15-424/15-624 Recitation 4
Eventful Tactical KeYmaera X Proofs

1 Assignment 1 + Lab 1

In recitation we discussed common mistakes on the assignments, but we don't want to give out solutions in the recitation notes. If you have questions, come talk to us.

2 I played ping-pong so much, I even played it in my sleep.

The topic of today's recitation is how to do more complicated proofs in KeYmaera X, especially by writing your proofs as programs called *tactics*. For our example we will use event-based controllers as discussed in yesterday's lecture, and review common modeling mistakes as they appear in event-based controllers.

Our main example will be a ping-pong match between two players: Forrest and Dan. Forrest and Dan are both very good at ping-pong, so our safety theorem will say that the ping-pong ball never gets past either of them. Forrest is standing on the left at position l and Dan is standing on the right at position r and the position of the ping-pong ball will be named x so we could start our model by saying:

$$\texttt{Preconds} \rightarrow [\{\texttt{???}; \{x' = v \& \texttt{???}\}\}^*] l \leq x \leq r$$

At this point we know the safety condition ($l \leq x \leq r$) and at least something about the dynamics ($x' = v$ where x is the ball's position and v is the ball's velocity). Next let's figure out the system's controller, domain constraint, and preconditions.

Preconditions: Well, we need to know the system is safe initially, so we definitely want $l \leq x \leq r \land l < r$ as a precondition. Do we need any preconditions on v? The ball is going to travel both ways eventually, so maybe we won't need a precondition. But for now let's make life simpler by assuming Forrest hit the ball most recently, so $v \geq 0$.

3 How Not To Model An Event-Driven System

What about the evolution domain constraint? Our first guess might be something like $\leq x \land x \leq r$. However, this is wrong, as per last week's recitation:

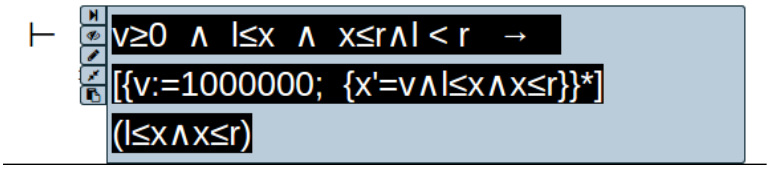
Major Red Flag: If we can prove safety while totally ignoring the controller, our model is broken.

Let's add a totally bogus controller to the model ($v := 1000000$) and prove it, just to drive home how this model is bad. And since this is a KeYmaera X recitation, let's do the proof in KeYmaera X.

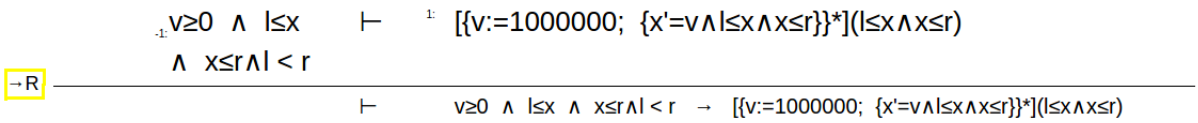
$v \geq 0 \land l \leq x \leq r \land l < r \rightarrow [\{\ v := 1000000;\ \{x' = v \& l \leq x \land x \leq r\}\ \}*](l \leq x \leq r)$

3.1 Manual Proof

Let's prove this VERY BAD theorem by interacting with KeYmaera X manually. The simplest way to do manual proofs is by mousing over the formula and clicking. Note that different subformulas will highlight based on where you mouse. Clicking will perform the "most obvious" next step for the highlighted formula, where "most obvious" means the same thing it does in `master`.



KeYmaera X knows what to do for implications, and it gives us back a new goal to prove!



Like you would do if you proved this by hand, KeYmaera X applies the →R proof step to the formula in the succedent. You can tell this is what it tactic did by observing the label next to the horizonal line in the proof.

3.2 Automated Proof (and Inspection!)

Notice that now the antecedent has the conjunction $v \geq 0 \land l \leq x \land x \leq r \land l < r$ in it. We could break this apart by clicking on it repeatedly, but this is easy enough that KeYmaera X's automation should be able to handle it. Let's click the "Unfold" button which expands away

all the "easy" logical connectives/programs. Now we have four assumptions, one for each conjunct!

You might wonder: What did that automatic tactic actually do? KeYmaera X has a new feature that lets you explore the steps taken by automation. After running `unfold`, right-click on the word `unfold` that appears at the left of the latest proof step and click "Details". This will bring up a screen that shows you all the tactics that ran as part of `unfold`. You will see that `unfold` ran `chase` (which gets rid of easy programs) and `normalize` (which gets rid of easy logical connectives). But we don't know what `chase` and `normalize` are! Luckily you can ask for *more* details. Ask for the details on `normalize` and it will show you that it ran `andL` three times to break the formula apart.

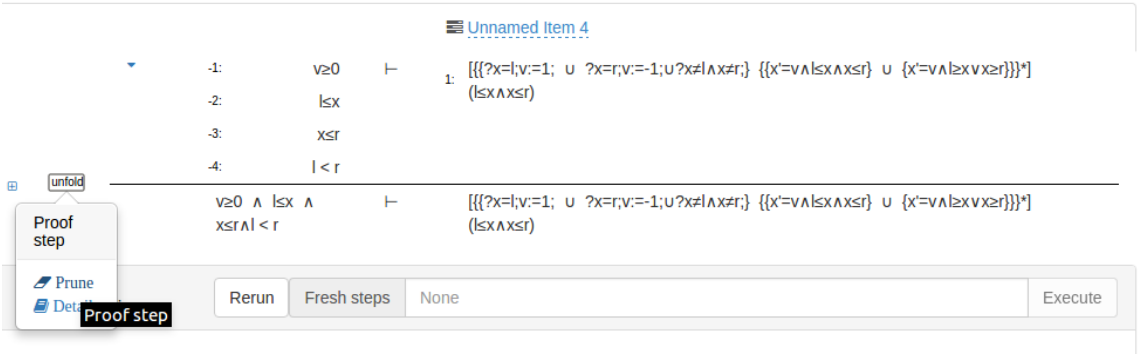
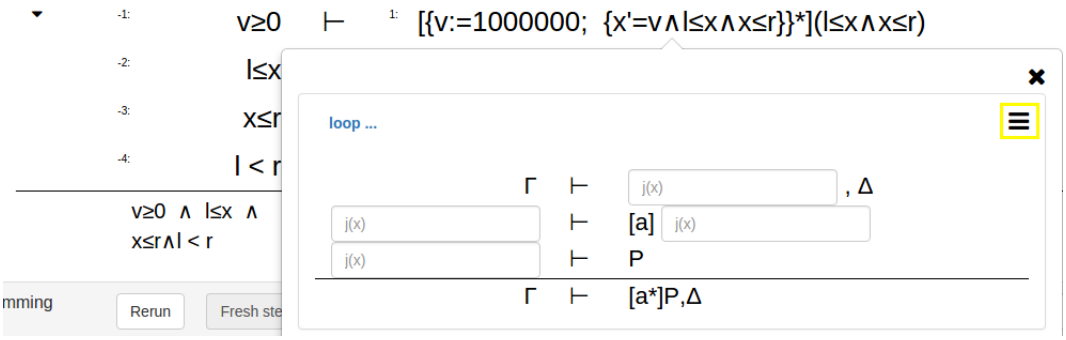


Figure 1: Details button

3.3 Invariants

The top-level operator on our new formula is a $[\alpha^*]$, so let's apply the `loop` rule. Unlike the rules we used so far, this one requires some input from us: we need to tell it what loop invariant we want to use. To type in a loop invariant, right-click on the formula and then click on the ... next to `loop` on the window that pops up.

KeYmaera X Protip: Proof rules that need input are in the right-click menu. Note the button on the top-right, shown in yellow. This button will show you the definition of any rule so you can know what it does before you use it.



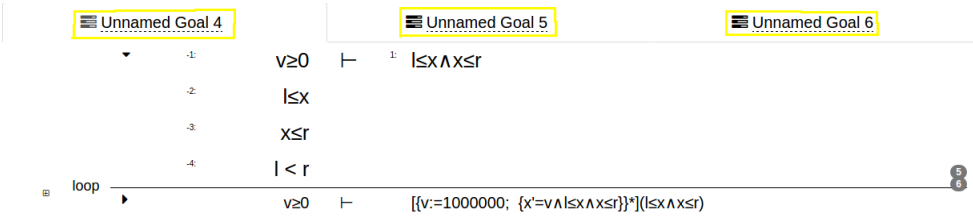
What's a good loop invariant, anyway? Remember that our loop invariant must be:

- *At least as strong as* the postcondition
- *No stronger than* the preconditions
- *Preserved* every time you run the loop body

Thus a good way to come up with a invariant is to look for weaker versions of your preconditions or stronger versions of your postconditions, then make sure the invariant really doesnt vary (is preserved by the loop body). In this case the postcondition appears in the precondition, so the postcondition $l \leq x \leq r$ is a good guess for loop invariant. In this case it will, indeed, be the correct invariant.

Important Proving Strategy: If you pick the wrong invariant, your proof might not work even when the theorem is true. If you get stuck and you think the theorem is true, try picking a new invariant and trying again!

Once we put in the loop invariant and run `loop` by clicking on its name, we get back 3 subgoals. In the sequent proof above this was represented by two new goals on top of a single old goal. To keep things readable, KeYmaera X places each subgoal in a separate tab:

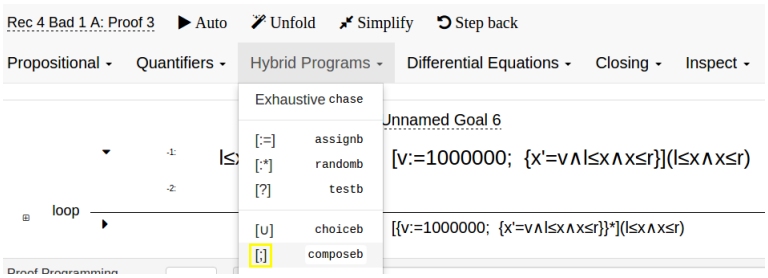


And now we can continue on with the proof in each branch. The first two branches look really trivial, so we prove them with `master` and they close immediately.

The third subgoal (showing that the loop invariant really is an invariant) looks a lot more complicated, so let's prove that manually. This is another common pattern in proofs:

Important Proving Strategy: Usually the hard part of proving a loop invariant is showing that the body preserves the invariant. Always try showing "precondition implies invariant" and "invariant implies postcondition" first with `master` because if it fails, you just found a mistake in your invariant for free!

So let's prove the body preserves the invariant. On the outside we've got a semicolon so we should be able to apply the [;] rule. We could just click on the formula, but let's use a different KeYmaera X feature instead. If we click on the Hybrid Programs menu at the top of the screen, we'll get a list of proof steps for hybid programs. When we click on the [;] entry in the menu, it runs the [;] rule for us. This menu (and the other menus) are useful when you're not sure what different proof rules are available, because you can browse through all the proof rules.



Let's recall that the point of doing this proof was to demonstrate how we didn't even need the control logic at all to prove the theorem, and in fact we don't even need anything except the domain constraint. Next let's use a proof rule that says "throw away the control and keep just the physics". Since this rule mostly comes up in bad models, naturally KeYmaera X makes us go out of our way to use it. To find it, open up the right-click menu.

KeYmaera X Protip: Advanced rules are in the right-click menu. The menu adjusts itself based on where you click so it only shows you rules that you can use right now.

The rule we want is called `GV`. This is like the G rule from last week:

$$\text{G}\,\frac{\vdash \phi}{\vdash [\alpha]\phi}$$

except it also keeps around assumptions when possible, if it can easily tell the program α won't make the preconditions stop being true.

This gives us a $\forall$ that we get rid of by clicking, and now we just have the ODE. We could solve the ODE by clicking on it, but we really want to drive home that the physics itself

doesn't matter, just the domain constraint. We actually have a rule that says just that! We'll discuss it in much more detail in lecture next week when we talk about more complicated ODE proofs, but for now just know there's a rule called *differential weakening* or `dW` that throws out an ODE and keeps just the domain constraint:

$$[x' = \theta \& \phi]\psi \leftarrow (\phi \rightarrow \psi)$$

Tangent: Notice this looks a *lot* like the axiom for tests $?(\phi)$! This is why ODE's have to blow up if the constraint is initially false, just like tests blow up.

Now, `dW` is used even when proving good models, but it's also an advanced rule, so we find it in the right-click menu too.

3.4 Tactics

The nice thing about doing proofs on the UI is that it's easy to get started and easy to explore. The bad thing is that it gets really repetitive for bigger models. Even worse, if we make a small change to our model, we don't want to have to redo the entire proof from scratch: We want keep most of our proof the same and just change the parts that matter.

To fix these problems, we can write our proofs as programs called *tactics* instead. Without tactics, proofs would get too slow. Or in the words of the great mathematician Sun Tzu:

> Strategy without tactics is the slowest route to victory.

So to start learning tactics, let's look at a tactic that does the same proof we just did by hand:

```
implyR(1) ; loop({`l<=x&x<=r`}, 1) ; <(
  master,
  master,
  composeb(1) ; GV(1) ; allR(1) ; dW(1) ; master
  )
```

As you can see, the syntax for tactics is much more intimidating for newcomers than the UI. For this reason, I encourage you to start out using the UI first when starting a proof, or even working on paper, and then use tactics to make repetitive tasks easier once you understand the structure of the proof. If you only use tactics, you will likely get distracted by the syntax. Or to complete the Sun Tzu quote:

> Tactics without strategy is the noise before defeat.[1]

[1] He also said "All warfare is based on deception," but please do not try to deceive course staff with your homework submissions

What does the syntax mean anyway? The `;` symbols mean "do the left and then the right (but only if the left succeeded". The < symbol is used whenever the proof branches from one goal to two (or more) goals. Some tactics have arguments, which go in parentheses, others don't. If a tactic operates on a whole subgoal (like `master` or `QE`) then we don't need to give it any numeric arguments. If it operates on one specific formula, then we have to give it a *position* argument saying which formula. Thankfully, the Web UI already shows you the number for each formula, and most of the time you will want formula number 1 (the first formula in the succedent). Some tactics like `loop` take more complicated arguments like formulas: those arguments go inside funny brackets with backticks that look like `{‘ARG‘}`.

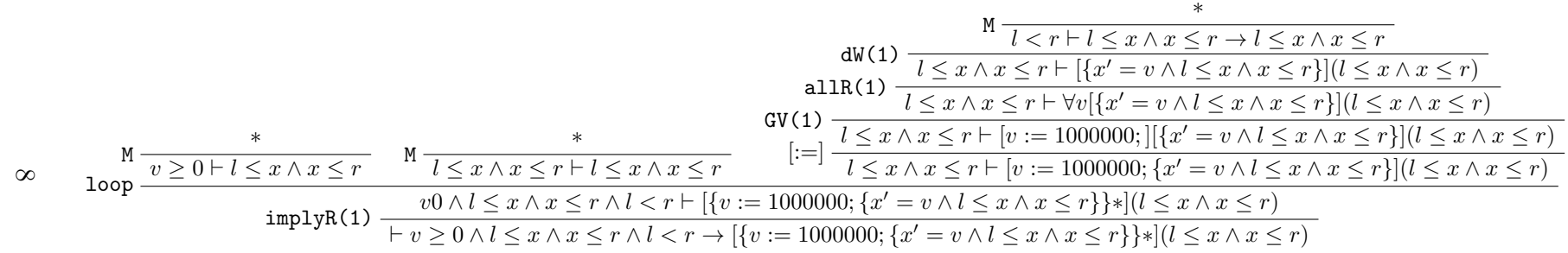
Exercise: I claimed that this proof "ignores the control and ODE". What this means is that the same exact proof, and thus the same exact tactic, will work if we change the controller and ODE! Write a new model with a different controller and different ODE (but same domain constraint) and rerun the proof by copy-pasting the tactic into the Proof Programming dialog!

Now, when we have more interesting models that aren't broken, we might have to make some changes to the tactic, but we can reuse most of it.

For more details on tactics combinators (;, <, etc.) see the KeYmaera X cheat sheet http://www.ls.cs.cmu.edu/KeYmaeraX/KeYmaeraX-sheet.pdf.

Another nice thing about tactics is that they give us a really concise way to write down proofs, which makes them easier to read when you (or your grader) are trying to understand them. Compare the tactic with the proof tree written in Figure 3.4: The proof tree gives us more details, but the tactic is much more concise. Essentially, the tactic lets us stop writing down all the formulas, and instead just write down the rule names and the shape of the proof tree.

Let's save this proof as `broken1.kyt` for later. If you want to execute this tactic, just open the tactic editor in the KeYmaera X proof UI, copy/paste, and click execute:



Where `loop` is short for `loop({‘l<=x&x<=r‘}, 1)`, `M` is short for `master`, and [:=] is short for `composeb(1)`

Figure 2: A proof tree that was very difficult to fit on one page

4 Tactics From Manual Proofs

Note that I didn't have to type any tactics myself for the last proof: KeYmaera X did all the work for me. Whenever you perform a proof step, either by a tactic or manually, KeYmaera X will add that step to the tactic shown in the editor. That is, the Proof Programming pane will always display a tactic that represents your entire proof so far.

We encourage you to pay attention to the Tactic Editor when you do manual proofs, as this will help you passively learn the tactic language, even when you're not writing tactics! Any time you want to write a tactic, but don't know the name, you can do the same proof step on the UI and read the tactic name from the Tactic Editor. Furthermore, if you have a proof with several repetitive cases, you can do the first case manually to make sure it succeeds, then copy-paste and modify the auto-generated tactic to finish the remaining cases.

Note that auto-generated tactics look a little different from what I wrote in the notes: there's an additional `nil` tactic. As the name suggests, the `nil` tactic does nothing, so you can ignore it. It's just a byproduct of the tactic generation process.

4.1 More Advanced Bad Models

The fundamental problem with the previous model is that *domain constraints are assumptions*. If all we have is an ODE with the domain constraint $l \leq x \land x \leq r$ this is saying that at all times, we're assuming $l \leq x \land x \leq r$. It's like there's a physical constraint that x can never leave the playing area, or it's like the playing area is the only thing that exists in the whole universe. That's not good! In the real universe, there are places other than the ping pong table, but we want to *prove* that the ball doesn't leave the table area.

In a more advanced, but equally bad model, we can add an extra ODE whose domain constraint is the *negation* of the first ODE:

$v \geq 0 \land l \leq x \leq r \land l < r \rightarrow [\{ v := 1000000; \{x' = v \& l \leq x \land x \leq r\} \cup \{x' = v \& l < x \lor x > r\} \}*](l \leq x \leq r)$

If we try to prove this model, we will succeed. Here's a tactic for the proof. Notice the proof is very similar. The main difference is that we need two cases because there's a choice in the problem.

> Mama always said nondeterministic choice is like a box of chocolates, you never know what you're gonna get.

The second case actually won't close by `dW`, but it will close by `solve` or `master`.

```
  implyR(1) ; loop({`l<=x&x<=r`}, 1) ; <(
    master,
    master,
    composeb(1); GV(1) ; allR(1) ; choiceb(1) ; andR(1) ; <(
      dW(1) ; master,
      solve(1)
  )
)
```

Our controller is still bogus, so why did this proof succeed? The answer is that once we get on the ping pong table, the model keeps us stuck on the ping pong table!

More formally, we should think of an event-driven model as dividing the universe into several distinct *modes* or *zones*. The model describes where the modes are and when we can switch between modes, but it should always be *physically possible* to switch between zones: A mode that we can't get to might as well not exist. Once we've divided the world into modes, the safety proof consists of proving that *each mode* is safe. The safety argument for each mode will be based on the physics and on the invariants.

In this model, there are two modes. Let's call "the ping-pong table" Mode A, and "everywhere else" Mode B. As shown in Figure 4.1, Modes A and B do not intersect at all (represented as a dotted line for their boundary). If we start in Mode A, the domain constraint says we stay in Mode A as long as the ODE is running. Because Mode A doesn't overlap Mode B, we will never start being in Mode B and so we never have to argue why Mode B is safe.

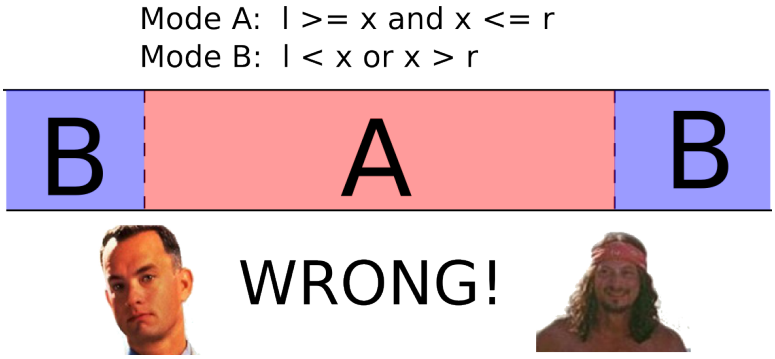


Figure 3: Incorrect: Can't switch between modes

4.2 Another, Simpler Bad Model

In fact, there's an even simpler model with the same problem. The border between the US and Canada is at the 49th parallel. Imagine you're starting out at the 35th parallel and keep

moving north. Eventually you'll get to Canada. But according to a broken event-triggered model, you would never get to Canada! For example, this theorem is true:

```
lat = 35 ->
[vel := 1000000;
{   {lat' = vel & lat <= 49}
 ++ {lat' = vel & lat > 49}}
] lat <= 49
```

Why? The position `lat = 49` functions like a barrier: You can keep traveling in the USA up to and including the 49th parallel, but you aren't allowed into Canada until you're already PAST the 49th parallel. This means no matter how long you keep trying, you're always still in the USA. In short, *don't put border walls in your event-triggered models.* Not only will your model be very broken and receive little credit, but the ACLU will probably sue you.

5 A Correct Model

What's the solution? We have to make Modes A and B overlap just a little bit, by turning the dotted line into a solid line and turning $x < l \vee x > r$ into $x \leq l \vee x \geq r$, as in Figure 5

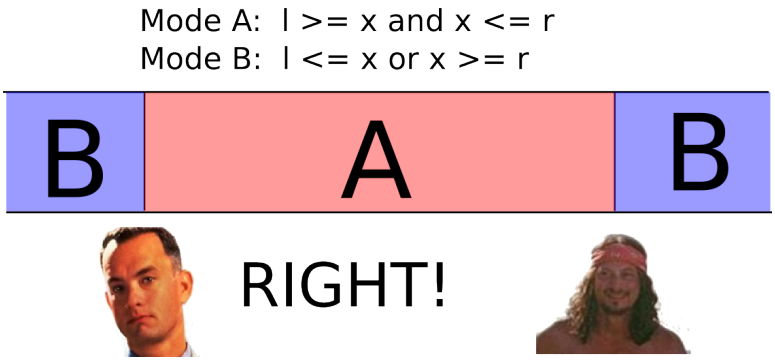


Figure 4: Correct: Can switch between modes

Note the first iteration of the control loop is the same in both cases: the ball can only move within Mode A. However, the second iteration is extremely different: The broken model forces the ball to stay in Mode A forever, but in the new model we can go anywhere in Mode B. That is, of course, as long as we ended Mode A with $x = l$ or $x = r$ exactly. The region $x = l \vee x = r$ has a special importance: This is the only place we can switch between modes A and B. We call this area the *Event Trigger*, because any time we enter the event trigger, we know we will get a chance to make another control decision before any more physics happen. Thus an event-based system is safe if we always make a safe control decision when we reach the event trigger.

Speaking of safety, if we try to prove the new model, it will finally fail. If you look closely, you'll notice that Mode A is safe, and the proof fails in Mode B. This makes sense: Mode A was defined to be exactly the safe region. In this case, a safe controller for Mode B needs to make sure we never leave Mode A (i.e. we immediately bounce the ball back toward our opponent). This model has a totally broken controller, so when we enter Mode B we keep moving to the right, leaving Mode A immediately.

We can fix the model by adding controls that do the right thing if we're exactly at the event trigger. Specifically, at the left boundary we should hit the ball to the right and vice versa:

```
v >= 0 & l <= x & x <= r & l < r ->
[{
    {   {?(x = l); v := 1;}
    ++ {?(x = r); v := -1;}
    ++ ?(x != l & x != r);}
 {   {x'=v & l <= x & x <= r}
 ++ {x'=v & l >= x | x >= r}}
}*](l <= x & x <= r)
```

And indeed, this model proves successfully.

6 Manual proofs

6.1 The ODE solve rule

The ODE solve rule, which deconstructs the program $[x' = f(x)\&H]\phi$ is fairly complicated. Luckily, it's usually used in a really straightforward way. In fact, you often need to make a single choice, and that choice is usually clear. Let's start:

$$\text{ODE solve}\ \dfrac{\forall_R\ \dfrac{\forall_R\ \dfrac{\rightarrow_R\ \dfrac{\rightarrow_R\ \dfrac{\forall_L\ \dfrac{?}{t_0 \geq 0, \forall s.\,(0 \leq s \wedge s \leq t_0 \rightarrow \langle x := \varphi(s)\rangle H) \vdash \langle x := \varphi(t_0)\rangle \phi}}{t_0 \geq 0 \vdash \forall s.\,(0 \leq s \wedge s \leq t_0 \rightarrow \langle x := \varphi(s)\rangle H) \rightarrow \langle x := \varphi(t_0)\rangle \phi}}{\vdash t_0 \geq 0 \rightarrow \forall s.\,(0 \leq s \wedge s \leq t_0 \rightarrow \langle x := \varphi(s)\rangle H) \rightarrow \langle x := \varphi(t_0)\rangle \phi}}{\vdash \forall t\,(t \geq 0 \rightarrow \forall s.\,(0 \leq s \wedge s \leq t \rightarrow \langle x := \varphi(s)\rangle H) \rightarrow \langle x := \varphi(t)\rangle \phi)}}{\vdash [x' = f(x)\&H]\phi}$$

But when we assume that something is true for all s, then we can choose what we want s to be. A smart choice will help us prove our property. A bad one will not.

In order to get some intuition, think back on lab 1, where your robot had to stop before the charging station. If you chose your acceleration just right, the robot would stop right at the charging station.

Because of non-determinism, we allowed the continuous evolution to stop at any point (as long as the velocity was non-negative). But really, if the robot was moving forwards, what would be the duration that would make the robot most likely to go past the station?

The longest duration possible! In the proof, this is represented by the free variable t_0. In fact, it's at time t_0 that ϕ needs to hold. So let's try to substitute that in.

$$\forall_L, s := t_0 \dfrac{\rightarrow_L \dfrac{\text{ax}\dfrac{*}{t_0 \geq 0 \vdash 0 \leq t_0 \land t_0 \leq t_0} \quad \dfrac{\text{rest of proof}}{t_0 \geq 0, \langle x := \varphi(t_0)\rangle H \vdash \langle x := \varphi(t_0)\rangle \phi}}{t_0 \geq 0, 0 \leq t_0 \land t_0 \leq t_0 \rightarrow \langle x := \varphi(t_0)\rangle H \vdash \langle x := \varphi(t_0)\rangle \phi}}{t_0 \geq 0, \forall s.\, (0 \leq s \land s \leq t_0 \rightarrow \langle x := \varphi(s)\rangle H) \vdash \langle x := \varphi(t_0)\rangle \phi}$$

Given that choice, the rest becomes pretty easy! One of the branches proves trivially. The other, $t_0 \geq 0, \langle x := \varphi(t_0)\rangle H \vdash \langle x := \varphi(t_0)\rangle \phi$, essentially means $H \rightarrow \phi$, since the assumption and the desired conclusion are talking about the same state!

6.2 Cut - bring your knife! haha so fun

It is a truth universally acknowledged that computers are in want of intelligence! Sometimes, they need guidance when we tell them to find proofs for a given formula. They get easily confused by formulas that say what we mean in strange ways.

To illustrate, let's start working with the following:

$$\big((x-y)^2 \leq 0 \land \phi(x)\big) \rightarrow \phi(y)$$

The proof starts easily enough...

$$\rightarrow_R \dfrac{\land_L \dfrac{\dfrac{?}{(x-y)^2 \leq 0, \phi(x) \vdash \phi(y)}}{(x-y)^2 \leq 0 \land \phi(x) \vdash \phi(y)}}{\vdash ((x-y)^2 \leq 0 \land \phi(x)) \rightarrow \phi(y)}$$

But now what? well, if ϕ is a FOL formula and it's small and simple enough, it might be possible to get KeYmaera X and its QE procedure to do the work for us.

Unfortunately, that's often not the case. Formulas are long, complicated, and involve lots of variables, and so KeYmaera X is going to really struggle.

What can we do? Looking at the assumption $(x-y)^2 \leq 0$, we can draw some conclusions. We know that $(x-y)^2$ as to be non-negative, which with our assumption gives us $(x-y)^2 = 0$. in turn, this tells us that $x = y$. But wait! If they are equal, then the formulas $\phi(x)$ and $\phi(y)$ are equivalent, making them trivial to prove!

That is a lot easier for KeYmaera X to realise, so we are going to cut in the fact that $x = y$. Furthermore, we will also use the weakening or hiding rules to make sure KeYmaera X focuses on the right subproblems of our proof, instead of worrying with extraneous stuff.

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{(x-y)^2 \leq 0 \vdash x = y}\,\text{QE}}{(x-y)^2 \leq 0 \vdash x = y, \phi(y)}\,W_R}{(x-y)^2 \leq 0, \phi(x) \vdash x = y, \phi(y)}\,W_L \qquad \dfrac{\dfrac{*}{x = y, \phi(x) \vdash \phi(y)}}{(x-y)^2 \leq 0, x = y, \phi(x) \vdash \phi(y)}\,W_L}{\dfrac{(x-y)^2 \leq 0, \phi(x) \vdash \phi(y)}{(x-y)^2 \leq 0 \wedge \phi(x) \vdash \phi(y)}\,\wedge_L}\,\text{cut}$$
$$\dfrac{(x-y)^2 \leq 0 \wedge \phi(x) \vdash \phi(y)}{\vdash ((x-y)^2 \leq 0 \wedge \phi(x)) \to \phi(y)}\,\to_R$$

This practice of weakening/hiding and then applying QE is extremely helpful to save time in your proof attempts!

6.3 Quick Help

We covered a lot of KeYmaera X features in this recitation. You can always come back to the notes if you forget, but it would also be nice if KeYmaera X can remind you about its features. In fact it can! If you go to the omnipresent "Help" menun and click "Quick Usage Help", it will show you what different parts of the UI do. This help screen is a little basic right now, but we are hoping to make it more and more useful as the semester goes on.

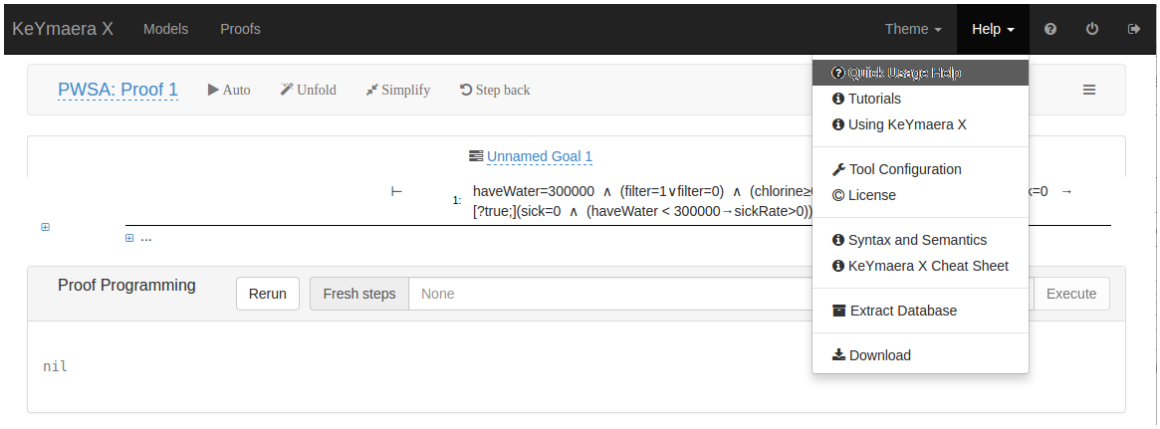


Figure 5: Quick Help