FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Review report of a final thesis

Reviewer:	Ing. Konrad Siek, Ph.D.
Student:	Bc. Rudolf Rovňák
Thesis title:	SOM Virtual Machine Implementation
Branch / specialization:	System Programming
Created on:	27 May 2021

Evaluation criteria

1. Fulfillment of the assignment

[1] assignment fulfilled
▸ **[2] assignment fulfilled with minor objections**
[3] assignment fulfilled with major objections
[4] assignment not fulfilled

The tasks specified in the assignment are for the student to:
1. familiarize himself with SOM,
2. design and implement SOM in C++,
3. emphasize clarity of design over performance, and
4. compare overhead vs existing SOM++ VMs.

Ch. 2 of the thesis shows familiarity with SOM. Ch. 3 and the attached artifact prove that the student did in face design and implment a VM in C++. Ch. 4 compares performance of his implementation and other SOM implementations. My contention is that the comparison with existing implementations is done shallowly.

2. Main written part

80 / 100 (B)

The structure of the thesis is straightforward and effective: an introduction, a chapter describing SOM in terms of language, runtime features, and related work, a chapter describing the students implementation, an evaluation chapter, and conclusions.

For the most part the written part of the thesis was good. I think the text was laid out well and the concepts contained in the thesis were communicated clearly and efficiently. I especially liked the student's use of verbatim quotes to introduce various parts of SOM. If not for the problems described below, I would have found the thesis to be excellent.

The thesis has three major problems. One is the lack of a comprehensive examination of the existing work in the field of SOM implementations. The students spends about a paragraph on it without providing much detail or differentiation. I specifically think the description should examine the difference between the student's C++ SOM implementation and SOM++, the existing C++ SOM implementation. I believe that performing this sort of analysis is a key component differentiating the skill set of a bachelor and a master.

The other problem with the thesis is the evaluation section. It is bare and lacks both impactful analysis and sufficient discussion. The thesis only performs 3 microbenchmarks. Dismisses using bigger benchmark as obviously not worthwhile, but this was not obvious to me. There is no significant discussion of performance. What discussion there is, is highly speculative, but the source code of the other implementations is available, and the systems are relatively straightforward (by design) so the student should be able to reason firmly about the source of the difference. The evaluation results are only shown averaged over an aggregate of iterations, with no distributions. The naming scheme in the evaluation chapter is very confusing. It seems that the student's implementation is never formally named, so one has to figure out which of the C++ implementations it is on the graphs (CppSOM, SOMpp, or SOM++).

The third (and least) problem is the description of the bytecode used in the implementation. It was not clear where the bytecode used for the student's implementation originated. The text suggest that vaguely that it is standard for SOM implementations, but a comparison with other SOM implementations shows that they are different (no dups, no supersend, no jumps) and some of the terminology used in the bytecode reminds me more of Feeny. If I were to venture a guess I would say the bytecode was borrowed from other SOM implementations and tailored to the needs of the student's implementation, which is fine (even interesting), but should be explained. The bytecode description itself lacks description of specific elements, e.g. parameters in SEND i n, GET LOCAL i.

Other minor criticisms: I would also have liked more examples of code when syntax of SOM was introduced to complement the throughout description in prose. The description of the GC needs a corresponding description of the allocator. There are also some negligible language mistakes and some minor typos in examples.

Citations and outside concepts and work are all properly differentiated from the body of the work. Citations are done correctly as quotes or as references to bibliography. The student does not provide citations for other implementations described in text and used in the evaluation (but the work of other implementations is at no point claimed to be the student's own work, just the reference items are missing). No ethical infraction was committed.

I found no factual errors, etc. The work contains no formal notation (this is in line with the implementation nature of the work).

I have identified ANTLR as a dependency for the students work. It is used according to its license.

Directive 26/2017 was superseded by directive 38/2019. The relevant article there is Art. 3. WRT Art 3. pt. 1. the required parts are included in the

thesis. the thesis is within the prescribed length (at 87 pp). WRT Art. 3 pt. 2. all required formalities are applied accordingly. WRT Art 3. pt. 3 the student uses the prescribed LaTeX template. Art 3. pt. 4-7 are not relevant to the written part.

3. Non-written part, attachments

80 / 100 (B)

My inspection of the source code is hampered by my relative lack of experience with C++. The design is relatively clear. The AST interpreter is implemented with a visitor. The BC interpreter is implemented using a big switch. These are classic designs for these mechanisms, so they add to clarity of design. Inspecting the evaluation of each opcode makes it relatively clear what each opcode does to the state of the machine and GC executions are clearly marked.

The VM internals (heap, frame stack, scope stack, operand stack) are implemented straightforwardly, usually using vectors as their backbone. As a minor point, I would have preferred to see a more low level implementation that controlled its own memory layout, which would make memory fragmentation a visible issue with the mark-and-sweep GC. Currently references to the heap are implemented by shared pointers. With clarity in mind, I would have preferred to see references implemented as simple indices into the heap. In that context I also encounter references to shared pointer and unique pointers, whose purpose I do not understand.

The documentation for building and running the system is incomplete. It contains a few small omissions that make it very annoying to work with for a first time user (download ANTLR, but which version; run cmake, but what argument; no mention that make must follow cmake). Command line interfaces should have `--help` too.

The error messages from the interpreter are obtuse. For example: "Unresolved class name: Object " means "No method `f` in object `1`" and "terminate called after throwing an instance of 'std::out_of_range'" probably means "No operator `+` in object of class String."

4. Evaluation of results, publication outputs and awards 80 /100 (B)

I believe from the description of the task and from the goals of SOM implementations in general, the goal of the thesis is educational in nature. The clarity of design is meant to make it easy for engineering students to pick up and work on. The thesis fulfills this in some aspects. For the most part, the source code is readable and clear, and the design is straightforward. I think the code requires a documentation effort to be completely usable: either in situ or in an external documentation. A few of the curious design and implementation decisions might require reversing (e.g. use of casts and references to smart pointers replaceable by visitors and more a straightforward heap indexing scheme). I think for educational purposes, the heap should be more straightforward: an area of preallocated memory rather than a vector, to provide the verisimilitude of memory management. The implementation itself is badly documented and hostile to new users. It requires a revamp of error messages. Nevertheless the results can be deployed in practice with some various adjustments.

The overall evaluation 80 /100 (B)

I believe the assignment was fulfilled. The execution leaves something to be desired, however. The thesis is well written, but is missing some significant features. The evaluation and overview of similar existing works is performed superficially. The presented software does implement SOM and works. The design is mostly clean and the code readable, but contains some mysterious choices.

Questions for the defense

1. What are some of the reasons for implementing SOM in C++ if there already is a C++ implementation (in the student's opinion, apart from it being the in the task spec)?

2. What is the allocator for student's SOM implementation?

3. What was the student's process of designing the bytecode (or adapting it from existing bytecode)?

4. What are the key differences between the student's implementation of SOM and the existing C++ implementation?

5. What conclusions can be drawn from the performance comparison between this and other implementations?

6. What is the purpose behind functions returning references to shared pointers and unique pointers?

Instructions

Fulfillment of the assignment

Assess whether the submitted FT defines the objectives sufficiently and in line with the assignment; whether the objectives are formulated correctly and fulfilled sufficiently. In the comment, specify the points of the assignment that have not been met, assess the severity, impact, and, if appropriate, also the cause of the deficiencies. If the assignment differs substantially from the standards for the FT or if the student has developed the FT beyond the assignment, describe the way it got reflected on the quality of the assignment's fulfilment and the way it affected your final evaluation.

Main written part

Evaluate whether the extent of the FT is adequate to its content and scope: are all the parts of the FT contentful and necessary? Next, consider whether the submitted FT is actually correct – are there factual errors or inaccuracies?

Evaluate the logical structure of the FT, the thematic flow between chapters and whether the text is comprehensible to the reader. Assess whether the formal notations in the FT are used correctly. Assess the typographic and language aspects of the FT, follow the Dean's Directive No. 26/2017, Art. 3.

Evaluate whether the relevant sources are properly used, quoted and cited. Verify that all quotes are properly distinguished from the results achieved in the FT, thus, that the citation ethics has not been violated and that the citations are complete and in accordance with citation practices and standards. Finally, evaluate whether the software and other copyrighted works have been used in accordance with their license terms.

Non-written part, attachments

Depending on the nature of the FT, comment on the non-written part of the thesis. For example: SW work – the overall quality of the program. Is the technology used (from the development to deployment) suitable and adequate? HW – functional sample. Evaluate the technology and tools used. Research and experimental work – repeatability of the experiment.

Evaluation of results, publication outputs and awards

Depending on the nature of the thesis, estimate whether the thesis results could be deployed in practice; alternatively, evaluate whether the results of the FT extend the already published/known results or whether they bring in completely new findings.

The overall evaluation

Summarize which of the aspects of the FT affected your grading process the most. The overall grade does not need to be an arithmetic mean (or other value) calculated from the evaluation in the previous criteria. Generally, a well-fulfilled assignment is assessed by grade A.