ECE 597MB/622 - Spring 2022
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 hybrid systems modeling and verification using PHAVer

General instruction:

1) Lab reports should be typed, with name and ID at the beginning of the lab report.
2) You can work alone or in groups of 2
3) All screenshots and text should be clearly readable

Sources:

- PHAVer: http://www-verimag.imag.fr/~frehse/phaver_web/
- PHAVer Manual: http://www-verimag.imag.fr/~frehse/phaver_web/phaver_lang.pdf
- Reading material: "PHAVer: Algorithmic Verification of Hybrid Systems past HyTech". Int J Softw Tools Technol Transf 10, 263–279 (2008). https://doi.org/10.1007/s10009-007-0062-x (paper is on Moodle)

PHAVer Installation:

Ubuntu OS (or other Linux OS) is strongly recommended in this Lab. PHAVer is a powerful academic tool, but installing PHAVer can be a challenge, so do it early and use Piazza or office hours for help. Once you get PHAVer running, the rest of the lab should be fun.

- The following steps are recommended and known to work:
  1. Use Ubuntu 16.04 32-bit version. You can use Virtualbox for it.
  2. Open terminal and do "`sudo apt-get update`" and "`sudo apt-get install libstdc++5 libstdc++5:i386`".
  3. Download the v0.38 linux executable of PHAVer from the PHAVer webpage where it is listed under the heading "Method 2: Ready-to-Use Executables."
  4. Extract the zipped executable file into a directory.
  5. In terminal, cd to the directory with the PHAVer executable and type "`chmod 777 phaver`"
  6. Now, typing "`./phaver`" should show the usage.
- There are other ways of installing PHAVer. You are free to use these if they work for you. Students have had mixed success with them. For example, there is a compiled 64-bit version of PHAVer on the following website that seems to work on 64-bit Ubuntu: http://depend.cs.uni-sb.de/tools/prohver/

Introduction:

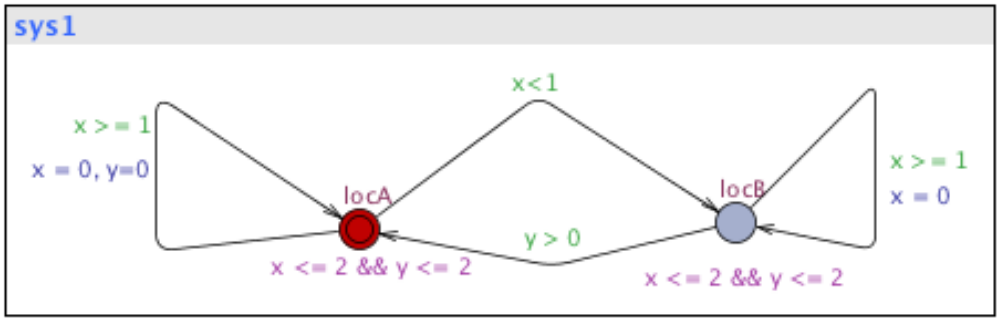
While timed automata from lab 2 are relatively simple to verify, general hybrid systems with continuous and discrete dynamics are more challenging to verify formally. The continuous dynamics are governed by equations in each location. The discrete dynamics are the jumps between locations. A common approach for hybrid verification is to represent continuous dynamics by piecewise over-approximation as shown in lecture. The conservative over-approximate model used by PHAVer is a sound abstraction of the original hybrid system model. The abstract model will reach all parts of the state space that the original model can reach, but due to its over-approximation it may also reach parts additional parts of the state space. This has two important implications:

1. Any part of the state space that is unreachable for the abstract model is also unreachable for the original model. Therefore, **we can use the abstract model to prove that something is unreachable.**
2. Any part of the state space that is reached by the abstract model may or may not be reachable for the original model. Therefore, **we cannot use the abstract model to prove that something is 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 will require huge runtime to prove that a condition is unreachable.

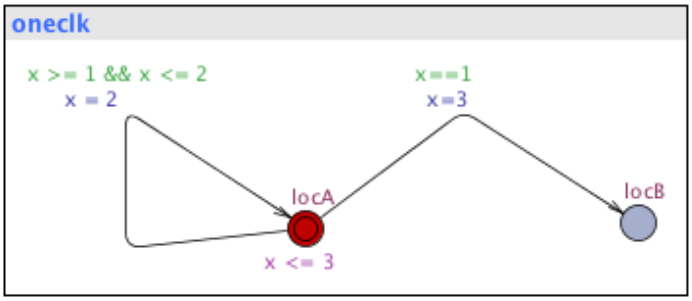
PART A: Getting Familiar [complete and submit Part A as homework #4]

Since timed automata are a restricted case of general hybrid automata, we can start by modeling a timed automaton in PHAVer. The first model you will work with is "`lab3a.pha`" and we've provided its PHAVer model as part of the lab 3 assignment. The model is based on the picture below from UPPAAL of a model you used in homework 3. The model is simple but introduces all of the syntax needed for lab 3. Take some time to understand the PHAVer model and read carefully the comments included in the model that explain all the syntax.



1. Run PHAVer to find the reachable states of the model using the command `phaver lab3a.pha`
    a. How many locations are reached in the search? These locations are partitions of the state space.
    b. What two conditions are checked for reachability in this model?
    Which of the two is found to be reachable? Does that match your understanding of the model?
    c. Plot the reachable parts of the state space. If you install the plotutils package you can do this using the command: `graph -T X -C -B -q0.5 out_example.`
    You are free to use other plotting programs if you prefer.
    d. When the model jumps from location locB back to locB, it updates the value of x to 0 but does not change the value of y. Annotate the plot to show where this jump is occurring.

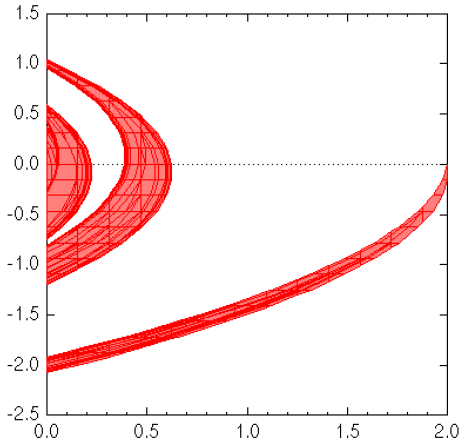
2. Now use your understanding of PHAVer to create a PHAVer model for the figure shown below. This model only has one variable x (which would be a "clock" in UPPAAL). Add another variable "t" to represent global time. Now we can plot "x" against "t" to see how the reachable areas of "x" change with time. "t" should always have a derivative of 1 and should never get reset during jumps. If you don't limit the value of "t" using location invariants, the model will continue reaching new parts of the space as t increases forever. To prevent this, use location invariants to enforce that t is always less than or equal to 5.
    a. Use a reachability query to check whether your system can be in location locA when t is 5 and x is greater than 5. Is this reachable?
    b. Use a reachability query to check whether your system can be in location locB when t is 5 and x is greater than 5. Is this reachable?
    c. Include your model file, including its reachability checks, in your lab submission (it need not be a separate file, you can simply paste the model text into your writeup)
    d. Include a plot of reachable parts of state space (with both x and t as the state variables).
    e. Circle all parts of the plot that are reached only in location locA, and do the same (using a different color or text annotation) for locB. The location information is not provided by PHAVer, but you can figure it out by understanding the model and identifying the jumps in the plot.



PART B: Bouncing Ball [50 pts]

Next you will analyze a bouncing ball model like the one shown in the lecture slides on timed and hybrid systems. You will explore the impact of trying different levels of abstraction, as we've done during lecture when discussing the heating system model. The basic model of the bouncing ball is provided with the lab files. Using this model you will answer the following questions.

1. Use PHAVer to compute reachable states of the bouncing ball model. Run the model using the command `phaver bouncing_ball.pha`.
    a. How many locations are reached? These locations are the partitions of state space that the PHAVer tool uses. Give this number in the report.
    b. Generate the plot of reachable states as you've done in Part A. You don't need to include this plot with your report, but it should look as shown below if you've done everything correctly.



2. Check whether it is possible, using the default settings in the model, to prove that the velocity of the ball is always below positive 1.05. You can do this by showing that the system cannot reach any part of the state space in which the velocity equals or exceeds 1.05. In other words, you are proving the LTL property **G**(v<1.05) by showing that it is impossible to satisfy its negation, **F**(v≥1.05). If the abstract model cannot reach this part of the state space, then we know that the original system also cannot reach this part of the state space, and therefore that its velocity always stays below 1.05. If the abstract model can reach this part of the state space, then we don't know whether velocity can exceed 1.05 in the original system.
    a. Check whether the condition is reachable by visually inspecting the set of reachable states shown in the plot you created with graph.
    b. Have PHAVer check this formally by using the "is_reachable" command as in Part A. Is the text output produced by PHAVer consistent with your visual inspection? Explain this. Include in the report the command that you added to the model for checking this.

3. Now repeat questions 1 and 2 using different levels of abstraction by changing the way that PHAVer partitions the continuous state space. Do this by changing the value of the partition constraint (pc) to each of the following values: {1.0, 0.8, 0.4, 0.3, 0.25, 0.2, 0.15, 0.1, 0.01, 0.001}. For each value of pc, rerun PHAVer to check the property using the `is_reachable()` command from the previous question. Fill up table 1 with your findings and explain the results. If the runtime for any value of pc exceeds 15 minutes, you can stop PHAVer and report the runtime as "timed out."

Partition Constraint PC	Condition reachable?	v < 1.05 proved?	Num Locations reached	CPU Time (sec)
1				
0.8				
…				
0.001				

4. Provide the plot showing all reachable states for pc = 0.25, 0.15, and 0.01. If 0.01 times out, then show the plot for the smallest value of pc that completes within 15 minutes.
    a. Which value of pc results in the simplest analysis?
    b. Which value of pc is the most precise analysis of reachability?
    c. Which value of pc results in the largest amount of unreachable state?
    d. Which value of pc is the best one for checking this property, on this particular model? Justify your answer using the findings from the table.
    e. Describe a part of the state space that is unreachable with pc=0.4. Verify that it is unreachable by using an `is_reachable()` command. Name this model, including the reachability query, as `B4e.pha` and include it with your submission.
    f. For the same state that is unreachable with pc=0.4, one could also check its reachability using other pc values. Without running additional checks, based on the principle of sound abstraction, can you determine whether the same state is reachable for the other values of pc in the table? If so, indicate for each pc value whether the state is definitely reachable, definitely unreachable, or whether it cannot be determined without running the reachability check. Justify your answer.

5. Change the initial condition of the model to have a position that is not fixed but is instead only known to be between 1.5 and 2.5. Plot the reachable states of this model with pc=0.1 and include the plot in your report. If pc=0.1 times out, then use the smallest value of pc that completes.

PART C: Heating System [50 pts]

In this question, you'll be analyzing the heating system model shown below, which is from lecture. First, you'll have to model the system in PHAVer and then explore reachability in its state space. Because the property being checked involves time, your model will need to include a time variable as in part A of this lab.

1. Use PHAVer to compute reachable states of this model with partition coefficient value of 0.1. Indicate how many locations are reached and include the plot of reachable states. The plot should have an axis representing temperature, and an axis representing time; add these labels to the axes. You may want to consider adding an invariant that prevents the value of time from exceeding 10, and/or add location invariants restricting the value of X in the cool and heat locations, so that the system must transition to the opposite location as soon as it is allowed by the guard condition. If you do so, explain in your report why you think.

2. Next, check reachability using different abstractions by varying the value of the partition coefficient (pc). As in the bouncing ball, the idea is to see the effects of over-approximation in abstract models. Check whether your system can reach a condition in which it **is in the heating state, has a time value of 1.3 and has a temperature that is greater than or equal to 20.8**. You must describe this property in PHAVer and use the "`is_reachable`" command as in the bouncing ball example. 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 0.04 of the real answer. In other words, your answer will be considered correct if it shows the condition to be unreachable, but a pc value that is 0.04 larger shows the condition to be reachable. Submit your model, including the reachability query, as `C2.pha`.