Lab 1: Charging Station
15-424/15-624/15-824 Foundations of Cyber-Physical Systems
TA: Brandon Bohrer (bbohrer@cs.cmu.edu)
Revision: 2

Betabot Due Date: Friday, February 3rd **BEFORE 3:00pm Pittsburgh Time with NO late days**, worth 20 points
Veribot Due Date: Thursday, February 9th (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 The footer should either say "KeYmaera X is up to date" or else should inform you that KeYmaera X is out of date. (e.g. update from 4.3.1 to 4.3.2)

Whenever you notice it 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`

Usually, that's all you have to do. Sometimes (including this time) we have to change the file format used by KeYmaera X's internal database. When this delete or move/rename the current database. The database contains all your proofs and models, so if you move it instead of deleting it you can still access the proofs with an older KeYmaera X. But at this point you've just got the Lab 0 proofs you proved by master, so deleting the database won't hurt. The database is spread throughout 3 files:

1. `~/.keymaerax/keymaerax.sqlite`
2. `~/.keymaerax/keymaerax.sqlite-shm`
3. `~/.keymaerax/keymaerax.sqlite-wal`

The default Windows shell doesn't know what `~` but it's just `C:\Users\<your_username>`.

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 containing your conjecture. Proofs are required for the VeriBot assignment.

1 Note: Syntax

In this lab you will start writing more complicated hybrid programs, safety properties and proofs by yourself. Pay careful attention that the models you write down mean what you think they mean, and use parentheses when you're not sure. For example $A \wedge B \implies C$ parses as $(A \wedge B) \implies C$, which is easily confused for the very different proposition $A \wedge (B \implies C)$. When you need parentheses to group statements in a hybrid program, use curly braces instead, like this:

```
{x := 1;
 ?(x < 2 & x>0);
} ++
x := 1;
```

As in C, note that the KeYmaera X syntax uses semicolons as statement terminators for atomic statements like `x := 0;` and `?x < 1;` but does not need extra statement separators. For example, when sequencing two nondeterministic choices, we do not separate them with a semicolon:

```
{x := 0; ++ x := 1;}
{y := 0; ++ y := 1;}
```

2 Autobots, Roll Out

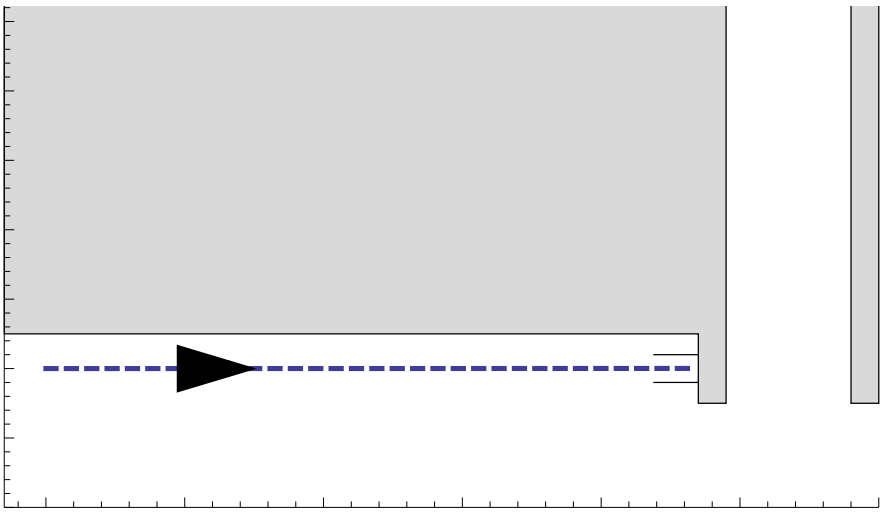

Figure 1: Charging station & wall.

As the n00b in your robotics group, the senior members gave you the broken robot. Of course. 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.kyx`.

3 (VeriBot). Use KeYmaeraX to prove the velocity of the robot is always positive. Download the resulting proof archive as `L1Q1.kya`. All of the tasks in this lab should prove automatically, but keep in mind that manual proof parts will significantly help scale to more complex challenges in later labs and it helps if you start early!

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

3 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. You never know how long it takes for the the next human to come by. 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 forever! 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.kyx`.

2 (VeriBot). Use KeYmaera X to prove that the hybrid program is safe and efficient. Download the resulting proof archive as `L1Q2.kya`.

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

4 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 your 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.kyx`.

2 (VeriBot). Prove the formula and download the proof archive as (`L1Q3.kya`).

3. **Question:** What is your efficiency condition? Is it different from Part 2, why or why not? Submit your answer in `discussion.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)*/
```

5 Implementing an Interpreter

To get some practice with semantics, you're going to program the semantics of first-order logic! That is, for terms you will write a function that computes their value in a given state. For formulas (which might have quantifiers) you will write a function that computes whether they are true in a given state. We use the same definition of first-order logic as given in Definitions 2.2 and 2.3 of the Lecture 2 notes:

`http://symbolaris.com/course/fcps17/02-diffeq.pdf`

Terms are defined as

$$\theta ::= x \mid c \mid \theta_1 + \theta_2 \mid \theta_1 \cdot \theta_2$$

where x is a variable and c is a literal number like 1 or 2.5.

Formulas are defined as

$$P, Q ::= \theta_1 \geq \theta_2 \mid \theta_1 = \theta_2 \mid \neg P \mid P \wedge Q \mid P \vee Q \mid P \rightarrow Q \mid P \leftrightarrow Q \mid \forall x P \mid \exists x P$$

A few parts of this are tricky and hard to do exactly in a computer program. Most real numbers require infinite storage space, making them hard to implement on a computer. Instead you should use an approxima-

tion of the real numbers, such as floats or rationals. Floats will most likely be easiest. [1] Quantifiers are also problematic because there are so many reals we could never try them all. When instantiating quantifiers, pretend the only real numbers are $0, 0.5, 1, \ldots, 10$. [2]

Your interpreter also needs to manage the *state* telling you which variables have which values. This is abstracted out into a state data structure which maps variables to their current values. This structure is provided and should have all the operations you need for the assignment, but you will need to use it correctly.

Because you might not all know the same programming languages, we have provided starter code in 3 languages: Standard ML, Java, and C++. The size of each file is indicative: if you know Standard ML, you should probably use that because functional languages make it very easy to write an interpreter. You should also feel free to use the language of your choice for this problem and translate the starter code. If it won't be totally obvious to the TA what language you're using, please indicate that in your solution.

1. Implement the two `interpret` functions, one for terms and one for formulas. Submit one file: `src/L1Q4.sml`, `src/L1Q4.java`, `src/L1Q4.cpp`, or whatever the file is named in your favorite programming language.

   For the Betabot, submit your progress so far (which should include, at the very least, writing new tests). For the Veribot, submit the completed interpreter.

2. (Veribot) Write a formula and a state for which your interpreter says it's true, but it should be false. Why is that? Write your answer in `discussion.txt`

3. (Veribot) Write a formula and a state for which your interpreter says it's false, but it should be true. Why is that? Write your answer in `discussion.txt`

If you're curious about how this works for real when there are quantifiers, we've never had a final project on quantifier elimination before :-).

6 Interacting with KeYmaera X (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 $\forall$ symbol and **right**-click. You will see a list of applicable proof rules. Use the "$\forall$ R" rule by clicking on the name of the rule in the top-left portion of the box describing the rule (see Figure 2). This gets rid of the $\forall$ symbol, leaving you only with something like $x > 0 \land x < 1$. Notice how the previous formula moved below a horizontal line and you now have a new formula that was produced by the $\forall$R rule. The proof tree keeps track of everything you are doing to the original formula!

The formula under the Current Goal panel is now a conjunction, $\land$. 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 $\land$ symbol, and select "$\land$ R". This rule applies the reasoning we just made, where to prove $x > 0 \land x < 1$, you must prove each conjunct separately.

[1] For a sound way of approximating the reals, ask the TA about his research.

[2] Writing this interpreter with exact reals and quantifiers is the same as implementing *quantifier elimination*, an important tool used to solve arithmetic questions in KeYmaera X.

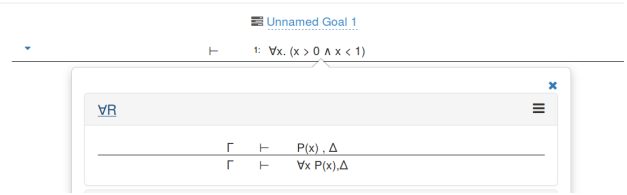

Figure 2: Applying the ∀ R rule by clicking on the ∀R text.

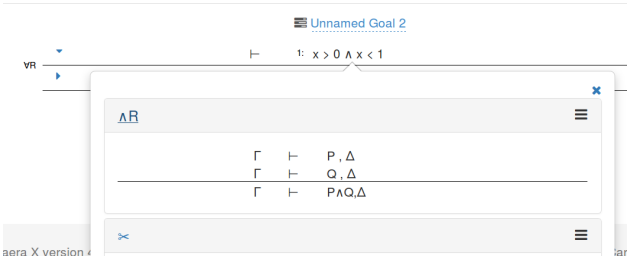

Figure 3: Applying the ∧ R rule.

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.

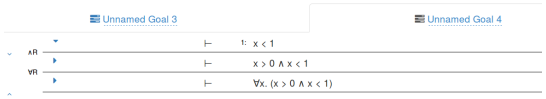

Figure 4: The proof tree is actually a tree now!

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).

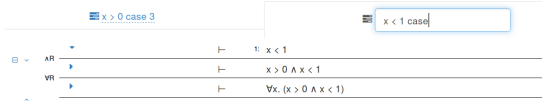

Figure 5: Naming open proof goals

Since you haven't proven either of the cases, they are called *open goals*. Select each open goal, and notice how KeYmaera X 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.

7 Submission Checklist

Use the provided templates, and *do not forget to fill in the section at the top.* 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 02/03

   If you have GNU make installed, you can create the `.zip` handin file using the `Makefile` in the handout `.zip`. Use the make target for whatever language you wrote the interpreter in (i.e. run one of the following):

   - `$ make handin-sml`
   - `$ make handin-java`
   - `$ make handin-cpp`

   The BetaBot zip file should contain:

   - `L1Q1.kyx`
   - `L1Q2.kyx`
   - `L1Q3.kyx`
   - `src/L1Q4.sml` or `src/L1Q4/interpreter.java` or `src/L1Q4/interpreter.cpp`

2. **Final submission (VeriBots):** the final submission works the same way, but instead of the .kyx files you download and submit the .kya files, which include both the model and your proof tactic. To receive credit, proofs must be complete (i.e. the "Proof Found" window appears after copy/pasting the tactic into the tactic dialog in KeYmaera X and running it).
   Due Thursday 02/09

   The zip file should contain:

   - `L1Q1.kya`
   - `L1Q2.kya`
   - `L1Q3.kya`
   - `src/src/L1Q4.sml` or `src/src/L1Q4.java` or `src/src/L1Q4.cpp`
   - `discussion.txt`