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

Software Verification (Autumn 2015)
Course Project

Hand-out date: 7th October 2015
Team registration: 14th October 2015
Due date: 27th November 2015

1. Summary

This project consists of verifying some basic algorithms using auto-active verification tools. The first part of the project will consist of verifying the correctness of some given Eiffel routines using AutoProof [4]; the second part of the project will consist of constructing and verifying a particular sorting algorithm in Boogie [5]. 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 14th October** to register your team (even if you plan to work on your own).

3. Project Description

The project consists of three parts: a verification task in AutoProof [4], a verification task in Boogie [5], and a report. Sections 3.1-3.3 describe these tasks in more detail; Section 3.4 describes how marks will be awarded.

Note that extensive documentation for the tools is available online:

- AutoProof: http://se.inf.ethz.ch/research/autoproof/
- Boogie: http://research.microsoft.com/en-us/projects/boogie/

3.1. Task 1 of 3: AutoProof

Your first task is to verify the features in the **SV_AUTOPROOF** Eiffel class, which is available to download from the Software Verification 2015 course webpage. This is to be achieved by adding missing postconditions and loop invariants, then verifying them in AutoProof [4].

Please take note of the following requirements:

- We strongly recommend reading at least the AutoProof tutorial [4] before commencing this verification task.
- The existing implementation of SV_AUTOPROOF **must not** be altered.
- SV_AUTOPROOF already contains a number of assertions; likewise, these **must not** be altered or removed.
- We have explicitly indicated where assertions are missing using Eiffel comments ("--"); specifications should be added to **all** of these places.
- Your specifications should be strong enough to **completely characterise** the functional behaviour of the features (only partial credit is available if features are verified with respect to weaker specifications).

3.2. Task 2 of 3: Boogie

Your second task is to implement, specify, and verify a particular sorting routine in Boogie [5], using the **sv_boogie.bpl** template provided on the Software Verification 2015 course webpage.

The Boogie template models a fixed-length integer array as the map *'a'* from indices 0 through to *N−1*. You should implement, specify, and verify the procedure *'sort'* with the following behaviour:

- if the elements of *'a'* all have values in the range *-3N* to *+3N*, then *'a'* should be sorted using **Bucket Sort (with three buckets)**;
- otherwise, *'a'* should be sorted using **Quick Sort**.

Please take note of the following items:

- Descriptions of Bucket Sort and Quick Sort are available at [1,2,3].
- The implementation of Bucket Sort can recursively call Quick Sort to sort each subarray in the "buckets".
- For full marks, you should specify and verify the **complete behaviour** of the algorithm (i.e. that the resulting array is both sorted *and* a permutation of the original array).
- Unlike "Task: AutoProof", the provided template (sv_boogie.bpl) is just a starting point, and you can alter it if you prefer to model and verify the algorithm in a different way.

3.3. Task 3 of 3: Report

The final task is to provide a written report, fully describing your design choices, experiences, and evaluations of using AutoProof and Boogie in the above two tasks.

Please note the following requirements:

- The report should be written as an **academic report**, i.e. sensibly structured, well written, with your activities fully documented, discussed, and evaluated.
- Particular effort should be made to contrast your experiences doing verification at the program level (AutoProof) and intermediate verification language level (Boogie).
- 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.
- The Marking Scheme (Section 3.4) mentions a number of particular points that your report should cover. You may wish to structure the report similarly.

3.4. Marking Scheme

The list below indicates the weighting of the **25 available points** across the three tasks. Note that points are determined from the combination of your source code *and* your report.

1. **AutoProof verification (10 points).** Add missing specifications to all features of the SV_AUTOPROOF class as embedded contracts (without altering the implementation). Describe how you were able to specify complete specifications, and any problems that occurred. Using AutoProof, verify as many features of SV_AUTOPROOF as possible. Discuss if there were any aspects 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 the limitations were that prevented you from doing it.
2. **Boogie implementation (2 points).** Implement the sorting algorithm described in Section 3.2. Discuss your design choices, and how you modelled the algorithm using the primitives of Boogie.
3. **Boogie specification (4 points).** Specify the complete behaviour of the sorting algorithm using the specification primitives of Boogie. Discuss your specification choices, in particular, how you modelled the “permutation” property for the resulting array. Describe any difficulties and how you overcame them. Contrast the specification language of Boogie with the specification language of AutoProof.
4. **Boogie verification (7 points).** Verify your Boogie program using Boogie. Report any significant problems 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 the limitations were that prevented you from doing it. Explain how you achieved modular verification.

5. **Structure and writing (2 points).** Two points are awarded for reports that are well structured, well written, with good spelling and grammar.

4. Web-Based 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_AUTOPROOF 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 instructions on the tool webpages [4,5].

5. Submission

Please submit a **zip file** containing your source code and 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/2015b_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: http://se.inf.ethz.ch/research/autoproof/
[5] Boogie: http://research.microsoft.com/en-us/projects/boogie/