Lab 1: Charging Station Revision 2
15-424/15-624/15-824 Foundations of Cyber-Physical Systems
TAs: Nathan Fulton (nathanfu@cs), Anastassia Kornilova (akornilo@andrew)

Betabot Due Date: Friday, January 29th **BEFORE 3:00pm Pittsburgh Time with NO late days**, worth 20 points
Veribot Due Date: Thursday, February 4th (2 late days, max 6 per semester), worth 70 points

Update!

KeYmaera X is undergoing active development, and we'll probably update in-between most labs as our favorite users find bugs. So, let's practice updating KeYmaera X!

Check if you need to update. To do so, **ensure you're connected to the internet** and then start KeYmaera X. Check the footer of any page on the KeYmaera X web interface. The footer should either say "KeYmaera X is up to date" or else should inform you that KeYmaera X is out of date. (Hint: you'll all need to update from 4.1b1 to 4.1b2!)

Whenever you notice is time to update, complete these steps:

1. Shutdown KeYmaera X
2. Delete your keymaerax.jar
3. Download the latest keymaerax.jar, which is always available from `http://keymaerax.org/keymaerax.jar`

Sections 1 through 3 will ask you to fill out some model templates and then construct safety proofs for the models you write down. The BetaBot submission only needs to contain the filled-out model templates, whereas the proofs are required for the VeriBot assignment.

1 Autobots, Roll Out

As the n00b in your robotics group, the senior members gave you the broken robot. Its steering is jammed, so it can only move in a straight line and control acceleration.

1 (BetaBot). Fill in the missing continuous dynamics in the hybrid program below that will model your robot accelerating in a straight line, with position `pos`, velocity `vel`, and acceleration `acc`.

2 (BetaBot). Fill in the safety condition that will ensure the velocity of the robot is never negative. Save this file as `L1Q1_name.kyx` (where name is replaced with your Andrew ID).

3 (VeriBot). Use KeYmaeraX to prove the velocity of the robot is always positive. Translate the resulting proof into a tactic and save the tactic in `L1Q1_name.kyt`. All of the tasks in this lab should prove automatically - but keep in mind that proving manually will significantly help be necessary in the later labs and it helps if you start early!

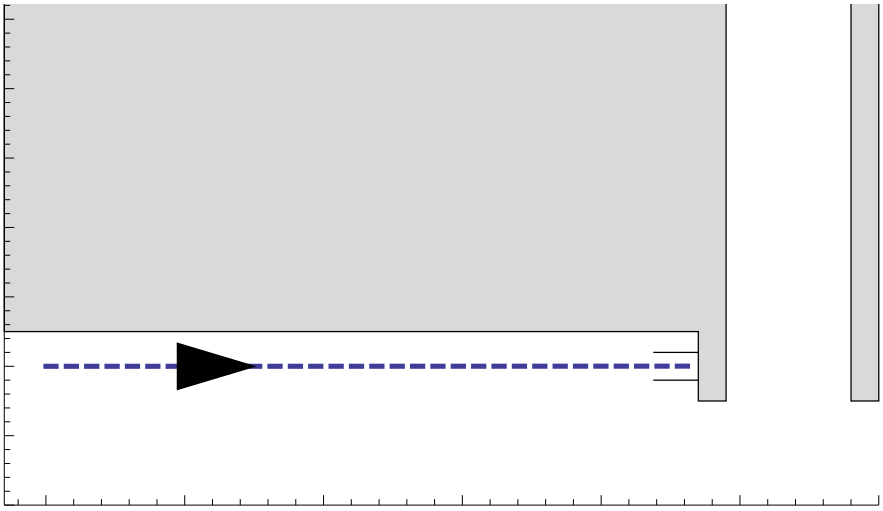

Figure 1: Charging station & wall.

```
(vel = 0 & acc >= 0)     /* Requires */
->
[{--------}]           /* Continuous dynamics */
(--------)               /* Ensures (safety condition) */
```

2 Charging Station, Single Control

Oh no! Your robot's battery is almost dead! Luckily the robot is already on a straight trajectory to a wall charging station and has positive velocity. With its remaining power, your robot has just *one chance* to engage the brakes properly to bring the robot to a stop at the right place.

- If the robot brakes too hard, it will not make it to the charging station and will have to wait for a human to plug it in. Running out of power is *inefficient*.
- If the robot doesn't brake hard enough to stop at the charging station, it will dent the wall behind the station. When building manager Jim Skees finds out, he'll have your robot banned from the building! Running into walls is *unsafe*.

Hints: think carefully about all the variables that the acceleration/braking should depend on. If you find you can't prove the property, it could be that you missed something and your property is falsifiable. Don't forget you learned how to find counter-examples in lab0!

1 (BetaBot). Specify the missing safety and efficiency requirements that the hybrid program below should ensure. Then fill in the missing control and continuous dynamics. Save the resulting file as `L1Q2_name.kyx`.

2 (VeriBot). Use KeYmaeraX to prove that the hybrid program is safe and efficient and translate the proof to a tactic. Save the resulting tactic as `L1Q2_name.kyt`.

3. **Question:** What is the evolution domain for the continuous dynamics in this hybrid program? Why is it necessary? Submit your answer in `discussion_name.txt`

```
(pos < station & vel > 0) /* Requires */
->
[
    acc := _________;      /* Assign a safe acceleration. */
    {____ & vel >= 0}     /* Use continuous dynamics from part 1. */
]
(_____________             /* Ensures (safety condition) */
 &_____________)           /* Ensures (efficiency condition) */
```

3 Charging Station, Double Control

In part 2, you are always able to coast the robot all the way to the charging station, since it is already moving. But what if the robot is stopped? In this question, the goals are the same as in part 2; however, the robot starts with zero velocity. You have two chances to control the robot, first to get it moving by accelerating, and then to bring it to a stop at the right point by braking.

The robot also has a time limit `T` on how long it can accelerate before exhausting the remaining battery life. Once the brakes are engaged, they will stay engaged until the robot comes to a complete stop.

1 (BetaBot). Save the filled in formula below as `L1Q3_name.kyx`.

2 (VeriBot). Prove the formula and save your tactic as (`L1Q3_name.kyt`).

3. **Question:** What is your efficiency condition? Is it different from part 2, why or why not? Submit your answer in `discussion_name.txt`

```
(pos < station & vel = 0 & 0 < T)        /* Requires */
->
[
    t := 0;
    acc := _________;                    /* Assign a safe acceleration. */
    {____, t' = 1 & vel >= 0 & t <= T};
    ?(t > 0);
    acc := _________;                    /* Assign a safe deceleration. */
    {____, t' = 1 & vel >= 0}
]
(_____________                           /* Ensures (safety condition) */
 &_____________)                         /* Ensures (efficiency condition)*/
```

4 Interacting with KeYmaeraX (VeriBot)

Recall the formula $\forall x(x > 0 \land x < 1)$ from lab 0. You will manually interact with this formula now, by trying to prove it. Since the formula is false you won't be able to, but you will see how doing so affects the proof tree.

Highlight the entire formula by hovering over the ∀ symbol and **right**-click. You will see a list of applicable proof rules. Use the "∀ R" rule by clicking on the name of the rule in the top-left portion of the box describing the rule (see Figure 6). This gets rid of the ∀ symbol, leaving you only with something like $x > 0 \land x < 1$. Notice how the previous formula moved below a horizonal line and you now have a new formula that was produced by the ∀R rule. The proof tree keeps track of everything you are doing to the original formula!

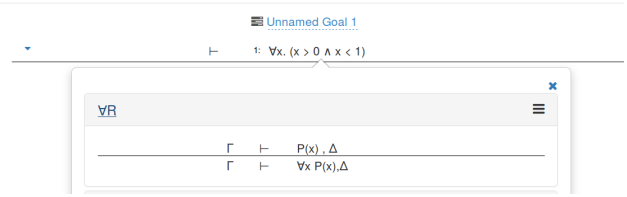

Figure 2: Applying the ∀ R rule by clicking on the ∀R text.

The formula under the Current Goal panel is now a conjunction, ∧. To prove a conjunction, both conjuncts need to be true. In this case, for $x > 0 \land x < 1$ to be true, you'd need to prove both $x > 0$ as well as $x < 1$. Hover your mouse over the ∧ symbol, and select "∧ R". This rule applies the reasoning we just made, where to prove $x > 0 \land x < 1$, you must prove each conjunct separately.

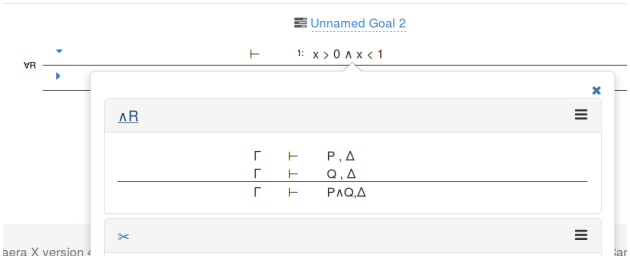

Figure 3: Applying the ∧ R rule.

Look at the tabs! Two cases were added, one for each conjunct. The tree is now *actually* a tree, not just a sequence of nodes! Each tab shows the current goal and the sequence of steps that produced that goal.

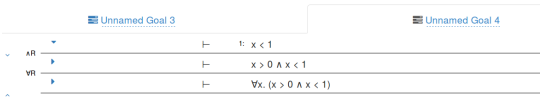

Figure 4: The proof tree is actually a tree now!

You can also re-label proof tree nodes, which can be helpful in large proofs (just like giving your variables descriptive names like "lander_acceleration" instead of "la11" is useful when programming in the large).

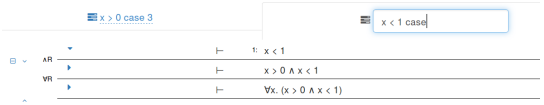

Figure 5: Naming open proof goals

Since you haven't proven either of the cases, they are called *open goals*. Select each open goal, and notice how KeYmaeraX updates your screen to show the selected goal. Each goal should represent one of the conjuncts! In a bigger proof, you'd now continue to try to prove each conjunct separately, growing that part of the tree.

5 New Procedure for Exporting your Database

We've simplified the process for exporting your database. When you're ready to submit your VeriBot, export your database by clicking the "Extract Database" link in the help menu.

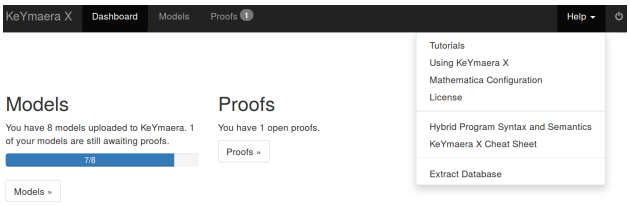

Figure 6: Accessing the database export tool

After clicking the "Extract Database" button, a modal dialog will appear telling you a location on your file system where KeYmaera X placed your exported database. Copy and rename this file per the instructions in the submission checklist below.

6 Submission Checklist

Use the provided templates, and *do not forget to fill in the section at the top with your AndrewID and KeYmaera X version.* It gives us important information when grading your submission!

1. **BetaBots:** submit a zip file on autolab containing your preliminary .kyx files for each of the tasks. This will enable us to give you feedback halfway through the assignment, so that you don't get stuck! If you want, you can include some *small* comments about your approach and questions you might have.
   Due Friday 01/29

   The BetaBot zip file should contain (where "name" is replaced with your AndrewID):

   - `L1Q1_name.kyx`
   - `L1Q2_name.kyx`
   - `L1Q3_name.kyx`

2. **Final submission (VeriBots):** the final submission works the same way, but you must include your final model in a .kyx file as well as tactic that produces a closed proof in the .kyt file. To receive credit, proofs must be complete (i.e. the "Proof Found" window appears after copy/pasting the contents of the kyt file into the tactic dialog in KeYmaera X). In addition, include your database file that contains all your proof attempts.
   Due Thursday 02/04

   The zip file should contain (where "name" is replaced with your AndrewID):

   - `L1Q1_name.kyx`
   - `L1Q1_name.kyt`
   - `L1Q2_name.kyx`
   - `L1Q2_name.kyt`
   - `L1Q3_name.kyx`
   - `L1Q3_name.kyt`
   - `keymaerax_exported_name.sqlite`
   - `discussion_name.txt`