Functional Programming Language Compiler on GraalVM Platform

Palacký University Olomouc

Author: Mgr. Ivo Horák **Supervisor: Mgr. Petr Krajča, Ph.D.** **Dept. of Computer Science, Palacký University Olomouc**

Motivation

One of the simplest approaches to language implementation is interpretation. Although interpreters are favoured for their simplicity and portability, the massive investment required to build advanced compilers, as evidenced by the efforts in JavaScript performance enhancement, is often too resource-intensive. Unfortunately, such investments are not often justified, especially for research projects or domain-specific languages (DSLs). However, modern meta-compilation techniques, such as tracing and partial evaluation, offer a compelling alternative. Platforms like RPython and Truffle utilize these techniques to allow basic interpreters to achieve the performance levels of top-tier virtual machines (VMs). While RPython [1, 2] uses trace-based JIT compilation, Truffle [3], which is a core component of the GraalVM platform, relies on partial evaluation for JIT compilation guidance. This thesis primarily focuses on the utilization of Truffle within the GraalVM ecosystem.

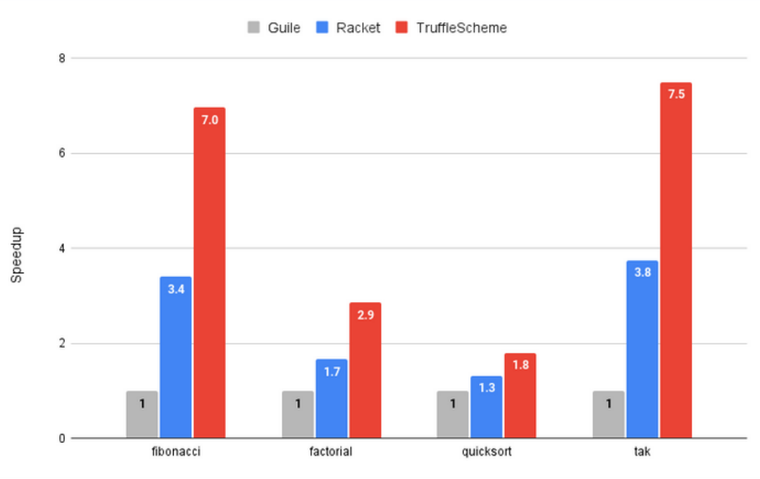
Goals

- Implement an interpreter for a subset of the Scheme specification on the GraalVM platform. The interpreter should:
  - Implement functional elements that have never been implemented on the GraalVM platform before.
  - Support polyglot programming.
- Compare the performance with the existing Scheme implementations.
- Describe and explain the theoretical concepts underpinning the GraalVM and the associated Truffle framework. Given the limited resources available online, this text aims to first introduce the theory behind the GraalVM platform and, secondly, to provide a comprehensive guide for creating one's own language implementation on the GraalVM platform.

Results

- TruffleScheme, a Scheme language interpreter supporting a subset of the Scheme specifications, has been created.
- TruffleScheme is on average 4.7 times faster than the Guile implementation and 1.8 times faster than the Racket implementation.
- TruffleScheme is the first publicly available implementation on the GraalVM platform that supports both Tail Call and Tail Recursive optimizations without compromising interpreter speed, especially on the JVM which does not inherently support TCO. Notably, the implementation of the Tail Recursive optimization significantly boosts the interpreter's speed by removing the recursive call and the overhead associated with it.
- TruffleScheme enables interoperability with languages on the GraalVM platform, facilitating polyglot programming. A new generic API was designed and implemented, allowing tasks like evaluating foreign language code or reading bindings from their global environments. To enhance the developer experience, numerous primitive procedures and special forms have been added to the language.
- A comprehensive document has been created, organized into two main sections. The first section explains theoretical concepts such as meta-compilation or the First Futurama projection. The second section provides a practical guide, filled with examples, on how to implement basic language elements such as global variables, local variables, and user-defined procedures

Benchmarks



Interoperability example

```
(eval-source "python" "def fibonacci(n):
                           if n in {0, 1}:
                               return n
                           return fibonacci(n - 1) + fibonacci(n - 2)")
(define python-fib (read-global-scope "python" "fibonacci"))
(python-fib 35)
```

References

1. C. F. Bolz and L. Tratt. The Impact of Meta-Tracing on VM Design and Implementation. Science of Computer Programming, 2013.
2. C. F. Bolz, A. Cuni, M. Fijalkowski, and A. Rigo. Tracing the Meta-level: PyPy's Tracing JIT Compiler. In Proc. of ICOOOLPS, pages 18–25. ACM, 2009.
3. T. Würthinger, A. Wöß, L. Stadler, G. Duboscq, D. Simon, and C. Wimmer. Self-Optimizing AST Interpreters. In Proc. of DLS, pages 73–82, 2012.