ETHZ D-INFK Prof. Dr. B. Meyer, Dr. C.A. Furia, Dr. S. Nanz

Software Verification (Autumn 2014) Course Project

Updated 23.10.2014

Hand-out date: 16 October 2014
Team registration: 22 October 2014
Due date: 5 December 2014

1. Summary

The project consists of implementing a basic list data structure and two sorting algorithms that operate on it, then specifying and proving the implementation as much as possible. The first implementation is in Eiffel for AutoProof [4,5]. Then, you will switch to Boogie [6] to have a more expressive language for specification, and thus more control over what you can prove correct. Your activities, experiences, design choices, and evaluations will be documented in a report.

2. Teams

You can work in teams of **up to three persons**. Please email the assistant (chris.poskitt@inf.ethz.ch) **no later than 22 October** to register your team (even if you plan to work on your own).

3. Project description

3.1. Deliverables

We require two deliverables:

- Verified implementations of the required data structure and sorting algorithms in Eiffel (using AutoProof [4,5]) and Boogie [6].
- A report fully describing your design choices, experiences, and evaluations of the tools in the context of your specifications and implementations.

The following two subsections express several requirements and tasks which must be addressed in the implementations, verification activities, and the report.

The report should be written as an *academic report*, i.e. sensibly structured, well written, with your activities fully documented, discussed, and evaluated. Submitting verified implementations alone is not enough to get full credit in this assessment; the report is an important component and the weighting of the marks reflects this (see the following subsections).

3.2. Requirements

We provide a single-class implementation SV_LIST of lists of integers, implemented using arrays in Eiffel. Class SV_LIST includes two publicly visible constant parameters declared statically:

- *Max_count*: INTEGER
  The maximum number of elements stored by the list (a positive number).
- *N*: INTEGER
  An arbitrary positive number, whose use is described below.

The API of SV_LIST includes the following features:

- *empty*: BOOLEAN
  True iff the list is empty.
- *count*: INTEGER
  The number of elements stored in the list.
- *at* (*k*: INTEGER): INTEGER
  The element stored at position *k*.
- *put* (*k*: INTEGER, *val*: INTEGER)
  Store value *val* at position *k*.
- *sequence*: SIMPLE_ARRAY [INTEGER]
  An array of integers, representing the current content of the list.

Note: a skeleton API is provided as a starting point on the course webpage.

You can add other features to SV_LIST's API, which you may also use in the implementation or in the specification.

Class SV_LIST also includes a routine named *sort* which sorts the list content in ascending order, using a combination of two different algorithms as follows:

- If the list stores at least *Max_count*/2 elements and all of them are between -3*N* and +3*N*, then it uses sort uses *Bucket Sort* [1,3] (with 3 buckets) to sort the list .
- Otherwise, it uses *Quick Sort* [2,3] to sort the list.

The implementation of Bucket Sort can recursively call Quick Sort to sort each sublist in a "bucket".

3.3. Tasks

1. **Eiffel implementation (2 points).** Implement the class SV_LIST with the features described above. Discuss any implementation choice that you evaluated at this point.
2. **Eiffel specification (4 points).** Add specification to all features of the class as embedded contracts (pre- and postconditions, class invariants, loop invariants, and possibly intermediate **check** assertions). Describe which aspects of an ideally complete specification you were able to express, and which you couldn't express. Note that while your implementation should always terminate, you do not need to prove termination.
3. **Eiffel verification (6 points).** Using AutoProof, verify as many features of SV_LIST as possible. Describe if there were any aspects of the implementation or of the specification you had to change to make them easier to verify. Describe which parts of the specification you could not verify, and what were the limitations that prevented you from doing it. Discuss whether and how the behaviour of AutoProof is affected by specific choices for the constant parameters Max_count and N.
4. **Boogie implementation (2 points).** Reimplement the functionalities of class SV_LIST in Boogie, as a collection of global variables, functions, and procedures. Instead of Eiffel arrays, you will use maps in Boogie as basic data structure for the implementation. Describe which aspects, if any, of the Eiffel implementation or specification you had to adapt to express them in Boogie.
5. **Boogie specification (3 points).** Improve the specification with some of the aspects you could not express as Eiffel contracts but you can express using Boogie's first-order specification language. Discuss these additions and mention possible other aspects you still could not readily express.
6. **Boogie verification (8 points).** Verify your Boogie program using Boogie. Report any significant problem you encountered, for example which procedures you could verify and which ones you could not. Describe if there were any aspects of the implementation or of the specification you had to change to make them easier to verify. Describe which parts of the specification you could not verify, and what were the limitations that prevented you from doing it.

3.4. Structure and writing

For reports that are well structured, well written, with good spelling and grammar, we will award up to **2 points**.

4. Tools

There are a number of web-based tools that will prove useful throughout the project. Please, however, ensure that you **regularly save your changes offline:**

- **AutoProof via Comcom** (http://cloudstudio.ethz.ch/comcom/#AutoProof): verify your SV_LIST class by pasting it into the "More AutoProof" tab and clicking "Run".
- **Boogie via Rise4Fun** (http://rise4fun.com/boogie): verify your Boogie program by pasting it into the form and clicking the play button.

It is also possible to use these tools offline - see the links on the course webpage.

5. Submission

Please submit a **zip file** containing the source code and the report, in **PDF format**, by email to the assistant by the due date. Your zip file should contain **two folders**: a folder (named 'source') for source code, and a folder (named 'report') for the PDF version of your final report.

6. Support

You can ask questions about the project during the exercise sessions on Wednesday. Outside of these sessions, the assistant is available to meet by appointment (please email chris.poskitt@inf.ethz.ch).

Questions and answers that are relevant to all groups will be anonymised and posted to the following page (please check it regularly):

http://se.inf.ethz.ch/courses/2014b_fall/sv/questions_answers.html

References

[1] Wikipedia: Bucket sort. http://en.wikipedia.org/wiki/Bucket_sort
[2] Wikipedia: Quicksort. http://en.wikipedia.org/wiki/Quicksort
[3] Cormen et al.: *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.
[4] AutoProof tool. http://cloudstudio.ethz.ch/comcom/#AutoProof
[5] AutoProof exercises. http://se.inf.ethz.ch/courses/2014b_fall/sv/exercises/problems2.pdf
[6] Boogie. http://rise4fun.com/boogie and
http://research.microsoft.com/en-us/um/people/leino/papers/krml178.pdf