Recitation 4: Eventful Tactical KeYmaera X Proofs
15-424/15-624/15-824 Logical Foundations of Cyber-Physical Systems

1 Announcements

- Theory 2 and Lab 2 have been released. Read the instructions for updating KeYmaera X and for submitting in pairs (if applicable) carefully!

2 Review: Theory 1 and Lab 1

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

3 Motivation and Learning Objectives

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*. This will be useful for Lab 2, because some of your models will be complicated enough that simply pressing the `Auto` button will not work anymore!

Note: This recitation was developed in KeYmaera X 4.5.0.

For our running example we will develop an event-based controller as discussed in Thursday's lecture, and review common modeling mistakes as they appear in event-based controllers.

The scenario we will be modeling is 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:

$$Pre \rightarrow [(Ctrl; \{x' = v \,\&\, Q\})^*] l \leq x \leq r$$

At this point we know the safety postcondition ($l \leq x \leq r$) and at least something about the dynamics ($x' = v$ where x is the ball's position and v is the ball's velocity). Next, we will need to add the controller ($Ctrl$), domain constraint (Q), and preconditions (Pre).

Exercise 1:
Suggest some preconditions that we will definitely need in the system.
Answer: Since the loop can always run for no iterations, we will definitely at least need the preconditions to imply the postcondition $l \leq x \leq r$. In addition, it will probably be nice if the players were not standing on the same spot, or more precisely, $l < r$. For simplicity, we shall just assume that the ball is in flight already, i.e., $l < x < r$, which also implies $l < r$.

Do we need any preconditions on v? The ball is going to travel both ways eventually, so maybe we will not need a precondition. But for now we will make life simpler by assuming Forrest hit the ball most recently, so $v \geq 0$.

4 How Not To Model An Event-Driven System

Let us start with the evolution domain constraint. Our first guess might be something like $l \leq x \leq r$. However, this is incorrect, as we saw in last week's recitation: it is a **major red flag** if you manage to prove safety while totally ignoring the controller. Your model is most likely broken.

As an example, let us just add a totally bogus controller to the model and prove that it is "safe". Here is the arbitrarily chosen controller we will use:

$$Ctrl_{bad} \stackrel{\text{def}}{\equiv} v := v + 1 \cup \{v' = 5\}$$

Since this is a KeYmaera X recitation, we will do the proof in KeYmaera X. We already saw a similar proof last week, but this time, we will do it manually and also explore KeYmaera X's tactics along the way. Here is the formula that we will prove valid:

$$v \geq 0 \land l < x < r \rightarrow [(Ctrl_{bad}; \{x' = v \,\&\, l \leq x \leq r\})^*] l \leq x \leq r$$

Note: The model for this is called rec4bad1 in the archive.
Note: Caveat – this controller was chosen so that KeYmaera X's heuristics will fail to complete the proof fully automatically. For simpler controllers, KeYmaera X might even be able to prove the corresponding formula fully automatically. Always be careful if your proof succeeds much faster than expected!

4.1 Initial Steps

The simplest way to do manual proofs is by mousing over the formula and clicking. Note that different sub-formulas will highlight based on where you mouse. Clicking will perform the "most obvious" next step for the highlighted formula, The "most obvious" step is usually what the `master` tactic tries when it is unfolding your formula.

In a sequent calculus, this is somewhat obvious for the propositional connectives: you simply use the appropriate rule for that connective. Similarly, for most of the hybrid program operators, there is really only one choice of axiom that can be applied.

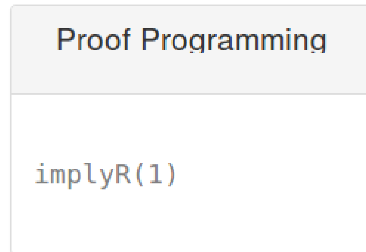
In our case, the topmost logical connective in the formula that we want to prove is an implication. KeYmaera X knows what to do for implications on the right (recall sequent calculus from last week), and it gives us back a new goal to prove:

Goal: →R (127,0)

Hint: LOOP | BOXAND

⊞ →R -1: v ≥ 0 ∧ l < x ∧ x < r ⊢ 1: [{ { v := v+1 ; ∪ { v' = 5 & true } } { x' = v & l ≤ x ∧ x ≤ r } }*] (l ≤ x ∧ x ≤ r)

Like you would do if you proved this by hand, KeYmaera X automatically used the $\rightarrow$R rule. You can tell this is what it tactic did by observing the label next to the horizontal line in the proof. The `Proof Programming` box shows that you applied the **`implyR`** tactic at position 1:



Notice that the antecedent has the conjunction $v \geq 0 \land l < x < r$ in it. We could break this apart by clicking on it repeatedly (which would apply the $\land$L rule), but this unfolding is easy enough that KeYmaera X's automation should be able to handle it.

By clicking the `Unfold` button, or simply typing in **`unfold`**, KeYmaera X will expand away all the "easy" logical connectives/programs. Now we have three assumptions, one for each conjunct!

But now KeYmaera X is stuck: the next proof step is not quite so obvious.

4.2 Loops Invariants

The top-level operator on our new formula is a loop $(\cdot)^*$. Recall the loop rule from lecture:

$$\text{(loop)} \quad \frac{\Gamma \vdash J, \Delta \qquad J \vdash [\alpha]J \qquad J \vdash P}{\Gamma \vdash [\alpha^*]P, \Delta}$$

The reason KeYmaera X got stuck unfolding the loop is because it does not know what formula to use as the loop invariant J.

Exercise 2:
Suggest a loop invariant.
Answer: The loop invariant in this case is fairly straightforward: we are just going to use the postcondition, i.e., $l \leq x \leq r$.
The requirements for J to be a loop invariant can be seen from the rule loop. It must:

- *Imply* the postcondition
- *Be implied by* the preconditions
- *Be preserved* every time you run the loop body

Note: If you pick the wrong invariant, your proof might not work even when the theorem you want to prove is true. If you get stuck and you think the theorem is true, try picking a new invariant and try again.

To hand KeYmaera X a loop invariant, right-click on the loop in the succedent and the menu of applicable rules will appear.[1] The loop induction rule should be at the top.

Loop Induction loop

$$\frac{\Gamma \vdash j(x), \Delta \qquad j(x) \vdash [a]\,j(x) \qquad j(x) \vdash P}{\Gamma \vdash [a^*]P, \Delta}$$

Notice that the $j(x)$ is highlighted in red. This is how KeYmaera X tells you that it needs some input for this rule. In this case, it needs the loop invariant as input.

Loop Induction loop

$$\frac{\Gamma \vdash \texttt{l()<=x\&x...}, \Delta \qquad \texttt{l()<=x\&x...} \vdash [a]\,\texttt{l()<=x\&x...} \qquad \texttt{l()<=x\&x...} \vdash P}{\Gamma \vdash [a^*]P, \Delta}$$

We can then run the loop rule by clicking on its name. This will result in 3 subgoals, corresponding to the 3 premises of loop:

Goal: Loop Induction (129,0) | Goal: Loop Induction (129,1) | Goal: Loop Induction (129,2)

Hint: QE | ABBRV | HIDEL |

-1: $v \geq 0$ ⊢ 1: $l \leq x \wedge x \leq r$

-2: $l < x$

-3: $x < r$

loop

Proof Pro... | Rerun | Fresh steps | None | Execute: | Atomic | Step-by-Step

```
implyR(1) ; unfold ; loop({`l()<=x&x<=r()`}, 1) ; <(
  nil,
  nil,
  nil
  )
```

[1] In general, you can do this for any formula in the antecedents/succedents.

Notice, also that the `Proof Programming` box has been updated with a call to the `loop` tactic with the loop invariant we just entered as its first argument.

We can now continue with the proof on each branch. The first two branches are pretty trivial, they are just arithmetic questions so we can just use `QE` and they close immediately.

The third subgoal (showing that the invariant is preserved) looks a lot more complicated, so we let us try and prove it manually.

Note: 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. In fact, KeYmaera X always arranges the subgoals so that these "easier" ones come first.

As we saw with →R, there is really only one way in which the top-level hybrid program operator (a sequential composition) can be decomposed, using [;]. Instead of clicking through the proof, we could again just call `unfold` to do all of this simple top-level unfolding for us:

unfold | unfold

Hint: ODE | solve | dC | dI | dW | dG | boxAnd | GV | MR | chaseAt |

-1: $l \leq x$
-2: $x \leq r$
⊢
unfo 1: $[\{x' = v+1 \ \&\ l \leq x \land x \leq r\}](l \leq x \land x \leq r)$

The tactic also (helpfully) unfolded the choice operator and gave us two subgoals. For this proof, however, this is slightly annoying because this means we will essentially have to deal with the ODE twice (the same way) in both branches.

Note: Applying the most obvious top-most unfolding of your subgoals will usually work, but may not be the most efficient way of completing your proofs.

Recall that the point of doing this proof was to demonstrate how we did not even need the controller at all to prove the theorem, and in fact we do not need anything except the domain constraint. Instead of completely unfolding the proof, we can use `composeb` to break up the top-level sequential composition, and then use `GV` from the right-click menu:

Gödel/Vacuous GV

$$\frac{\Gamma_{const} \vdash P, \Delta_{const}}{\Gamma \vdash [a]P, \Delta}$$

Note: Notice the $\Gamma_{const}, \Delta_{const}$ in the premise of `GV`. In comparison to the G rule that we saw in class, KeYmaera X does its best to preserve any constant assumptions. In fact, it also does this for loop invariants. Recall from class that it would be a massive hassle to add all of the constant assumptions to your loop invariants all the time. Fortunately, KeYmaera X also does that for us.

After using `GV`, our goal looks really simple. In fact, `master` now will be able to finish the proof:

Goal: Gödel/Vacuous (133,0)

Hint: ALLR |

$$\frac{-1:\; l \leq x \land x \leq r \;\vdash\; 1:\; \forall v\, [\{x' = v \,\&\, l \leq x \land x \leq r\}]\,(l \leq x \land x \leq r)}{}\;\text{GV}$$

But let us go one step further and see more precisely what we meant when we said that we only needed the domain constraint. After getting rid of the forall quantifier, we are left with a differential equation in the postcondition. In its right-click menu, you will see a rule called dW:

Differential Weaken dW

$$\frac{\Gamma_{const}, Q \;\vdash\; p(x), \Delta_{const}}{\Gamma \;\vdash\; [\{x'=f(x) \,\&\, Q\}]p(x), \Delta}$$

Exercise 3:
Give an intuitive reading of this proof rule.
Answer: We have not yet seen the dW rule in class, but the menu in KeYmaera X helpfully tells us what the rule does. In order to prove the postcondition, we could just prove it assuming the domain constraint.
Note: KeYmaera X provides a huge library of tactics that would be impossible to cover completely in the course. Most of these tactics contain tooltips like the one above that concisely explains what the tactic does. Try exploring the right-click menu and the tactic browser if you ever get stuck in a proof.

The proof is trivial after using dW. In fact, we do not even need any arithmetic. Propositional proving with `prop` will suffice.

4.3 Tactics

The nice thing about doing proofs by clicking around on the UI is that it is 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 do not 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. The `Proof Programming` box contains the tactic for the proof that we just did:

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

As you can see, the syntax for tactics is much more intimidating for newcomers than the UI. For this reason, you should start out using the UI for your proofs, then use tactics to make repetitive tasks easier once you understand the structure of the proof.

What does the syntax mean anyway? The `;` symbols mean "run the left tactic followed by 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, like `loop`, have arguments, which go in parentheses, others do not. If a tactic operates on a whole subgoal (like `master` or `prop`) then we do not need to give it any numeric position 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‘}`.

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

Now, we have carefully done the proof so that it does not depend on the controller at all. The exact same tactic that we have just produced should also work if we replaced the controller with another controller. Let us try doing exactly that.

Note: The model for this is called rec4bad1_complex in the archive.

5 Another Bad Event-Driven Model

The fundamental problem with the previous model is that *domain constraints are part of our modeling assumptions*. For the ODE with domain constraint $l \leq x \leq r$, our model only has runs where the solution stays in $l \leq x \leq r$ for its entire duration.

This is like saying that there is a physical constraint that x can never leave the playing area, or that the whole universe lies between $l \leq x \leq r$. But, as we saw in last recitation, that is not a good model of reality. In the real universe, there are places other than the ping pong table, and we want to *prove* that the ball does not leave the table area.

Thinking ahead for the moment, we are interested in modelling an event-triggered controller. The players Forrest and Dan (the controllers) will be hitting the ball when $x = l$ and $x = r$ respectively. Thus, we will still need the domain constraint $l \leq x \leq r$ in order for physics to correctly yield to the controller when one of the events (when $x = l$ or $x = r$) happens. So let us keep the ODE that we have, but try and fix the problem that we have not modelled the rest of the universe.

In our more advanced model, we shall add an extra ODE whose domain constraint is the *logical negation* of the first ODE:

$$v \geq 0 \wedge l < x < r \rightarrow [\big(Ctrl_{bad}; (\{x' = v \,\&\, l \leq x \leq r\}) \cup \{x' = v \,\&\, x < l \vee x > r\})\big)^*] l \leq x \leq r$$

Note: The model for this is called rec4bad2 in the archive.

Note: In KeYmaera X input files, you can annotate loops with the `@invariant(...)` annotation. These annotations are useful because they provide readers with helpful insights into your models. In fact, KeYmaera X will also use these loop annotations in its proof automation.
Here is the tactic that proves this new formula:

```
unfold ; loop({`l()<=x&x<=r()`}, 1) ; <(
  QE,
  QE,
  composeb(1) ; GV(1) ; allR(1) ; unfold ; <(
    dW(1) ; prop,
    ODE(1)
    )
  )
```

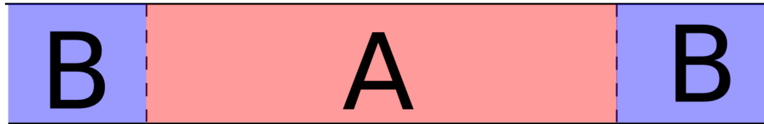
Notice that we have minimally changed the tactic for our earlier model. In fact, we only needed to change the tactic corresponding to the part of the model which we modified. In this case, we have two different ODEs that can be executed, so we have to deal with both cases separately.

However, remember that we are still using the bogus controller, so why did this proof succeed? The answer is that we have not quite split up reality correctly in our domain constraints. If we start on the ping pong table, then 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 never get to might as well not exist. Once we have 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. We will call "the ping-pong table" Mode A, and "everywhere else" Mode B. As shown below, 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 does not overlap Mode B, we will never transition to Mode B and so we never have to argue why Mode B is safe.

Mode A: l >= x and x <= r
Mode B: l < x or x > r

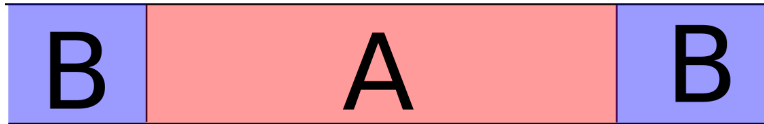


WRONG!

6 A Correct Model

We have to make Modes A and B overlap just a little bit, so that the model has a chance of even reaching either mode. This is done by turning the dotted line into a solid line, i.e., turning $x < l \lor x > r$ into $x \leq l \lor x \geq r$:

Mode A: l >= x and x <= r
Mode B: l <= x or x >= r



RIGHT!

Now we can finally try the proof with the bogus controller and the proof will no longer work. In fact, we can use KeYmaera X to spot where the proof gets broken. Following the proof steps similar to the ones we did before, here is one subgoal that you will reach that will not prove successfully, because it is not valid:

unfold | unfold

Hint: ODE | solve | dC | dI | dW | dG | boxAnd | GV | MR | chaseAt |

-1: $l \leq x$
-2: $x \leq r$
$\vdash$
unfo 1: $[\{x' = v \ \& \ l \geq x \lor x \geq r\}](l \leq x \land x \leq r)$

From the right-click menu, we can tell KeYmaera X to solve the differential equations, and then ask it for a counterexample:

Counter-example search result

Formula not valid, found a counter-example

Counter-example values

r	-12/5
x	-12/5
l	-12/5
v	-56

Speculated values for quantified variables

t_	3/5
s_	-9/5

Ok

What does this counterexample say? Consider the initial state where $x = -\frac{12}{5}, l = -\frac{12}{5}, v = -57$, then we can certainly run the differential equation until the postcondition is violated (in particular with $x < l$)!

Note: The model for this is called rec4bad2_fixed in the archive.

Note: There are several reasons why a subgoal cannot be proved successfully in KeYmaera X, here is a quick checklist:

- Your original formula was not valid. This is typically the common case, and inspecting the unprovable subgoal might allow you to figure out where your model could go wrong. This is one of the advantages of having a computer check your proofs for you.

- You used a proof step that was not complete. In particular, you could have lost some information when moving from the conclusion of the proof step to its premises. An important example of this is the loop rule – you might have chosen a bad loop invariant. When this happens, it is important to go back to your original question and see if you could have done the proof another way e.g., using an alternative loop invariant.

- You reached the current limits of KeYmaera X's proving capabilities. This will become relevant when we start looking at more complicated differential equations. Even though dL is theoretically complete for proving invariants of differential equations, the implementation in KeYmaera X does not cover all of the cases because they will almost never occur in practice.

- You found a KeYmaera X completeness bug. Our tactics can sometimes miss corner cases and some proofs may fail where they should not. If you are certain that your sequent is valid and your proof is correct but are not able to prove it in KeYmaera X, it might simply be a bug with the implementation of the tactics. Report it to us!

Finally, it is time to add in appropriate controls to the model. For illustrative purposes, we will assume that both players perfectly reverse the velocity of the ball when it is at their respective positions, i.e.,

$$Ctrl_{bad2} \stackrel{\text{def}}{\equiv} \textbf{if } x = l \lor x = r \textbf{ then } v := -v$$

Now, if we tried to prove:

$$v \geq 0 \wedge l < x < r \rightarrow [\big(Ctrl_{bad2}; (\{x' = v \,\&\, l \leq x \leq r\}) \cup \{x' = v \,\&\, x \leq l \vee x \geq r\})\big)^*] l \leq x \leq r$$

The proof would still fail!

Exercise 4:
Why?
Answer: The precondition says x could be anywhere $l \leq x \leq r$. In particular, when $x = l$ and the ball is traveling with velocity $v \geq 0$, the controller might flip the velocity around incorrectly.

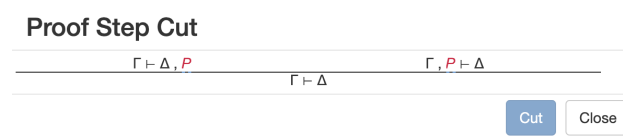
So we need to be very careful about our controller and ensure that the players actually hit the ball only when it is flying towards them.

$$Ctrl \stackrel{\text{def}}{\equiv} \textbf{if } x = l \wedge v \leq 0 \vee x = r \wedge v \geq 0 \textbf{ then } v := -v$$

The proof will still not quite work out, but for a rather subtle reason. We started off by assuming $l < x < r$, which implies $l < r$, but KeYmaera X does not know that. In particular, it does not know to keep the assumption $l < r$ in the loop induction.

Exercise 5:
What breaks without this assumption?
Answer: The domain constraint $x \leq l \vee x \geq r$ is equivalent to *true* when $l = r$.

We will have to start the proof slightly differently, by using a `cut` to introduce the new assumption $l < r$:



Now, the proof can be done successfully because `loop` helped us keep the assumption $l < r$ around in the proof. If we simply kept clicking on the UI though, we would get a rather tedious proof at the end:

```
unfold ; cut({`l() < r()`}) ; <(
  loop({`l()<=x&x<=r()`}, 1) ; <(
    QE,
    QE,
    unfold ; <(
      master,
      master,
      master,
      master,
```

```
      master,
      master,
      master,
      master,
      master,
      master
      )
    ),
  hideR(1) ; QE
  )
```

Instead, we could have programmed the proof more directly to hide the case splitting:

```
unfold ; cut({`l() < r()`}) ; <(
  loop({`l()<=x&x<=r()`}, 1) ; <(
    QE,
    QE,
    unfold ; master
    ),
  hideR(1) ; QE
  )
```

Note: The model for this is called rec4good in the archive.

Exercise 6:
Try to improve the speed of the above tactic.

7 Refining the Controller

We now have a verified controller but it is still unsatisfactory in some ways. For example, it assumes that the players are able to react instantaneously, regardless of how fast the ball is flying at them.

Let us try and relax this assumption by refining our controller so that the players will only hit the ball when $v^2 \leq 1$, and when hit the ball it loses a fraction of its velocity given by coefficient c:

$$Ctrl_2 \stackrel{\text{def}}{\equiv} \textbf{if } v^2 \leq 1 \land (x = l \land v \leq 0 \lor x = r \land v \geq 0) \textbf{ then } v := -cv$$

Note: Notice the pattern we have followed here: we first debugged and proved a simple controller model before moving on with the verification of a more refined/complex model. The simpler model debugging allowed us to catch many basic errors which might have been more difficult to catch if we had started straight away from the complex model.

Exercise 7:
We will need to some new assumptions on c and v initially. Suggest some possible ones.
Answer: We will certainly need $v^2 \leq 1$ and $0 \leq c \leq 1$ to be true initially.

Exercise 8:
What else do we need to change in order for the proof to work?
Answer: The loop invariant will not work anymore. We need to strengthen it with an assumption about v, namely $v^2 \leq 1$, so that the ball can actually be controlled by the players.

Exercise 9:
Complete the proof of the new controller.
Note: The model for this is called rec4good2 in the archive.