Lab 2: Follow the Leader
15-424/15-624/15-824 Logical Foundations of Cyber-Physical Systems
TAs: Irene Li (mengzeli@andrew.cmu.edu)
Yong Kiam Tan (yongkiat@cs.cmu.edu)

Betabot Due Date: Friday, September 28th **BEFORE 12:00 noon with NO late days**, worth 20 points
Veribot Due Date: Thursday, October 4th, 11:59PM (2 late days, max 6 per semester), worth 80 points

Lab Resources: `http://symbolaris.com/course/lfcps18/lab2.zip`

Update!

KeYmaera X is undergoing active development and we will sometimes release updates in-between labs as our favorite users find bugs. So, here is a quick guide to updating KeYmaera X!

Check if you need to update. To do so, **ensure that you are 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 version ... (latest release)" or else it should tell you that a new version is available (e.g., "version 4.6.0 is now available ...")

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 from: `http://keymaerax.org/keymaerax.jar`

Usually, that is all you have to do. The release notes at: `http://keymaerax.org/download.html#News` contains all of the changes made in each release and should also tell you about any incompatibilities. For the upgrade from KeYmaera X 4.5.0 to KeYmaera X 4.6.0, we have also improved the syntax for KeYmaera X's input files (e.g., the .kyx and .kya files). You will notice the changes in the template files for this lab. Briefly:

1. The period character '.' is used to end blocks and meta information. It is no longer needed at the start of each block e.g., instead of "ProgramVariables." write "ProgramVariables" instead.
2. The type identifiers "R" and "B" have been renamed to "Real" and "Bool" respectively.
3. Declarations now end with ';' rather than '.'.

1 Event-triggered Highway Driving

In this problem, you will design a hybrid program (HP) to model a controlled car (`ctrl`) following a lead car (`lead`) along a straight road. The requirements for the model are listed below in text; you should write a HP that appropriately models these requirements.

- The lead car should keep a constant and non-negative velocity (i.e., $vel_{lead} \geq 0$).
- The driver of the controlled car can only choose to accelerate at rate A, where $A > 0$, or brake at rate $-B$, where $B > 0$. The choice of acceleration A should only be available to the driver when it is safe, a condition that you will have to define, while the choice to brake should always be available.
- The controlled car has continuous access to the lead car's position and velocity (i.e., the controller you design should be *event-triggered*).
- Assume the cars are infinitesimal points. In other words, a crash occurs only if the position of the controlled car exceeds the position of the lead car (i.e., a crash occurs only if $pos_{ctrl} > pos_{lead}$).

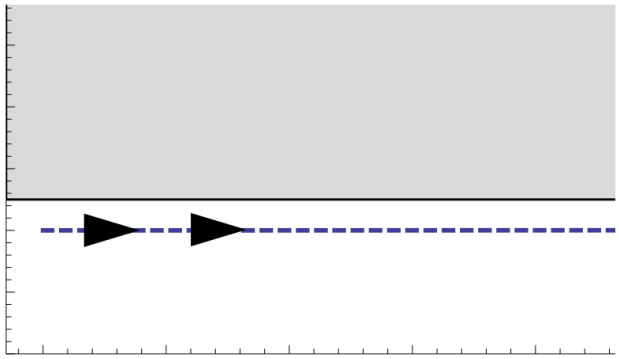

Figure 1: Lead and control car

- Your controller must always have a transition, otherwise the safety property would be vacuously true in some (or all) cases. For example, the safety theorem $[?\mathit{false}]safe$ holds vacuously for any postcondition $safe$ because the test $?\mathit{false}$ always fails. To avoid this, make sure your tests work like if-then-else statements.

  If you guard control decisions with a series of tests like: $(?\phi_1;\alpha_1) \cup \ldots \cup (?\phi_n;\alpha_n) \cup \beta$, make sure the conditions ϕ_i's are exhaustive, i.e., in every state, at least one of the guards is true, or that you have a fallback option β which is always available.

1. (Betabot). Write your answers to these three questions in lab2.txt:
   - What is a good *safety condition* for this system?
   - Under which *initial conditions* would the system satisfy your safety condition? For example, if the cars already start at $pos_{ctrl} > pos_{lead}$ then they have already crashed before your controller could do anything at all to save the day!
   - What would be a good criteria for claiming that your model is *efficient*?
2. (Betabot). Fill in the missing parts of the HP in the given template file. Also fill in your **safety** and **initial** conditions from the previous part as logical formulas in the template. Save this file as L2Q1.kyx.

   While a proof is not required at the Betabot due date, you should strive to get the model and controller correct, because that will give you a better basis for the Veribot that you will be proving.
3. (Veribot). Use KeYmaera X to prove that the HP you designed satisfies your safety condition and download the resulting L2Q1.kya file.
4. (Veribot). **Bonus:** Drivers get uncomfortable when their car gets too close to the car ahead. Update your safety condition to require that the cars never come within a constant distance `c` of each other. Update your model to satisfy this requirement and prove it safe in KeYmaera X. Submit the resulting file as L2Q1_bonus.kya.

   Only attempt the bonus problem **after** successfully proving safety without the buffer. You will only get bonus credit if your model without the buffer is proved successfully in the Veribots submission.

2 Time-triggered Highway Driving

In this problem, you will increase the fidelity of your model by changing some of the modeling assumptions. First, the lead car is now allowed to arbitrarily either accelerate at rate A or brake at rate $-B$. Second,

when your (controlled) car chooses an acceleration, it may be stuck with that choice for some time. You will therefore need to design a time-triggered controller instead of an event-triggered one. Your model will now have a "stopwatch" which must be set to 0 before each continuous evolution.

- The lead car may accelerate or brake arbitrarily at rate A or $-B$ respectively. The controlled car never has access to the lead car's acceleration.
- In the event-triggered controller, your car could only accelerate at rate A or brake at rate $-B$. This means that once it comes to a stop, it has no option but to accelerate. If acceleration is not safe, then the controller has no control options and the safety property would become vacuously true. To address this issue and avoid vacuously true theorems, in this problem you are additionally allowed to set your car's acceleration to 0.
- The controlled car has intermittent access to the lead car's position and velocity. The time between updates is variable, but is guaranteed to be less than time T (i.e., your controller must be *time-triggered*).
- The safety property should never be vacuously true (i.e., the transition semantics of your hybrid program should not be empty). Ensure that the tests you use for guarding control decisions are exhaustive.

1. (Betabot). Using the given template, design a time-triggered controller and model the system as a hybrid program. Then, write a dL formula expressing safety under suitable initial conditions for this new controller. Submit this file as L2Q2.kyx.
2. (Veribot). Use KeYmaera X to prove that your time-triggered controller is safe. Download the resulting L2Q2.kya file.
3. (Veribot). **Question:** Compare and contrast the Event-triggered and Time-triggered highway driving. Describe their relationship. Which was easier to prove safe? Which would be easier to implement? Why and what caused these differences? Submit your answer to this question in lab2.txt.
4. (Veribot). **Question:** Suppose now that the lead car has faulty brakes. When it decides to brake, it could be braking at any one of rates $-B, -\frac{B}{2}, -\frac{B}{4}$ or $-\frac{B}{8}$ instead. Is your controller still safe? Explain why or why not. Submit your answer to this question in lab2.txt.

3 Submission Checklist

This lab and all remaining labs in this course may be submitted in groups of two. If you are working with a partner, then you must submit a file called andrewids.txt containing both of your Andrew IDs. To make the grading infrastructure happy, please put them on a single line separated by a space, e.g.:

```
mengzeli yongkiat
```

Make sure you submit this file when working in a group. Otherwise, one of you will not get credit because there is no record of your submission. Additionally, ONLY one of you should submit on Autolab so that we do not end up grading your submissions twice.

For both Betabot and Veribot submissions, remember to check the Autograder's output on Autolab to ensure that your files were submitted in the right format, parse correctly, etc. If you are working in a group, please also ensure that the Autograder correctly reports your Andrew IDs in its output, e.g.:

```
==> Group Andrew IDs: mengzeli, yongkiat
```

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. **Initial submission (Betabot)**. Submit a zip file on Autolab containing your preliminary .kyx files for each of the tasks as well as the Betabot discussion file. This will enable us to give you feedback halfway through the assignment, so that you do not get stuck! If you want, you can include some *small* comments about your approach and questions you might have.

   While a proof is not required at the Betabot due date, you should, nevertheless, strive to get the model and controller correct, because that will give you a better basis for the Veribot that you will be proving. It will also result in a higher Betabot grade and allow us to provide more useful feedback.

   The Betabot zip file should contain:

   - `L2Q1.kyx`
   - `L2Q2.kyx`
   - `lab2.txt` (with your answer to Q1.1)
   - `andrewids.txt` (only if working in pairs)

2. **Final submission (Veribot)**. The final submission works the same way, except you submit `.kya` files (which contain both the model and the proof) and your Veribot discussion file. To receive full credit, proofs must be complete (i.e., a successful "Proof Result" window appears after running the tactic you have submitted).

   The Autograder for this lab (and all remaining labs) will not check that your proof works automatically because the models and proofs are a lot more complicated. Thus, you will not see any `==> Succeeded at proving ...` lines in its output.

   The Veribot zip file should contain:

   - `L2Q1.kya`
   - `L2Q2.kya`
   - `L2Q1_bonus.kya` (only for bonus credit)
   - `lab2.txt` (with your answer to Q2.3 and Q2.4)
   - `andrewids.txt` (only if working in pairs)