phi(x, v)$. This means that $\phi(x, v)$ holds at the end of every run of the hybrid program. Since α can stop at all possible times such that the evolution domain $v \geq 0$ still holds, we've actually ensured that property $\phi(x, v)$ holds throughout the nondeterministic evolution. This is great, since we almost always want to prove properties about the system for the entire time it runs, rather than just when it stops.

[1] Unless you find and exploit soundness bugs, which I encourage you to do because it's fun.

Important point: When we say a system is *safe*, it must be safe *at all times*.

It is not useful if, for example, our car hits another car at time 1 but we then prove at time 2 we are no longer hitting the car. Whenever you write down a program α, it needs to be the case that $[\alpha]\phi$ really means ϕ is true at all times. We usually don't talk about this explicitly, but think about the nondeterminism that comes from ODEs: an ODE can run for any amount of time, so really this formula does mean safe at all times! Cool!

$[\beta]\phi(x,v)$:

Now suppose you have found a proof of property $[\beta]\phi(x,v)$. Again, this means that $\phi(x,v)$ holds at the end of every run of β; HOWEVER, some runs of β have been omitted, specifically whenever the velocity does not end with value exactly zero. This means, first, that $\phi(x,v)$ only holds at the end of the run, not necessarily throughout. But it has the bad and unintended effect that if you happen to have set your acceleration to be a positive value, and if velocity starts positive, too, then your system will never brake to a stop, so it is excused from satisfying property $\phi(x,v)$. In general, it is a bad idea to add tests that guard on state variables (like x, v in this example), because those are the variables that we want to prove properties about. Or to put it differently, $?\phi$ is saying "I assume ϕ is gonna be true right now". Unless you feel comfortable making that an assumption without proving it, you should not write down $?\phi$. The reason we use ? notation is because you should always question whether this is an okay place to use a test!!!

To really drive it home, let's prove that $[\beta]v = 0$ is true all the time, even when a is positive, and our proof won't even consider the ODE at all.

Major Red Flag: If you can prove a safety property about your system and that proof *totally ignores* some significant part of the program, then your program, theorem, or both are wrong.

In this proof I'm going to use a nonstandard axiom, $(P \to P) \leftrightarrow true$. Normally you shouldn't do this in your assignments (and you won't need to do it), but I'm honestly doing a pretty weird proof here for demonstration purposes, that's why it came up. You should believe this axiom is true, and you could prove it with a semantic argument like we'll use to prove other axioms later in the recitation.

$$\begin{array}{l}*\\[?]\\[;]\end{array}\dfrac{\dfrac{\dfrac{[\{x'=v,v'=a\ \&\ v\geq 0\}](true)}{[\{x'=v,v'=a\ \&\ v\geq 0\}](v=0\to v=0)}}{[\{x'=v,v'=a\ \&\ v\geq 0\}][?(v=0)]v=0}}{[\{x'=v,v'=a\ \&\ v\geq 0\};?(v=0)]v=0}$$

a

Ok at this point we should be very afraid. What does the top of this proof say? What is says is all we have to prove is $[\{x'=v,v'=a\ \&\ v\geq 0\}](true)$, meaning the trivial formula $true$ is always true after we ran the program $\{x'=v,v'=a\ \&\ v\geq 0\}$. Well, true is always true no matter what, regardless of whether we ran the ODE first or not. This is the red flag in action: Your proof should require you to take advantage of the fact that you had physics. Here all we had to do was prove something that is true *all the time*, regardless of physics.

Note: This paragraph is advanced material, it's just here for completeness, don't worry if you didn't get it in recitation. You will not need it on the assignments. How do we finish this bogus proof, by the way? Well there's a rule called G (Gödel generalization) that says if a formula is valid (true all the time), it's still valid if we stick a program in front of it.

$$\text{G}\ \dfrac{\text{true}\ \dfrac{*}{true}}{[\{x'=v,v'=a\ \&\ v\geq 0\}](true)}$$

So that would be how we get rid of the ODE. And then *true* is trivially true all the time, as the "true" rule tells us. But don't worry about the G rule too much. It's only relevant when the program really doesn't matter at all, and as I just suggested, if your program doesn't matter at all you probably have a mistake in your model.

$[\gamma]\phi(x,v)$:

In this case, we still have a guard, but it is on time instead of on the system state. The variable t has simple and well defined dynamics, so we aren't too worried about it exhibiting unexpected behaviour. Additionally, forcing the evolution to stop only after it has evolved for some minimum time is a behaviour we can actually build into the real cyber-physical system. Compare this to trying to build a system that has to force the velocity of a robot to be zero. Now, even though this is a system we can actually implement, we still have to be careful interpreting the theorems we prove about it. Because we have disallowed runs that evolve for less time than T, the property is no longer guaranteed to hold at those times.

Important point: When we prove $[\gamma]\phi(x,v)$, we are **NOT** proving that it is safe at all times t! Sometimes this still might be an interesting theorem in its own right, for example maybe if we wanted to prove that T is exactly how long it takes us to finish some task like braking. But it is almost always not what we want when we prove safety!

$[\delta]\phi(x,v)$: If we removed the test, we get program δ, which says the property has to hold throughout all durations up to T. Typically, these programs are repeated using α^*, which means that at most time T will pass before the program loops again. The idea is that in real systems, the control loop is guaranteed to run every T time units, thus keeping the CPS safe. It's also possible that maybe it runs a bit faster than expected, so we also consider the case where it takes less than T time, but adding this extra flexibility rarely makes the proof any harder. This is all fine and good because we're proving safety for all times t and because it's something we can really implement. We will want to make sure that there really exists a bound on the time T, but that's an operating systems problem: lots of people have written operating systems that can provide this guarantee. And without a bound on how often the controller must execute it would be impossible to guarantee any safety properties!

Executive Summary:

Suppose we have found proofs for $[\alpha]\phi$, $[\beta]\phi$, and $[\gamma]\phi$. Then, property ϕ holds *throughout all* executions of α. Property ϕ only holds at the *end of some* (but not all) executions of β. And property ϕ holds at the *end of all* executions of γ.

Generally we want to prove things throughout all executions of a hybrid program, so we design our HPs to look like α. Sometimes we want to prove properties that hold only at the end of a hybrid program, so we design our HPs to look like γ. We NEVER want design to design programs with nonexhaustive tests on state variables like β. Why? Because these tests constitute hidden preconditions. Put preconditions where they belong: in the safety theorem. For example, if we wanted to prove an efficiency theorem that says we reached our destination by the time $v = 0$, we could write $[\alpha](v = 0 \to x = \text{destination})$, and then the theorem statement makes it obvious that our theorem only tells us something about what happens if $v = 0$. This has exactly the same meaning as $[\beta]x = \text{destination}$[2], but there is a very important difference in readability. Furthermore, when we write the model like β, it becomes totally impossible to write safety theorems, just efficiency theorems. When we write the model like α it's easy to write both.

2. **Soundness.** What does soundness mean for an axiom? Soundness means that all instances (e.g. all choices of α, β, P, and Q) are valid. Why do we want that? Because we need to use these axioms to do proofs, and when we encounter them in a proof we'll want to use them as valid formulas for lots of different α's and β's.

[2]How do you know they're the same? By the axiom $[?\phi]\psi \leftrightarrow (\phi \to \psi)$

Here goes a quick, direct, formal proof of the $\cup$ axiom.

$$[\alpha \cup \beta]\phi \leftrightarrow [\alpha]\phi \wedge [\beta]\phi$$

The axiom is of the form $\phi_1 \leftrightarrow \phi_2$. The idea is to apply the semantics definitions of the formulas and programs to one side, say ϕ_1 until we reach a clear understanding of what the formula means (in a combination of math and English). Then we shuffle that understanding around a bit to get the meaning of ϕ_2, and then syntactically rebuild ϕ_2 from the semantics.

Let's try it! To prove the axiom is sound, we must show it holds for all states ν. To prove an equivalence, we have to prove implication from both sides. Let's start with $[\alpha \cup \beta]\phi \rightarrow [\alpha]\phi \wedge [\beta]\phi$.

Let ν be an arbitrary state. Assume $\nu \models [\alpha \cup \beta]\phi$.

$\omega \models [\alpha \cup \beta]\phi$	iff
for all ν such that $(\omega, \nu) \in [\![\alpha \cup \beta]\!]$, then $\nu \models \phi$	iff
for all ν such that $(\omega, \nu) \in [\![\alpha]\!] \cup [\![\beta]\!]$, then $\nu \models \phi$	iff
for all ν such that $(\omega, \nu) \in [\![\alpha]\!]$ or $(\omega, \nu) \in [\![\beta]\!]$, then $\nu \models \phi$	iff
for all ν such that $(\omega, \nu) \in [\![\alpha]\!]$ then $\nu \models \phi$ and for all ν such that $(\omega, \nu) \in [\![\beta]\!]$ then $\nu \models \phi$*	iff
$\omega \models [\alpha]\phi$ and $\nu \models [\beta]\phi$	iff
$\omega \models [\alpha]\phi \wedge [\beta]\phi$	

But 'lo and behold, all of the steps are in fact equivalences! So instead of proving a single direction of the equivalence, we actually proved both directions at once. This isn't always the case, so be careful!

And that's how you do a no-bullsh*t axiom proof![3]

*Note: We can do this because: A or B $\implies$ C iff (A $\implies$ C) and (B $\implies$ C)

[3]Fun fact: When I did the machine-checked proof of this axiom, it proved automatically. But you are not a computer, therefore we request detail in your proofs.

Additional example with composition axiom:

$$([;]): \quad [\alpha;\beta]\phi \leftrightarrow [\alpha][\beta]\phi$$

Let ω be a state. Assume $\omega \models [\alpha;\beta]\phi$.

$\omega \models [\alpha;\beta]\phi$ iff
for all ν s.t. $(\omega,\nu) \in [\![\alpha;\beta]\!]$, $\nu \models \phi$ iff
for all ν_0 s.t. $(\omega,\nu_0) \in [\![\alpha]\!]$, then for all ν s.t. $(\nu_0,\nu) \in [\![\beta]\!]$, $\nu \models \phi$ iff
for all ν_0 s.t. $(\omega,\nu_0) \in [\![\alpha]\!]$, then $\nu_0 \models [\beta]\phi$ iff
$\omega \models [\alpha][\beta]\phi$

Since these are all equivalences, then both directions of the axiom hold.

3. **Quiz**

   Suppose you have a proof for the following $\mathsf{d}\mathcal{L}$ formula:

$$\begin{aligned}&[t := 0;\\ &\{x' = v, v' = a, t' = 1 \ \&\ t \leq T\};\\ &?(t = T);\\ &\{x' = v, v' = a, t' = 1\}]\ \phi(x,v)\end{aligned}$$

   For what values of t do you know that your hybrid program ensures the property $\phi(x,v)$?

4. **Overly Conservative Controllers** Another way we can fail to prove what we meant to prove is when we design a controller that is far too *conservative*. For example, if we have a car that never ever accelerates, it will certainly never crash, but it would also be totall useless.

   To prevent this, we can also prove an *efficiency* theorem that shows our controller isn't too conservative. For example, an efficiency theorem for a car might tell us that it eventually reaches its destination, or that it only brakes when necessary.

   Let's explore efficiency further by looking at a highly relevant example. Say you're operating the water authority for a city of 300,000 people in western Pennsylvania. You have two mechanisms to prevent water contamination: A water filter and chlorination. If either of these mechanisms is working, your 100,000 customers in the central triangle of Pittsburgh won't get sick. If you take away their water (ok, actually, just tell them to boil it), they also won't get sick. But boiling water is an inconvenience, so an efficient controller would only issue a boil water order if both the chlorine level is too low and the filter is broken (filter = 0)

   Here is an example dL formula which, if proved, would give us both the safety theorem for the water authority (yinzers never get sick off water) and the efficiency theorem (we only issue a boil water order if someone would have got sick).

   Unfortunately, the efficiency theorem is false because this controller, as used by the PWSA, is inefficient. Can you find the bug?

```
   haveWater = 300000
& (filter = 1 | filter = 0)
& (chlorine >= 0 & chlorine <= cmax)
& cmax > 0
& sick = 0
->
```

```
[ /* Control logic */
  {{?(chlorine < cmax); haveWater := haveWater - 100000;}
  ++ {?(chlorine >= cmax);}}
  /* Physical dynamics */
  rate1 := 1-filter;
  rate2 := 1-(chlorine/cmax);
  sickRate := rate1 * rate2;
  {?(haveWater = 300000); sick' = sickRate;
  ++ ?(haveWater != 300000);}
] (

/* Safety: No sick yinzers */
   sick = 0
/* Efficiency: We only remove service when someone would have got sick */
&  (haveWater < 300000 -> sickRate > 0)
)
```

Now we can't blame the PWSA too much for being inefficient. At least they were safe. In the later labs (Lab 3 and 4 especially) you may run into a similar scenario where you have to pick between being totally safe and totally efficient. You should always prioritize being totally safe, and only then try to be as efficient as possible. On labs we will expect your models to satisfy basic efficiency properties (e.g. on Assignment 1 the spaceship should eventually reach the station), but it's often acceptable if more advanced efficiency properties don't hold (it's not that bad if you brake more often than strictly necessary).

Exercise: Fix the bug and prove the theorem.

In recitation we talked a bit about how one would do the proof. Recall from lecture that it's often useful to prove intermediate formulas using the HM rule:

$$\text{HM}\ \frac{[\alpha]B \qquad B \rightarrow [\beta]\phi}{[\alpha;\beta]\phi}$$

What the HM rules says is if we want to prove $[\alpha;\beta]\phi$, we can first dream up an intermediate formula B that's always true after B. We then actually *prove* that B is true after α and use it as an assumption to prove $[\beta]\phi$. Why is this useful? After all, it seems more complicated than the axiom $[;]$.

What it's useful for is allowing us to simplify the argument for $[\beta]\phi$, usually by reducing the amount of case analysis. Like most models, the model for PWSA can be divided into (ctrl; physics): that is, some discrete control logic followed by continuous physical dynamics. Any time that we have a complex controller with lots of different cases, we might not want to redo the entire proof for every case. Often there's some nice simple concise formula that should always hold at the end of controller, and which is sufficient for proving the physics correct. In this case, if $sickRate = 0$ holds at the end of the controller, we can use this to prove that both cases of the physics are correct.

Since proving $[\alpha]sickRate = 0$ only took 3 branches and proving $sickRate = 0 \rightarrow [\beta]safe$ only took 2 branches, we've only proved 5 cases total instead of the 6 we would have to prove if we looked at every case separately. This time we only saved 1 case, but the savings can be bigger for more complicated models like you might see in your course projects. Moreover, $sickRate = 0$ is really a nice way to summarize the core safety argument for your proof. So even when you don't totally need this rule, asking yourself "what should be true at the end of the controller?" can be a really good way to organize the proof!

5. **Sequent Calculus and Real Arithmetic:**

Yesterday in lecture we talked about doing proofs in sequent calculus, including proofs about real arithmetic. What is sequent calculus? Sequent calculus is just a more convenient way to prove that formulas are valid: as we go through the proof, it lets keep a list Γ of all our assumptions (which we can then use in the proof). It also turns out to be convenient that sequent calculus lets us keep a list Δ of conclusions and prove whichever of those conclusions we want.

More formally, sequent calculus proves the validity of sequents instead of formulas. We say a sequent $(\Gamma \vdash \Delta)$ (pronounced "Gamma entails Delta") is valid iff for all states $\omega, \omega \models (\Gamma \vdash \Delta$, where $\omega \models (\Gamma \vdash \Delta)$ means $(\forall \phi \in \Gamma\ \omega \models \phi) \implies (\exists \psi \in \Delta\ \omega \models \psi)$. We could also think of sequents like formulas. We could define a formula $\text{fml}(\Gamma \vdash \Delta) \equiv (\bigwedge(\phi \in \Gamma) \to \bigvee(\psi \in \Delta))$ and then proving the validity of the sequent $\Gamma \vdash \Delta$ is just the same as proving the validity of the formula $\text{fml}(\Gamma \vdash \Delta)$.

Moreover, why did we suddenly switch from proving formulas to proving sequents? Well at the end of the day we're still really proving things about individual formulas. When we want to prove a formula ϕ, this is exactly the same as proving the sequent $\vdash \phi$ (there's no assumptions and one conclusion). At the end of the day we're not proving something fundamentally different, sequents just give us a more convenient way to do the proof.

The $\to R$ rule lets us add assumptions to the list of assumptions:

$$\to R\ \frac{\Gamma, P \vdash Q, \Delta}{\Gamma \vdash P \to Q, \Delta}$$

The hypothesis rule (or *id* rule) allows us to finish a proof when we already have the conclusion as an assumption:

$$\text{id}\ \frac{*}{\Gamma, P \vdash P, \Delta}$$

(the * on top indicates that this part of the proof is done)

We also have sequent rules that let us talk about quantifiers:

$$\to R\ \frac{\Gamma \vdash p(x), \Delta}{\Gamma \vdash \forall x\ p(x), \Delta}\ x \notin \Gamma, \Delta$$

Why is this rule sound? Why can we just get rid of the quantifier? This is because when we're doing a proof, we're always proving *validity*: the formulas are true in all states. This means that when a rule assumes $p(x)$, it's also assume $p(x)$ is true in all states.

Recall that when we're proving validity, it's like assuming all variables have universal quantifiers: proving $\forall x\ x^2 + y^2 = 0$ valid is therefore just like proving $\forall y \forall x\ x^2 + y^2 = 0$ valid: we automatically get a universal quantifier. In contrast, proving $\forall x\ x^2 + y^2 = 0$ *satisfiable* would be like proving $\exists y \forall x\ x^2 + y^2 = 0$ satisfiable because satisfiability turns free variables into existentials. But when we do a sequent calculus proof or a KeYmaera X proof we're always doing validity.

Ok, but why do we need the side condition $x \notin \Gamma, \Delta$? This is super subtle. The short answer is that validity always adds a quantifier at the outside of the *entire sequent*, regardless of whether some of the formulas in the sequent have quantifiers (note, there is no way to write down a quantifier that works "across" multiple formulas in the same sequent).

Say I want to prove the sequent $\vdash (\forall x\ x > 0), x \leq 0$. This sequent is actually not valid. For example it is false in the state $x = 1$ because the formula $(\forall x\ x > 0)$ is *never* true ever. (Because it has a quantifier it ignores the state and uses its own x). That means proving this formula should be as hard as proving $x \leq 0$ valid, and that's also not valid because it's false when $x = 1$. Why do we care? Because without the side condition we would have been allowed to do this:

$$\forall\mathrm{R}\,\frac{\mathbb{R}\,\dfrac{*}{\vdash (x > 0, x \leq 0}}{\vdash (\forall x\ x > 0), x \leq 0}$$

Removing the quantifier would actually turn this into a valid sequent because the sequent $\vdash x > 0, x \leq 0$ or equivalently formula $x > 0 \lor x \leq 0$ *is* valid: all x are either positive or not.

Another way to understand this is that we're always allowed to use $\forall R$ but sometimes we have to rename the variable to a new variable first.

$$\begin{array}{r} \forall\mathrm{R}\,\dfrac{\vdash y > 0, x \leq 0}{\vdash (\forall y\ y > 0), x \leq 0} \\ \text{rename}\,\dfrac{}{\vdash (\forall x\ x > 0), x \leq 0} \end{array}$$

In other words, before we can do $\forall R$ we have to make it obvious that the two x's we saw in the previous formula really honestly ought to be different variables even though the have the same name (the forall quantifier introduced a new variable that was unfortunately also named x). Once we rename the variable, we end up with a sequent $\vdash y > 0, x \leq 0$ which is not valid because $y > 0 \lor x \leq 0$ isn't valid either, e.g. when $y = 0, x = 1$.

Also, not mentioned in recitation: Technically, any time we have a first-order logic formula (just quantifiers and arithmetic) there's an algorithm that can tell us whether it's valid. However, that algorithm is remarkable slow on large inputs, so if pushing the play button doesn't give you an answer quickly, you need to do some work yourself to finish the proof. Instantiating quantifiers is one really good way to make the proof close faster. Note that $\forall R$ is not instantiating a quantier, instantiating a quantifier would be $\forall L$:

$$\forall\mathrm{L}\,\frac{\Gamma, p(\theta) \vdash \Delta}{\Gamma, \forall x\ p(x) \vdash \Delta}$$

When you apply this rule, you plug in some term θ and instead of assuming $p(x)$ for all x you now just assume it for $x = \theta$ and forget all the other cases. Removing the quantifier like this can make the proof finish much faster, but you're also losing information because you forgot $p(x)$ for all the other x's. So make sure you pick the right θ!