Mandatory Assignment 2 Option

Baggage Sorting Facility (contd.)

In this assignment, you are supposed to elaborate on your work from Mandatory Assignment 1. Specifically, you should device, verify, implement and possibly test your own sorting algorithm.

1 The Assignment

In this part, you should address the following tasks:

1. Model the physical system (i.e. the LEGO construction) in Uppaal. [Mostly carried over from Assignment 1.]
2. Device a control strategy and model a control program in Uppaal.
3. Formalize and verify the following properties (again):
   - Bags are delivered at the right destination.
   - Every bag is eventually delivered (unless subject to starvation).
   - No collisions take place.
   - No bumping occurs.
   - While a bag is turning (in section c), neither the Feed Belt nor the Distribution Belt is stopped or reversed. [Subject to amendments — see below.]
   - The distribution belt is never stopped or reversed when it carries a bag.
   - Any other properties that must hold for the verification to be valid, e.g. freedom of deadlock and avoidance of erroneous control.

   The usage assumption needed for the properties to hold must be stated and possibly verified.
4. Device some *performance measures* and use Uppal to determine them. Examples could be:
   - The maximum number of bags that can be handled simultaneously.
   - The fastest possible handling of a bag.
   - The maximum handling time of any bag.

   The *throughput* (say bags per minute) for specific scenarios such as:
   - A sequence of bags for Destination A arriving at Check-In 1.
   - A sequence of bags for Destination B arriving at Check-In 1.
   - An alternating sequence of bags for A and B arriving at Check-In 1.
   - Two sequences of bags for the same destination arriving at both Check-Ins.
   - A sequence of bags for Destination A arriving at Check-In 2 and a sequence of bags for Destination B arriving at Check-In 1.
   - Etc.
5. Write a Java program corresponding to the Uppaal model of the control program. The

Java program should be able to compile under the LeJOS environment and should run with the simulator provided. See details on the project page.

6. Test your program on the simulator.
7. [Optional]
   Test your Java/LeJOS program on the real LEGO construction.
8. [Optional]
   Try to apply *statistical model checking* to your system. E.g. in order to extend your coverage or to determine performance distributions. Note that this will require you to transform your system model in order to meet the requirements of Uppaal SMC.

Groups of three persons are expected to make more comprehensive solutions. Either by applying a more advanced sorting strategy, by making a larger number of performance analyses or by applying SMC for analysis. All groups are encouraged to test the solution on the real system.

2 Amendments

In the first assignment, some constraints were made more restrictive than physically necessary in order to simplify the verification. If, however, you wish to optimize the performance of your system, you may use the following relaxations:

- When entering the distribution belt from a feed belt, the feed belt may be stopped as soon as the bag is properly off the feed belt. This will be the case 1.8 seconds after having entered the c) section.
- The c) and d) sections [belonging to the same feed belt] need not totally exclude each other. If a bag on the c) section has turned sufficiently much onto the distribution belt, another bag may safely enter the d) section. This will be the case 1.8 seconds after the first bag has entered section c).

  Likewise, when a bag is passing in front of the feed belt in section d), another bag may safely enter section c) when the first bag has less than 0.3 seconds left of section d).

Since these relaxed constraints are harder to express in the verification, you may choose to stick to the more strict constraints stated in the first assignment.

3 The Stop Problem Revisited

As in the first assignment, the stop problem on the feed belts must be addressed.

It is not required that the system is able to handle more than one bag at a feed belt at a time, so you may stick to a single stop position and handle that as in Assignment 1.

If you really would like to go for multiple bags on the feed belts, two options are available:

- You may use the the *delta time* model, where a bag is stopped a multiple of delta time units. See the project page for details.
- You may use the *stopwatches* available in Uppaal to model stopping. Note, however, that this well render your model undecidable. Therefore you will be limited to make stochastic model checking only and the whole model will have to be transformed in order to meet the requirements for this.

4 Implementation Guidelines

For you Uppaal model to be of use, it is paramount that it reflects the real system, both the physical part as well as the control program running on the RCX brick.

Therefore, your control program must closely correspond to the to the control part(s) of your model. Especially you should consider:

- The interface between the physical model and the control model should correspond to the actual interaction through the sensors and motors. In particular, they should not enforce any constraints on each other.
- The structure of the control model and the Java program should be similar. Eg. each control process is expected to be implemented by a Java thread.
- Timing in the control model must be properly implemented by `sleep`, `wait` or `Timer` operations in the Java threads. Busy waiting will violate the assumption that scheduling needs not be taken into account and is therefore banned.
- Any synchronization among the control processes must be properly implemented by similar synchronization in the Java program. The implicit atomicity of transitions in the control model might need to be explicitly implemented in the Java program. Alternatively, the atomicity may be relaxed in the control model if not needed for proper operation.
- If you use channel communication within your control model, this must be explicitly implemented or rewritten to fit the mechanisms available in Java.
- The limitation of the LeJOS Java environment mentioned on the LeJOS page must be taken into account. Especially, you should be careful to avoid recreation of objects since LeJOS provides no garbage collection.
- The control model may be more abstract than the program. Eg. a simple Java monitor may be represented by a single-location automaton with looping atomic transitions rerlecting the monitor operations.

You may find that you will have to modify, restructure or even rewrite your control model in order to enable a corresponding Java implementation. In that case, your verification work must be redone, of course.

5 Debugging and Testing

If properly implemented, your program should function as expected. However, in practice any manual process is subject to introducing errors. Many such errors are quite mundane (eg.*off-by-one*) and can be found by debugging and testing.

For debugging purposes you may use the *simulator* provided via the project page. The simulator is a very simple Java GUI-program that tries to resemble the behaviour of the physical system as far as possible. The environment provided is the same as in the real LeJOS environment.

Further, your will be able to test your (debugged) control program on the physical Lego construction in May. Instructions for running a program and guidelines for planning the test will be available separately via the project page. Note that even if the program runs in the simulator, it may not work in the physical system if it does not adhere to the LeJOS limitations as described on the *LeJOS page*.

6 Report Requirements

Your work must be documented by a report that covers all the stated requirements to this assignment. The report should take into account the remarks that you have received as feedback on the first part.

Thus, the report for this part should comprise the following:

- A brief introduction including an overview of the changes you have made to your model and the verifications from the first assignment.
- An overview of the major components of your system, their partitioning into a physical part and a control part, and their means of interaction (variables, channels of various kinds). The overview should be illustrated by a figure.
- A (possibly modified) description of the the physical model as in Assignment 1, including your modelling deliberations and statement of assumptions made.
- A description of your overall control strategy idea and your UPPAAL modelling of this.
- A (possibly modified or extended) description of your verification attempts. For each property, the query should be explained. The extent of the class of usage scenarios for which the properties have been verified should be made clear.
- A description of how you have systematically implemented your control model as a Java program. Especially you should argue that issues of timing, synchronization, and atomicity correspond properly.
- A description of your approach to testing the control program on the simulator.
- Possibly, an account of any physical testing on the LEGO construction. See the *test instructions* to appear on the course page.
- Possibly, a description of your attempts at applying Uppaal SMC to your model.
- A discussion of the whole development approach. For instance: What has been gained by the modelling and verification? How well do the verifications you have made cover all scenarios. How trustworthy is the final system?
- A conclusion in which you should summarize your findings of the assignment.

Report Form

The report must be uploaded to DTU Learn as a *pdf*-file.

The report must have a front page identifying the course, the assignment, the group number, and the participating students.

The final report on the assignments should be concise and is expected to be in the range of **10-15 pages**. To this you may add appendices. Printouts of your automata and their corresponding declarations should occur in the report or as appendices. The program code should appear within the appendices.

The UPPAAL model files as well as the Java implementation must be handed in digitally (for details, see the *project page* found via the course home page).

General

The report and all files must be uploaded to DTU Inside no later than **Tuesday May 21, 2024, at 23.59**.

Please note:

- The report must be self-contained, but you are allowed to reuse any relevant parts from Assignment 1. Notice that for Assignment 2, you should not address the given sorting program `SingleSort.java` — only your own.
- You are, of course, expected to address the issues that are pointed at in the feedback for Assignment 1. The report should have an introductory section describing what changes your have made.
- There should be an introductory section declaring the contributions of the participants to the various parts of the work and the report. **Each group member must be made main responsible for a major part of the work.**
- Any collaboration with other groups on smaller parts of the assignments must be declared and clearly identified. Collaboration on major parts is not acceptable.
- Any undeclared use of others' work is *strictly prohibited*

As for Assignment 1, clarifications, FAQs, and practical details will be put on the *project page* found via the link on DTU Learn. You should consult this if you encounter problems.