ECE 522/622 - Spring 2025
Modeling and Verification of Embedded Systems
Lab #2 Verification of Timed Automata using UPPAAL

Objectives of this lab:

1) To understand symbolic analysis of timed automata.
2) To gain exposure to modeling and verification of real-time systems using UPPAAL.

General lab submission instructions:

1) Lab reports should be typed, with name and ID at the beginning of the Lab report.
2) All screenshots and text should be clearly readable.
3) Your submission **must** include the source files (in .xml format) of your UPPAAL models and queries as indicated in the lab assignment. The .xml file that you save from UPPAAL will contain both the model and queries. You can verify this by reading the file or closing and reopening to check that the queries are saved.
4) You can work alone or in groups of two.
5) If your group contains a student taking the ECE622 version of the course, you must complete step B.5. Otherwise, you can skip B.5 and the points of other questions will be scaled up.

Introduction:

Hybrid systems are systems that combine discrete and continuous dynamics. Timed automata are the simplest non-trivial hybrid systems. Most real-time systems require measuring the passage of time and performing actions at specific times. A device that measures the passage of time, a clock, has particularly simple dynamics: its state progresses linearly in time. In this lab you will use UPPAAL (version 5.0.0, as in problem set 2) to check the properties of a real-time system model.

UPPAAL is an integrated tool environment for modeling, simulation, and verification of real-time systems, developed jointly by Aalborg University in Denmark and Uppsala University in Sweden. It is appropriate for systems that can be modeled as a collection of non-deterministic processes with real-valued clocks, communicating through channels or shared variables. Typical application areas include real-time controllers and communication protocols where timing aspects are critical.

Questions:

Part A *[30 points]*

Download "`bridge.xml`" from the "Examples" section on Canvas and open it in UPPAAL using (`File->Open System`). This file is also included with UPPAAL, so you may find the same file using (`File->Open Example`). The scenario modeled by this file is as follows: Four Vikings need to cross a valley using a damaged bridge in the middle of the night. The bridge can only carry two Vikings at a time and to find their way over the bridge (in either direction) the Vikings need to bring a torch. The Vikings have different speeds, and need 5, 10, 20 and 25 minutes respectively to cross the bridge in each direction. See course slides for additional discussion of the model. Spend time to understand the model, and then use UPPAAL to answer the following questions:

A.1) Does an execution exist in which all four Vikings get across the bridge in less than or equal to 60 minutes? If yes, provide the schedule of bridge crossings, and explain how you used UPPAAL to get the result. Provide a schedule of the crossings as a table in your report. Each row of the table should correspond to one trip across the bridge. The columns of the table should indicate:

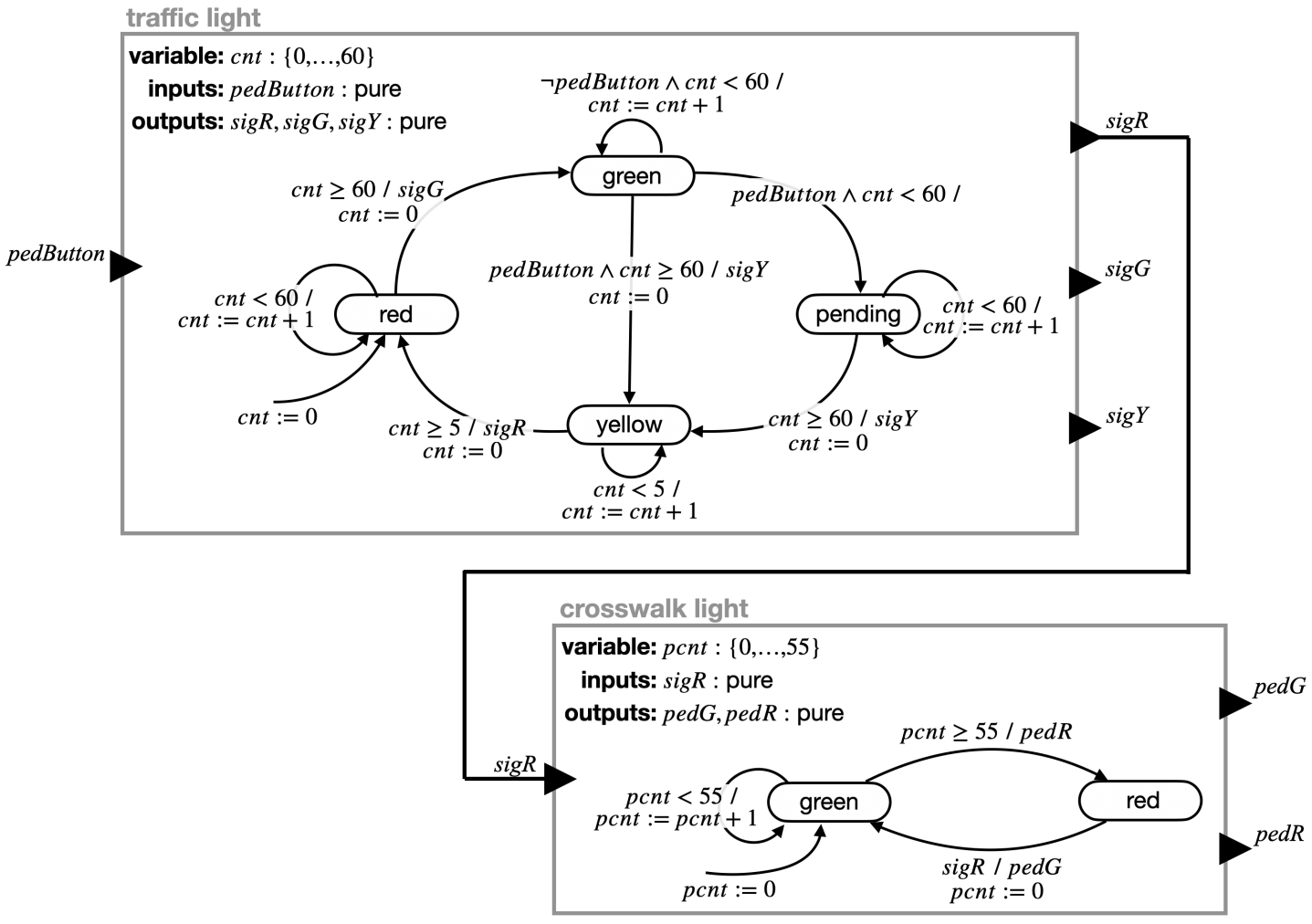
1. Which Viking or Vikings cross the bridge on the trip. You can identify them by their crossing times.
2. The start time of the trip.
3. Whether the trip starts from the safe or unsafe side of the bridge.

A.2) Now add a fifth Viking that needs 40 minutes to cross the bridge in each direction. What is the shortest time for all five Vikings to get across the bridge? Explain how you modified the system model to realize this change and explain what queries you used to find the shortest possible crossing time. Submit your modified model with queries as filename `<lastname>_a2.xml` (for example, my file would be called `holcomb_a2.xml`). As before, include a table to show the schedule of crossings.

A.3) Building on question A.2, now consider a scenario in which each of the five Vikings has the option to reach the safe side by talking a longer path through the valley below, instead of crossing the bridge. Crossing through the valley does not require a torch. It takes each Viking 4 times longer to cross the valley than it does for them to cross the bridge. The bridge can still carry two Vikings, as before. There is no limit to the number of Vikings that can cross through the valley. In this modified scenario, what is the shortest time for all five Vikings to reach the safe side? Include in your report an image showing what modifications you've made to the model to answer this question, explain the modifications in your report, and explain what queries you've used to find the shortest possible crossing time. As before, include a table to show the schedule of crossings, but now your table must have a 4th column indicating whether each crossing is by bridge or valley. Submit your modified model and queries as `<lastname>_a3.xml`

Part B *[70 points]*

Now that you have gotten some practice with UPPAAL, it is time to design and verify your own version of a traffic light controller and crosswalk light controller. A discrete (FSM) version of the system you will implement is shown below. The traffic light model interacts with the crosswalk light model by asserting *sigR* when the traffic light turns red. The discrete system operates on a time trigger and therefore has exactly 1 reaction per second. The *pedButton* input is synchronized to the time trigger. Since the *pedButton* input is unconnected, your analysis of the system must conservatively allow that it could be either true or false at each reaction.



B.1) For properties (b) through (i) in the table below, given as English sentences, explain whether the discrete system will satisfy the property or not. To do this, you should first devise a reachability check that could determine whether the property is true or false for the system; you are free to decide whether reaching the target state would show that the corresponding property is true for the system, or false. Then use your understanding of the model to predict the outcome and give a brief justification of how the target can be reached, or why it cannot be reached. Note that you are solving this problem with intuition and are not using tools to perform a reachability check on the discrete model. For some of the properties, you might have to include extra states or variables to track information needed for checking the property.

(a) It is impossible for the traffic light to be in the pending state 70 seconds after initial condition
(b) It is possible for the traffic and crosswalk lights to be green at the same time.
(c) The traffic light will never stay in the green (non-pending) location longer than 90 seconds
(d) The traffic light will never stay in the pending location for 55 seconds or longer.
(e) The traffic light will never stay in the pending location for 65 seconds or longer.
(f) It is possible that crosswalk light is red when the traffic light is green
(g) Whenever the crosswalk light is red, the traffic light must be green.
(h) The 3rd occurrence of sigG must not happen within the first 300 seconds after initial condition
(i) The 3rd occurrence of sigG must not happen within the first 400 seconds after initial condition

For demonstration purposes, here is an example of a suitable answer for condition (a):

Reachability check: I would add a variable called globalCount that initializes to 0 and increments in every reaction. I would check reachability of "globalCount == 70 and traffic light == pending". If that state is reachable, then property (a) is false.

Expected outcome: The property is false. The state can be reached by pushing pedButton right after the traffic light reaches "green".

B.2) In UPPAAL model the traffic light and crosswalk light system as timed automata. Test your system (e.g. using concrete simulation or simple reachability checks) to gain confidence that it works as intended. You will then test it more formally in the next step.

- Your implementation must use timed automata and therefore will have clock variables to track time and must not have discrete count variables that increment once per second to track elapsed time.
- Remember that enabled jumps in timed automata are not compulsory unless there are also location invariants that force the jump to occur. This is different from the discrete model, which will always make a jump if at least one guard evaluates to true.
- The inputs and outputs of the traffic light and crosswalk light must be connected to communication channels. Every communication channel requires synchronized transmit and receive events (e.g. `take!` and `take?` in the Viking model):
  - Channel *sigR* will have traffic light and crosswalk light as transmitter and receiver.
  - Since the *pedButton* input to the traffic light is unconnected, your model must include an additional component to generate it non-deterministically (to model that *pedButton* could be pushed at any time) and transmit it to the traffic light over the channel. Include in your report a screenshot of the component you created that non-deterministically generates *pedButton*.
  - The unused outputs (*sigG*, *sigY*, *pedG*, *pedR*) will have traffic light or crosswalk light as transmitter, so you will need to add corresponding receivers for these communication channels and ensure they are always ready to receive, so as not to block jumps within the transmitting module.
- Some edges in the discrete model use an input in the guard condition and assert an output when taking the guarded transition. UPPAAL will not allow a single jump to interact with multiple communication channels. You can avoid this problem by adding an urgent location to split a single jump into two jumps with no time elapsing between them.

B.3) Map each condition from the table in B.1 to an appropriate query that can be checked in UPPAAL on your model from B.2, and report the result. Provide in your report a screenshot of the verifier window that shows all queries and whether they are reachable (UPPAAL indicates this by adding red or green circles next to each query once it is checked). Name the model, with all queries included, as `<lastname>_b3.xml` and include this file with the submission so we can check it. Some properties may require modifications in your model; for example, to keep track of intervals between events, or to count the number of times that an event occurs. You are allowed to make these modifications but must use a single model version for all queries since you will submit the model and queries as a single file. Include a query for property (a) even though we've already described above how it can be checked.

B.4) Now add another component to the system that models a pedestrian trying to cross the crosswalk. The pedestrian's input should be *pedG* from the crosswalk light, and the pedestrian's output should be *pedButton* going to the traffic light. The pedestrian model will therefore replace the component you were previously using to generate and transmit *pedButton* and also replace the component you were using to receive *pedG*. The pedestrian model should have an **idle** initial location, a **waiting** location, and a **crossing** location. When transitioning from idle to waiting, the *pedButton* output should get asserted to indicate that a request to cross the street has been made. The pedestrian starts crossing when the *pedG* input occurs. The pedestrian should spend a fixed amount of time in the crossing location, and that time must be a parameter of the model so you can easily change it, and we can easily test it; the name of that parameter must be `pedCrossTime`. Because the system already makes certain assumptions about how quickly a pedestrian crosses the street, the system may be safe or unsafe depending on the value of the `pedCrossTime` parameter.

- Describe the bad condition that can occur if the pedestrian crosses the street too slowly.
- What range of values for `pedCrossTime` allow the bad condition to occur, causing the system to be unsafe? Support your answer by using reachability queries to show that the bad condition can occur for some values of `pedCrossTime` but cannot occur for other values of `pedCrossTime`
- With `pedCrossTime` set to the smallest unsafe value, name the file as `<lastname>_b4.xml` and submit it with the report. Since this is the smallest unsafe value, decreasing `pedCrossTime` by 1 must change the result of the reachability query. We will test this.

B.5) [*ECE622 students*] Now change the model to be more robust, such that it is safe regardless of how long it takes the pedestrian to cross. Explain your change. Using your modified model, repeat your query from the prior step using a value of `pedCrossTime` that was unsafe in the previous step, and show that it is now safe after the change. You are allowed to change the traffic light and/or pedestrian light components in the new model if you find that helpful. Simulate and/or use other queries as well to gain confidence that your system works correctly. With `pedCrossTime` set to be a value that was unsafe in prior step, but is now safe, name the model file and query as `<lastname>_b5.xml` and submit it with the report.

Submit your assignment by uploading to Canvas your typed report, and the files from steps **A.2**, **A.3**, **B.3**, **B.4**, and (for ECE622 students) **B.5**. Each model will include your lastname as part of the filename. Be sure to close and then re-open each file to confirm that the file you are submitting is the file that you intend to submit, and that it includes the appropriate queries. If you work as partners, both student names should be included on the report, but only one student will submit the files for the group.