FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Review report of a final thesis

Reviewer:	Ing. Tomáš Pecka
Student:	Maksim Gusev
Thesis title:	TinyGo Language
Branch / specialization:	Computer Science 2021
Created on:	9 June 2024

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

I think the assignment was fulfilled with minor objections. Writing both a backend and a frontend of a compiler for (even simplified) programming language is a difficult job. However, the backend implementation and used algorithms seem very trivial.

2. Main written part 60/100 (D)

The thesis focuses on the creation and implementation of the TinyGo language and its compiler. TinyGo is a simplified version of the Go language. The thesis explores the components of the compiler and code generation techniques to some extent. Then it shows TinyGo features and compares them to features of C/C++.
The thesis summarizes the important principles needed to create a simple working compiler.

However, the thesis contains multiple typographical and language errors, even in chapter titles. Several sentences are incomplete, and some references are improperly compiled (e.g., "??") or likely refer to incorrect figures. Patterns like "(Section)" and "Figure later" suggest the thesis was not thoroughly proofread after compilation. Additionally, it would benefit from using a spellchecker. Despite these issues, the overall typographic and language quality is satisfactory.

There are some factual errors and inaccuracies. It seems to me that the text sometimes uses C and C++ interchangeably because some code samples are labelled as C, but it is clear that it must be C++. Sec. 2.1.1 states that compiler cannot fail in the lexing phase, but it can upon reaching something that it is unable to lex. Figure 2.2b is wrong, the leftmost leaf node should be labelled "Int". Figure 2.3 states that backend consist of a parser and type checker, which is unfortunately wrong.

Several parts of the thesis are confusing. To name a few: Page 11 mentions a "control flow graph" without defining it. Listings 2.2-2.4 show LLVM IR without prior introduction to (LLVM) IRs, potentially causing confusion. The statement "GCC (C Compiler)" on the last line of page 15 is misleading, as GCC is also a C++ compiler and supports other languages. The red frame around the "Input code" label in Figure 3.1 is unexplained. Section 3.3 only mentions a POSIX library for concurrency. Listing 3.8a uses different method name in declaration and definition. Section 4.2 starts with "The 2 chapters (...)," which is ambiguous given the preceding three chapters.

The thesis could benefit from more examples, such as in section 2.1.4.1, which enumerates types of IR without providing examples. Listing 4.3 would be more informative if it included the original Go program.

Although creating a full compiler is a challenging task, the absence of having at least a more-than-trivial backend is unfortunate. The compiler solves that problem by requiring the tiny86 VM to have basically unlimited number of registers. The backend currently intentionally does not perform neither any register allocation/spilling. The number of required registers can be really high (tens of registers) even for simple programs that loads a pair of numbers and then performs 4 simple arithmetic operations on the pair.

3. Non-written part, attachments 69/100 (D)

The compiler seems to work for the attached tests and a few simple programs I wrote. However, the tiny86 VM is not included, making it impossible to run the compiled programs directly. The README does not specify where to obtain the VM or provide any guidance on this matter.

I found a test folder with several test inputs, but there are no automated tests. I would expect at least dozens, if not hundreds, of test files and some form of automated testing (perhaps using ctest). I would also expect a CI pipeline running these tests.

The directory structure of the compiler, divided into three parts, is appropriate. However, the "Sources" and "Headers" directories do not follow best practices. The same goes for the choice of makefile generator tool. While using CMake is a good decision, its settings does not fully adhere to best practices for a C++ project.

The code itself can be difficult to understand at times. Several sections lack comments, do not follow C++ best practices, and resemble plain C. I would expect a little bit higher code quality for an educational program.
When I compiled the project with the address sanitizer (using g++ 14.1), the program failed due to unimplemented virtual destructors and some other undefined behaviors. The attachments also include unnecessary build and object files from the author's development machine.

4. Evaluation of results, publication outputs and awards 60/100 (D)

I am not sure if the resulting compiler can be effectively used for educational purposes. The created language resembles TinyC with slightly different syntax, which does not seem to offer any significant new concepts to the Tiny* universe. Implementing at least a limited form of inheritance would help TinyGo stand out. Additionally, the compiler code itself needs improvement to be suitable for educational use.

The overall evaluation 65 /100 (D)

The topic of the thesis was very challenging for an undergraduate. The written part has some deficiencies as well as the compiler written. Despite all of this, I recommend the thesis for defense and grade it with a D.

Questions for the defense

In section 4.2.3.2 you describe that your compiler intentionally does not care about register allocation and spilling because the Tiny86 VM supports an unlimited number of registers. Consequently, your programs use a very large number of registers. How would you improve your compiler if the number of registers was limited? What would be your next steps?

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. 52/2021, 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.