System validation, 2IW26

Jan Friso Groote (J.F.Groote@tue.nl, MF 7.070, 040-247 5003).
http://www.win.tue.nl/~jfg/educ/2IW26/lente2013/overzicht.html

The purpose of this course is to learn how to specify behaviour of systems and to experience the design of a system where you can prove that the behaviour is correct. So, you will learn how to formally specify requirements and to prove (or disprove) them on the behaviour. With a practical assignment you will experience how to apply the techniques.

The lectures are mainly dedicated to learn the foundations of the specification language mCRL2 and to use it as a manual specification and verification tool. There will be 2x2hours of lectures per week, on Mondays and Thursdays.

In parallel to the lectures there will be an assignment, which must be finished before the examination period at the end of the semester. The goal of the assignment is to apply the techniques and tools to the design of a small distributed and/or embedded system. The purpose is to design this system such that it is proven to comply with all the requirements which must have been formulated in advance.

The marks for the exam and the assignment contribute equally to the final score. The final mark is the average rounded to to a whole number in the ordinary way (7.5 is rounded up to an 8, 7.49 is rounded down). The mark of the resit of the exam will consist for 50% out of the result of the assignment and for the other half of the result of the exam. The result of the assignment is only valid for one year. If the course is redone in a subsequent year, the assignment must be redone.

Literature

The course material consists of

- J.F. Groote and M.R. Mousavi. Modelling and analysis of communicating systems. Lecture notes. 2013. Chapters 1, 2 (not 2.3.2, 2.3.3, 2.4.4), 3, 4, 5 (not 5.6), 6, 7, 9 (not 9.7 and 9.8.3), 10, 11.
- See `www.mcrl2.org` for the tools, manual pages etc.

The lecture notes by Groote and Mousavi can be obtained at the Student Shop. The document can be downloaded from the website (`www.win.tue.nl/~jfg/educ/2IW26/lente2013/overzicht.html`). There will be two versions, one that is equal to the reader, and one that is regularly updated. The exam will cover the indicated parts of the reader as well as everything said during the lectures. Updates of the reader are not part of the exam.

Assignment

The assignment consists of designing a controller for a small distributed and/or embedded system. Below a suggestion for such a system can be found. It deals with the safety controller for a bridge. But it is possible to design any embedded controller or distributed algorithm provided you obtain approval by the supervisor of your assignment. The assignment can be carried out in groups of one to four persons.

Carrying out the assignment consists of executing the following steps:

1. Identify in words global requirements for the whole system. Typical requirements are ‘a bridge will never open when the barriers are not closed’. These requirements are initially to be described in natural language.
2. Identify the interactions that are relevant for your system. Describe clearly but compactly the meaning of each interaction in words.

3. Translate the global requirements in terms of these interactions.

4. Describe a compact architecture of the structure of the system. It is required that the controller has at least three different parallel components.

5. Describe the behaviour of all controllers in the architecture using mCRL2.

6. Verify using the toolset that all requirements given in item 3 above are valid for the design in mCRL2.

The assignment must be documented in a technical report that covers all items above. This report must be a concise technical account of the system and must be written such that from it the requirements, action interface, architecture and behavioural design can be easily understood. It must also be clear how the requirements are verified, in such a way that this can easily be redone exactly without consulting any of the authors of the report. So, for instance the exact commands that are used must be listed, it must be obvious which version of the toolset was used for the verification and on which platform and operating system the verification was done.

Rijkswaterstaat is a Dutch governmental organisation responsible for the main infrastructure in the Netherlands. This comprises the dikes, tunnels, roads and waterways. All these systems contain active components, such as bridges and sluices. Under pressure of safety regulations (a.o., NEN 6787, Het ontwerpen van beweegbare bruggen - Veiligheid, 2003), different ways of working (remote control, bridge keepers with temporary appointments) and surprising experiences with software control, there is a need to have safety control layers in these active components. The safety control layers should be such that dangerous situations should never be able to occur, but it should never be the case that the flow of traffic is unnecessarily hampered.

An example of problematic situations occurred at the Ketelbrug (Ketelbridge), which is a bridge situated in the highway connecting Lelystad with Emmeloord. In a first incident the bridge openened while the barriers were not closed. Subsequently a safety system was built to prevent this from happening ever again. But the safety system happened to kick in when the bridge was closing but still open, as a connection to a sensor measuring whether the barriers were properly closed broke down. The effect was that the bridge stopped and could not be closed. As it took time to locate and repair the broken sensor, traffic was blocked for hours on the highway. The lesson learned was that simple requirements of the kind that the bridge should not move if it is not guaranteed that the barriers are closed, is too simple.

Rijkswaterstaat now wants to understand what an optimal set of software requirements is, guaranteeing absolute safety and optimal throughput. For this we study a simple model of a bridge with two lanes, four barriers, two pairs of stop signs at each side (the pre-signs and the primary signs). The bridge is constructed with a heavy but passive contra weight, such that a relatively small motor can open and close the bridge. The motor is autonomous in the sense that it can be switched on and off, and it can be asked for its status (moving up, moving down, stopped and broken). There are five position sensors, two to indicate whether the bridge is completely open, and three to indicate that the bridge is closed. The sensors do not have to provide consistent information, as they can be broken also. When the bridge is down, there are two interlocking pins that keep the bridge in place. The pins must be instructed to unlock the bridge before the motor is switched on. Cars are not allowed to cross the bridge if both locks are not in place. There is a position sensor on each pin. The primary stop signs and the pre-signs consist of two lamps. There are sensors measuring whether the lights are on. If none of the lights switch on, if both light bulbs are broken, the barriers are not allowed to be lowered. The bridge also has a brake to keep it into a firm position. The brake should always be applied, except when the bridge is supposed to be moving. The status of the brake can either be released, applied or 'out of order'. There is an alarm bel, which should sound directly before the bridge starts moving until the bridge is moving for some time to warn for instance maintenance personel that may be close to moving parts of the bridge.

The control system of a bridge consists of two parts. There is an interface controller operated by the bridge keeper and there is a safety control system that operates the actuators and sensors of the bridge. The interface controller is a complex piece of software assisting the bridge keeper (e.g., it animates opening and closing of the bridge). It is considered to be non-safe, in the sense that it can send erroneous and dangerous commands to the safety system. These commands are typically switch on the traffic signs, start opening the bridge and lower the barriers. We are not interested in this interface controller.

This safety controller is responsible for the safety of bridges and is our primary concern. The task of the safety controller is to block any command that is unsafe and execute any that is considered safe. When an unsafe command is received an error message is sent to the interface controller. This also happens when an hardware problem or other erroneous situation is detected. Although the major task of the safety controller is to avoid any dangerous situation, it should only hamper traffic when it is unavoidable.

In order to obtain an optimal safety controller, a set of requirements about safety and progress should be formulated. Subsequently, a model for the safety controller of the bridge should be made. As stated above, this controller should consist of at least three parallel components. It must be shown that the requirements hold for the model.

Tool set

See `www.mcrl2.org` and the webpage of the course.

Global time schedule

Below a global time schedule is indicated. There are exercises indicated, which upon request can be treated during the lectures. Students are supposed to make these exercises before the lectures, and compare their answers with those of the lecturer. The exercises and short indications of their answers can be found in the reader.

4/7-2-2013. Chapter 1. Chapter 2. Chapter 4. Transition systems, basic processes, process equivalences, conditional operator, time. Elementary reasoning with axioms. The toolset and its philosophy. Exercises 2.2.2, 2.2.3, 2.3.2, 2.3.8, 2.3.9, 2.3.10, 2.4.5, 2.4.6 4.2.2, 4.2.3, 4.3.1, 4.3.2, 4.4.1, 4.5.1, 4.5.2.

18/21-2-2013. Chapter 3. Chapter 4. Section 9.4. Appendix A. Abstract data types. Constructors. Built in data types, bool, quantifiers, pos, nat, int, real. list, set, bag, functions, structured type. Difference between $\approx$ and $=$. Predefined data types, induction. Exercise 3.1.2, 3.1.3, 3.1.4, 3.2.1, 3.2.3, 3.2.4, 3.3.3, 3.4.1, 3.5.1, 3.5.3, 3.5.4.

19/22-2-2013 Start of the weekly sessions discussing the assignments.

25/28-2-2013. Section 4.6. Chapter 5. Section 9.6. Recursion. RSP. Proving recursive specifications equal. Parallel processes and hiding. Expansion law. Communication, multi-actions. Exercises. 4.6.1, 4.7.1, 4.7.2, 4.8.1, 5.1.1, 5.2.1, 5.4.1, 9.6.4, 9.6.5, 9.6.6, 9.6.9.

4/7-3-2013. Chapter 6. The modal μ-calculus with data. Exercise 6.1.2, 6.2.1, 6.3.1, 6.3.2, 6.4.1, 6.4.2, 6.5.1, 6.5.2, 6.5.3.

11/14-3-2013. Chapter 9. Sketch of the lambda calculus. Sum axioms. Sum elimination theorem. Precise proof system. Exercise 9.4.1, 9.4.2, 9.4.3, 9.5.2, 9.5.3, 9.5.4, 9.5.5.

21-3-2013. Note that there will not be a lecture on March 18. This week there will also no discussion sessions for the assignment. Chapter 10. Chapter 11. Linearisation of processes. CL-RSP, CL-RSP with invariants. Exercise 10.1.4, 10.2.9, 10.2.10, 10.2.11.

25/28-3-2013. Confluence and τ-priorisation. Exercise 11.1.3, 11.1.4, 11.2.6.

1/4-4-2013. Reserve.

9-4-2013. Exam (9:00-12:00). Closing date for registration: 31-3-2013.

19-4-2013. Last date to hand in the pre-final report for the assignment. The report must be handed in on paper.

1-5-2013. Last date to hand in the final report for the assignment. The report must be handed in on paper and must be accompanied with the corrected pre-final report.

26-6-2013. Exam (retry, 12:00-17:00). Closing date for registration 16-6-2013.