Recitation 11: More KeYmaera X modeling tips, Validation, Simulation and Synthesis
15-424/15-624/15-824 Logical Foundations of Cyber-Physical Systems

1 Motivation and Learning Objectives

Now that you are done with the second mid-term, the only thing left for you to do in this class is finishing your final project and preparing to compete in the CPS V&V Grand Prix. This (short) recitation focuses on KeYmaera X tips and tricks for modeling projects. We will also discuss validation and simulation, which may prove to be of great relevance to modeling projects. Finally, we will talk of program synthesis in the hybrid setting, using dL. This is an area of active research, but we will discuss some higher level ideas.

For course projects, please come talk to us separately if you need help/advice.

1.1 Creating the Model

Here are some simple tips for creating your model. Many of these should have appeared on your Betabot/Veribot/Project proposal feedback already but here is a quick summary:

1. Always **document** your model. Adding comments to your model makes it much easier for others (and for yourself in the future) to understand what you were trying to model.

2. Structure your model, e.g., using braces and spaces (no tabs) and make use of the KeYmaera X's support for definitions with the Definitions block. Use braces to group relevant parts of terms/formulas/programs together. For example, the controller for your hybrid program model could be explicitly put in between braces: they do not cost you anything and greatly improves readability.

   Creatively grouping parts of your formulas together could also help to make your proofs more straightforward. For example, if you have a conjunction $P \wedge Q \wedge R$ in your postcondition and you know that P, R are the "straightforward" ones, then perhaps writing your postcondition as $(P \wedge R) \wedge Q$ would make it easier to use $[]\wedge$ afterwards.

3. Matter of taste: you can use the special functions max, min, abs in your model which are interpreted with their standard mathematical meaning in KeYmaera X. This not only improves readability, but could also be more intuitive compared to encoding these functions with formulas.

   The downside of this is that tactics like dI and Mathematica do not play very nicely with these special functions. Make sure to remove them in your proof before calling QE if your QE call gets stuck.

4. Matter of taste: it could help to introduce additional variables or change the meaning of existing ones if it helps to simplify your model.

5. Think backwards: start from the "worst case" for your controller/model and then design the rest of your controller so that it ensures that even in this "worst case" your controller is still safe.

   In addition, this "worst case" can often feature as the default option for your controller that immediately ensures that your controller is never vacuous (just think of the braking option in all of the labs you have done).

6. Start from an easy model e.g., with an easy controller. Verify it before moving on to more complicated ones. The simplification might even be "good enough" if you can justify it.

7. Make sure that you ask the right questions in your model i.e., can you justify why the safety postcondition truly reflects your intuition as to what it means for the robot to be safe?

1.2 Proving the Model

Here are some simple tips for working on the proof of your model after you have created it:

1. When iterating between updating your model and proving it, do not bother with branches that you already know will work.

   However, remember to keep partial tactics so that it is easier to piece together your proof attempts afterwards.

2. Be cognizant of where you are "in your model". Since most of your dL proofs work by removing dL operators step by step, you should have a high-level intuition e.g., of which branch of a choice you are currently working on for your controller.

3. Try the ODE tactic for simple looking ODE goals. If that doesn't work, then go back and redo the ODE proofs more manually. We will see some particularly useful ODE proof rules that KeYmaera X implements later.

4. Remember that dI can work directly with disjunctions. There is no need to manually split disjunctive cases that you know will prove by dI.

5. Help QE along manually if necessary. It is not the best at dealing with complicated goals. If you have an intuition for why your goal is going to be true, then make it more obvious e.g., by cutting or hiding goals. Sometimes, even manually splitting disjunctions can help to speed it up.

2 Validation

As you may have realized by this point in the course, even after proving controller safe in dL, there is a very real possibility of modeling bugs. Subtle mistakes can prevent verified

dL models from being faithful representations of the reality. It is hard to get a model right; mistakes often creep in and confirming model behavior is important in real world projects.

As a first step, using software like Mathematica or python to plot the behavior of the more complex components of your model will allow you to confirm that the behavior of your model matches your intuition for how it should behave. In class we saw an example: we saw the sketch of a model of train with resistance polynomial in velocity that used the second Taylor approximation of velocity in order to get a good upper bound of velocity. By plotting the obtained bound, we were able to see when our model was more efficient than a baseline model that just ignored resistance, and confirm our intuition that this was a small but non-empty region.

Modelplex can be a powerful tool for validation. Accessed from the buttons on the right of the rows in the proof menu of KeYmaera X, it automatically generates a controller and run-time monitor for the proved model in c.

You can run the controller on test input to confirm that its behavior is as expected. The runtime monitor will also confirm that the physics is behaving as the model anticipated.

3 Simulation

Human intuition for how a model is supposed to behave can only go so far, and may be insufficient to catch mistakes in more complex models. So one may want to compare the behavior of a cyber physical system against a secondary source that n=computes the behavior of the same system.

This could be real world experiments, in the ideal case, which would immediately reveal model fidelity issues.

Otherwise, simulation can be a very good tool for validation. It is important to have simulation math be somehow independent of the math of your dL model, however, so that your results do not match by construction.

If external simulators are not available, one can also implement simulation themselves. But when parts of the physical dynamics are exactly the same as in the dL model, it is important to be cognizant of what parts of the system you are truly testing.

4 Synthesis

Program synthesis is an important and fundamental problem in computer science. In the cyber physical systems domain, the synthesis question is one quantifier above the verification question.

For example, we want to ask the question of how to write the controller for lab 2 given the environment and safety condition that it must satisfy, we may write something like the following:

```
\exists e (assumptions ->
[ {? x-o>e; a=A; ++ a= -B;}
```

```
    {x'=v, v'=a, t'=1 & t<=T & v>=0};
  ]x<=o)
```

In effect, we have provided the environment that the controller must navigate and the safety contract that it must satisfy. We want to algorithmically identify what the controller should then be by figuring out what value placeholder expression e should take.

Proving the formula listed indicates that there is a satisfying controller, and instantiating the quantifier $\exists e$ with a concrete value solves the problem of synthesizing a controller. We additionally may want the controller to be as permissive as possible, so that the overall resulting logical theorem is as strong as possible.

Conventional synthesis usually relies heavily on automated theorem proving and SMT solving. In the CPS domain, we can leverage the theorem proving ability of dL and KeYmaera X. However, theorems are hard to prove in dL so it is hard to scale to more complex questions.

The question of synthesis is closely related to the problem of proof search. One way to see the connection is the observation that if we can propose synthesis candidates and let the prover either confirm that our proposed controller is ok or give us a counterexample, then with enough of tries, we would be able to synthesize a controller, assuming a valid controller existed. Another way is to go back to the earlier listed formula. Imagine using automated proof tolls like unfold and auto in KeYmaera X until left with only arithmetic proof obligations involving the expression e. Then the problem of synthesis could be resolved

We must guide proof search, then. One of the basic techniques that can be used is abduction. This means that given a conclusion that we need to be true and rules we can use to prove the conclusion, we identify and the premises that we would require to be true. While abduction is straightforward and well-defined for the tactics with no arguments such as assignment or solution, it becomes harder when we need to provide arguments to the tactics. For example, in order to reverse differential cut, where the tactic requires the cut expression, which is hard to determine?

The problem then becomes identifying what arguments to provide to the tactic. One approach is to use standard templates to instantiate the arguments with heuristic guesses. We do not have all the answers yet; this is very much an active area of research.