JIT Compiler Generation Through Meta-interpretation

December 5, 2022

Supervisor: Guillermo Polito

Emails: guillermo.polito@univ-lille.fr

Keywords: Compilers, Interpreters, Optimization, Code Analysis, Intermediate Languages

1 Context

JIT (Just-in-Time) compilers are an optimization technique often used for interpreted languages and virtual machines. They allow to spend time optimizing only frequently used code, while falling back in slower execution engines for non-frequent code. For example, the Pharo and the Java VM run on a bytecode interpreter and eventually compile machine code for methods that are frequently called.

Nowadays, the Pharo Virtual Machine is implemented in a subset of the Pharo language called Slang. The Virtual Machine developers then benefit from the high-level tools used to work with Pharo code, such as the code editors, testing frameworks and debuggers. In a later stage, the Virtual Machine code written in Slang is transpiled to C and then compiled to the target architectures.

The current Pharo JIT compiler that is part of the Virtual Machine, aka Cogit, implements an architecture based on templates of native code per bytecode. When a method is compiled, each bytecode is mapped to its corresponding template. All templates are concatenated to form a single machine code method. This architecture has as drawback that the behavior of the Pharo language is duplicated in both the bytecode interpreter and their corresponding machine code templates.

The topic of this internship is to explore the automatic generatation of machine code templates from the bytecode interpreter using an abstract interpreter on the existing bytecode interpreter. Such approach could benefit from having a single implementation of the interpreter and help in keeping both implementations synchronized.

2 Project and objectives

The main objective of this project is to generate ahead-of-time the code of a compiler using an existing interpreter. A meta-interpreter (i.e., an interpreter over an interpreter) will interpret the bytecode interpreter's code and generate code that knows how to compile at runtime. To avoid the overhead of meta-interpretation during the execution, such generation happens ahead-of-time (i.e., before the program execution). The objectives of the project are:

- Construct an abstract meta-intepreter to analise the existing bytecode interpreter
- Generate from such analysis an intermediate representation
- Create tools to test and debug such intermediate representation and the native code generated from it

The student will learn in this internship the following skills:

- How a template-based JIT works
- Abstract interpreters
- Compiler intermediate representations
- Low level memory optimizations such as tagged values
- The interations between generated machine code and runtime code (trampolines)
- How to apply standard software engineering practices (testing, automation, continuous integration) to low level domains

References:

- Practical partial evaluation for high-performance dynamic language runtimes https://dl.acm.org/doi/10.1145/3062341.3062381
- Structure and Interpretation of Computer Programs http://web.mit.edu/alexmv/6.037/sicp.pdf
- Fun with Interpreters https://github.com/SquareBracketAssociates/Booklet-FunWithInterpreters/releases/download/continuous/fun-with-interpreters-wip.pdf