ECE 522/622 - Spring 2025
Modeling and Verification of Embedded Systems
Lab #3: Hybrid Systems Verification

Objectives:

1) To understand the tradeoff between precision and complexity in hybrid systems verification
2) To gain exposure to reachability analysis using piecewise over-approximation of system dynamics
3) To gain exposure to modeling and verification using the PHAVerLite tool

General instructions:

1) You can work alone or in groups of 2.
2) Lab reports should be typed, with name and ID at the beginning of the lab report. If working in a group of 2, the report must include both names.
3) Some of the steps require that you name files according to your last name; if you are working in a group of 2, the files should be named according to the person that will submit the materials on Canvas for the group.
4) All screenshots, plots, and text should be clearly readable.
5) You should complete problem set 3 before this assignment. The instructions in this assignment assume that you have completed problem set 3 to learn PHAVer syntax and set up the PHAVerLite and graph tools.
6) Only groups that have one or more ECE622 students are required to complete step C.4.
    - ECE522 students can complete part C.4 for extra credit (total not to exceed 100 points).
    - ECE522 students that skip part C.4 will have the other scores scaled up proportionally.

Sources:

- G. Frehse, "PHAVer: Algorithmic Verification of Hybrid Systems past HyTech". *International Journal on Software Tools for Technology Transfer*, 263–279 (2008).
- PHAVer Manual: www-verimag.imag.fr/~frehse/phaver_web/phaver_lang.pdf

Introduction:

Hybrid systems have continuous and discrete dynamics. The continuous dynamics are governed by equations in each location. The discrete dynamics are the jumps between locations. A common approach for hybrid verification is to represent continuous dynamics by a simpler piecewise over-approximation. PHAVerLite uses this approach. The conservative over-approximate model it uses for reachability analysis is a sound abstraction of the original hybrid model described (precisely) by equations. The abstract model will reach all parts of the state space that the original model can reach, but due to its over-approximation may also reach additional parts of the state space. This has two important implications:

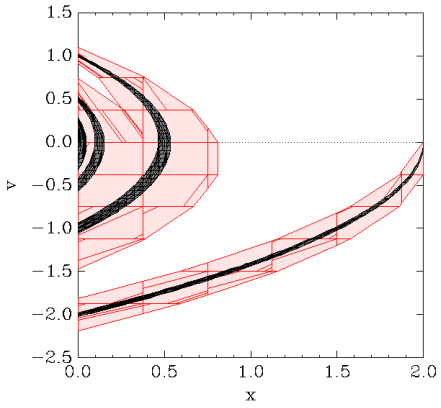
1. Parts of the state space that are unreachable for the abstract (over-approximate) model are also unreachable for the original (precise) model. Therefore, we **can use** the **abstract model to prove that parts of the state space are unreachable.**
2. Parts of the state space that are reachable by the abstract model may or may not be reachable for the original model. Therefore, we **cannot use** the **abstract model to prove that parts of the state space are reachable.**

This lab will demonstrate the importance of using the right level of abstraction when verifying properties of hybrid systems. A model that is too abstract may not be precise enough to prove a condition is unreachable. A model that is insufficiently abstract can incur high runtime to prove that a condition is unreachable.

PART A: Analysis of Bouncing Ball Model [20 pts]

First you will analyze again the bouncing ball model from problem set 3 and module 3 of lecture. You will explore the impact of trying different levels of abstraction by varying the value of the partition constraint (pc). Use the bouncing ball model (`bouncing_ball.pha`) from Canvas as a starting point to answer the following questions.

A.1) Use PHAVerLite to compute the reachable states of the bouncing ball model using pc=0.5, and then again using pc=0.05. Generate a single plot that shows the reachable states for pc=0.05 on top of the reachable states for pc=0.5. It should look like the one shown at right. The style of your plot can deviate from the figure shown, but must include both sets of reachable states on the same axes, and the two sets of reachable states must be clearly distinguishable based on color, shading, etc.



- Add the plot to your report.
- Based on the plot, when using pc=0.5, is x=0.6 and v=0.0 unreachable? Explain briefly.
- Based on the plot, when using pc=0.05, is x=0.6 and v=0.0 unreachable? Explain briefly.

A.2) Now try to prove that x=0.6 and v=0.0 is unreachable using the "`is_reachable`" command in PHAVerLite.

- First check reachability (using "`is_reachable`") for pc=0.5 and pc=0.05 to make sure that the outcome reported by the tool matches your expectation based on the plot from A.1
- Repeat the reachability check when pc is set to the values 0.5, 0.2, 0.1, 0.05, 0.01, and 0.005. If the runtime for any value of pc exceeds 15 minutes, you can report the runtime as "timed out." Fill in a table as shown below with the results of these reachability checks. Include this table in your report.

Partition Constraint PC	Condition reachable?	Num Locations reached	CPU Time (sec)
0.5			
…			
0.005			

- If you repeated this reachability check for pc=1.0, pc=0.15, and pc=0.0001, which of the three case(s) would definitely find the target state to be unreachable. Answer based on your understanding and the results in the table, without performing additional reachability checks. Briefly explain your answer.

A.3) We can see in the plot above that the ball rises less high with each bounce -- the ball is released at height of 2.0 but only rises to around 0.5 on the 1st bounce. This occurs because the upward velocity after bouncing is smaller than the downward velocity before the bounce.

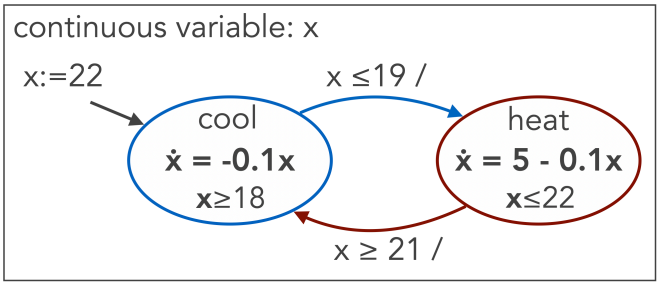
- Set pc to 0.05 in all parts of this step, so that the trajectory matches the plot above. Based on the existing plot of reachable states, does it look like the ball can have a height of exactly 0.15 at the peak of the 1st, 2nd, 3rd, and 4th bounces? Explain.
- Modify the bouncing ball to add another variable "n" that keeps track of how many times the ball has bounced. Remember that you will need to add the variable, initialize its value, specify its derivative within each location, and update its value on a jump.
- You should also add a location invariant to ensure that n does not exceed 10. This bounds the space that PHAVerLite must explore. Without this invariant, the runtime will be high.
- Add four `is_reachable` queries to the model to check whether x=0.15 and v=0.0 (i.e. peak height is 0.15) is reachable on the 1st bounce, 2nd bounce, 3rd bounce, and 4th bounce. Name the four conditions as n1, n2, n3, and n4 and check them with the four PHAVerLite commands below:

```
check=bball.is_reachable(n1);
check=bball.is_reachable(n2);
check=bball.is_reachable(n3);
check=bball.is_reachable(n4);
```

- Compare these results against your understanding from the plot to know whether your model is correct.
- Name the model, with all four queries, as `<lastname>_A3.pha` and upload it with your submission.

PART B: Model of Heating System [30 pts]

In this question, you'll be analyzing the heating system model shown at right. The model is the same one we analyzed manually in module 3, except that there are location invariants added so that the system cannot stay in each location indefinitely after the guard becomes true. You'll write this model using PHAVer syntax and then explore reachability in its state space using PHAVerLite.



Because the property being checked involves time, you will need to add a time variable to your model. The time variable should be initialized to 0, should retain its value across jumps, and should have a derivative of 1 in both locations. Add location invariants so that time is never below 0 or above 10 in either location (in other words, $0 \le t \le 10$). If you don't add these invariants, PHAVerLite has an unbounded space to explore, and may run for a long time since it can keep reaching new parts of the state space forever.

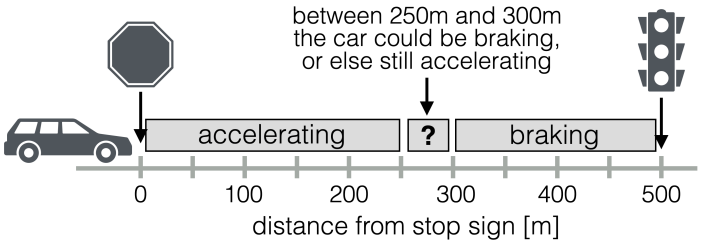
B.1) Create the model and use PHAVerLite to compute reachable states of this model with partition constraint (pc) value of 0.3. Add the plot of reachable states to your report. The plot should have an x-axis representing time, and a y-axis representing temperature.

B.2) Next, check reachability using different abstractions by varying the value of pc. The property to verify is as follows: **from the initial condition (cool, t=0, x=22), the temperature will go below 20 degrees within 0.98 seconds**. As we showed in lecture (see slide 79 of module 3), the method of verifying this property is to show that over-approximate reachability cannot reach parts of the state space that would disprove the property. The slide used $x>20$ and $t \ge 0.98$ as the counterexample, but here you should use $t==0.98$ instead of $t \ge 0.98$ because the system can reach $x>20$ at later times after jumps, which should not be considered as a counterexample to our property.

- Add to your model the appropriate "`is_reachable`" command for checking this property.
- Find the largest value of pc for which you can show the condition to be unreachable. How did you find this value of pc? The answer need not be exact but must be within 0.1 of the exact answer. In other words, your answer will be considered correct if it shows the condition to be unreachable, but a pc value that is 0.1 larger shows the condition to be reachable. PHAVerLite partitions the space slightly differently from the method we used in lecture, so you should not assume perfect agreement between PHAVerLite and the analysis we performed in class.
- Submit your model, including the reachability query, with largest value of pc that shows the condition to be unreachable, as `<lastname>_B2.pha`.

PART C: Model of Car Acceleration and Braking on Road [50 pts]

In this question, you'll be analyzing a simple model of a car on a road as described below. You'll have to model the system using PHAVer syntax, and then explore reachability in its state space using PHAVerLite. The model involves velocity and acceleration / deceleration of a car. The bouncing ball model from part A also involved velocity and acceleration, so it may be helpful to review that model if you are otherwise not sure how to proceed.



The scenario you are modeling has two intersections that are 500 meters apart (see figure). The first intersection has a stop sign, and the second has a traffic light.

- A driver makes a full stop at the stop sign and then accelerates toward the second intersection at 3.5m/s^2
- When the driver sees that the traffic light is red at the second intersection, she applies the brakes. At the earliest, she applies the brakes when 250 meters past the stop sign (250 meters from the traffic light). At the latest, she applies the brakes when 300 meters past the stop sign (200 meters from the traffic light).
- When braking, the car decelerates at a rate of 5.5m/s^2 until stopping; in other words, acceleration while braking is -5.5m/s^2.

C.1) Draw a model of your system, with each location represented by a circle, as in the picture given in part B. The model should have one variable representing position, and another variable representing velocity.

- Label the guard conditions on each jump.
- Label the derivative of position and velocity in each location.
- Label the location invariants of each location. Remember that hybrid systems are not forced to jump when guards evaluate to true, so location invariants are needed if you want to force a transition.

C.2) Translate your model into PHAVer syntax so that you can check reachability using PHAVerLite. As before, you will want to use reasonable location invariants to bound the search space of the tool. With the pc value set to 3.0, generate a plot that shows the reachable states. The x-axis should be the position of the car, and the y-axis should be the velocity of the car. Add the plot to your report.

C.3) Now that you've created the model, you will further analyze it. For each of the three properties in the table below, add a reachability check to prove that it is true, or to prove that it is false. Remember that, due to over-approximation, your answer must always be based on showing something to be unreachable. You can choose the value of pc. Include the following in your report:

- Your exact query, which must use the `is_reachable()` command.
- A screenshot showing the text output from PHAVerLite with the result all 3 reachability checks.
- An explanation of why the result of the reachability check proves that the property is true, or why it proves that the property is false.
- Upload the model, with all 3 queries and your chosen value of pc, as `<lastname>_C3.pha`.

Property 1	The car will definitely stop before it reaches the 2nd intersection
Property 2	The velocity cannot be above 46.5 m/s while the car is braking
Property 3	The velocity cannot be above 40.0 m/s while the car is braking

C.4) **ECE622 only:** Use PHAVerLite to estimate the 0-60MPH time of the car in this example. 60MPH is equivalent to 26.8m/s. You will need to add a time variable to the model to track when the car reaches this velocity. Use 3.0 as the value of pc. Your model should have two reachability queries. The first should show that velocity 26.8m/s is unreachable within a certain time, and the second should show that velocity 26.8m/s is reachable within a time that is 0.1 seconds greater. Based on that finding, give a range of values for the 0-60MPH time of this car. Upload the model, with both queries as `<lastname>_C4.pha`.