System Validation 2022
Homework Part 3 – ACSL & Frama-C

Deadline: **23:59 CET, Friday Nov. 4, 2022**

- The assignment can be made in pairs.
- While working together, be very careful with public repositories (if you decide to collaborate via e.g. Git), as we will check for fraud!
- A total of 100 points can be earned.

All solutions should be uploaded via Canvas. You should hand in a single ZIP file with:

- The report in PDF format. Do not forget to include your names and student numbers in the report.
- Note that you are not required to hand in any additional files for the first part of the homework assignment, so your grade for this part is based on the report only. Make sure you clearly indicate in the report what the answers are.
- Your BUFFER.C file. Make sure that we can reproduce your findings with Frama-C, so mention which command you used to verify your program.

Good luck!

Part 1 - Modelling questions (20 pts)

Specifications

Consider the requirements for the lock system, as established during the first lecture. Write formal specifications for the following requirements using the most appropriate formalism (*e.g.* use an appropriate temporal logic, or ACSL).

Exercises

i. Both doors can't be opened at the same time.

ii. Both doors should be closed before the water can be let out.

iii. A door only opens if the water level matches at both sides.

iv. During normal operation, at most one door can be opened at a time.

v. No matter the circumstances, a bout should eventually be able to leave from the other side.

vi. If the water level is being adjusted, both doors should be closed.

vii. It should be possible for the wildlife to bypass the lock.

viii. If a ship is in the lock, and another wants to enter, it should wait for the first one to leave.

ix. The lock can be closed for maintenance at most 1 week per year.

x. The boat only moves when the door is opened.

Part 2 - Static and runtime verification (80 pts)

In this assignment you are asked to verify a circular buffer.

The data structure

A circular buffer can be used to store elements in, and retrieve elements from a fixed-size array `buffer`, such that its maximum capacity `maxlen` can be used at any moment. This is done by using the array `buffer` as if its elements are stored in a ring: if an element is being stored at index `maxlen - 1`, the next element is stored at index `0`.

The integer `head` is used to keep track of the index after the most recently stored element, such that a call to function `push` can store a new element at this index. Function `push` updates `head` after storing a value at index `head` according to the ring structure, i.e. `head` is set to 0 if `head+1` is equal to `maxlen`, otherwise `head` is increased by one.

Likewise, the integer `tail` stores the index of the oldest stored element that has not been read yet, such that elements are read in order. Function `pop` returns the value stored at index `tail` and updates `tail` according to the ring structure.

In case the buffer is full `push` cannot store the provided element `data` and returns -1. Otherwise `push` stores `data` and returns 0. Similarly, if the buffer is empty, `pop` cannot retrieve any element and returns -1. Otherwise `pop` stores the obtained element in `output_arr` and returns 0.

You might notice that there is always one empty space after all the pushed values in the buffer. This is a design choice and allows us to keep the code simple.

A thorough explanation of how this circular buffer works can be found here:
`https://embedjournal.com/implementing-circular-buffer-embedded-c/`.

If you get the following warning during static verification, then you may safely ignore this:

- `Memory model hypotheses for function 'X': /*@ behavior typed: requires separated(...);*/`. This indicates that the program is verified under the assumption that the variables mentioned in the `separated` clause do not overlap in memory.

Exercises

As part of the exercise, you are given a file ‘buffer.c’.

You are encouraged to use runtime verification and/or testing throughout this assignment in order to test your suspicions and get a feel for what the program does.

i. Verify (statically) that the methods `pop` and `push` work as described above. Please mention which command you used to verify your program. **(30 pts)**

ii. Verify (statically) that the loops in the `main` method works as expected and verify that they terminate. **(10 pts)**

iii. Verify (statically) that the buffer is empty, i.e. everything that has been pushed into the buffer has also been read, at the end of the `main` method (before the final return statement). Try to make the verification as general as possible. For example, if you increase the size of the buffer and add more elements, then it should still verify. *Hint: use ghost variable(s).* **(10 pts)**

iv. Write annotations to check whether each element in the `buffer` is the sum of the previous two elements. Your annotations should work for buffers of all sizes greater than 2 (e.g. a buffer with 3 elements, a buffer with 10 elements and a buffer with 100 elements). You may assume that the buffer has length `maxlen`. **(20 pts)**

Check this property using runtime verification. If there are any executions for which the property does not hold, give an example and explain why it does not hold for this execution.

You are allowed to modify the main method to add/remove calls to push/pop to check the property for different executions but ALL calls to push should add the sum of the previous two elements! For example, it is not allowed to simply add a `push(0)`, but you may add an extra `push`($a + b$) after the first loop.

v. Describe how you used runtime verification to check the property mentioned in question iv. Why does your verification effort suffice to thoroughly cover any corner cases? **(10 pts)**

If you are unable to prove one or more of the properties for static verification, consider handing in a solution using runtime verification to still get some points for that part of the exercise. Make sure you describe what you did and how we are able to reproduce your results. If your report does not explain something, we cannot count it towards your grade!

Bonus: Statically verify that no runtime errors occur in the program (main, push and pop method) using the command `frama-c-gui -wp -wp-rte buffer.c`. **(10 pts)**