FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

A Rust-like Typed Assembly Language

Using affine types to avoid garbage collection

Mathias Svensson
`freaken@freaken.dk`

Supervisor: Ken Friis Larsen
`kflarsen@di.ku.dk`

Co-supervisor: Andrzej Filinski
`andrzej@di.ku.dk`

July 31, 2015

1 Project Description

Typed Assembly Language (TAL) is a form of proof-carrying code that potentially allows the distribution and safe executing of binaries from untrusted third-parties without the safety-features of an MMU. It has multiple potential benefits, such as:

- The typical proof-size is small compared other schemes, allowing for a small file-size overhead.
- A system built in TAL allows for a very small trusted computing base, as only the proof-checker and a small standard library is needed.
- It in principle allows for relatively efficient code; many compiler-optimizations or hand-optimized functions should be typable.
- It makes it easier to reason about the correctness of compiler-optimizations.

However it also has a number of disadvantages, some of which it shares other schemes:

- It is not possible to directly translate many programming languages into TAL. This includes e.g. the language C, which can only be embedded in TAL by either:
  1. Compiling C into intermediate code which is then interpreted by TAL,
  2. Using static code analysis,
  3. Using annotations made by the programmer or
  4. A combination of 2 and 3.
- It makes it harder to write a compiler, especially compiler-optimizations.
- Most variants of TAL depend on garbage collection.

Rust is a recently developed programming language, trying to solve a problem similar to TAL, i.e. avoiding invalid memory accesses. It does this by employing an affine type-system, which together with "borrowing", which allows for many common programming patterns without compromising safety or performance to any significant degree.

The main objective of this thesis is to investigate the possibility of using a Rust-like type system in the context of TAL. Such a combination should allow for the development of TAL without garbage collection, that still permits many useful programs. In particular it should be possible to compile code from a simplified version of Rust into this variant of TAL.

Core tasks

It is expected that all of these tasks are completed during the project:

- Design a TAL on top of a small toy assembler-language. This TAL should use a type-system inspired by the affine type system in Rust, preferable without the need for any garbage collection.
- Implement a type-checker and a way to test the code (such as an interpreter, assembler or native execution harness).
- Prove useful properties of the designed TAL such as soundness.
- Analyze how this TAL could be used to implement a useful sandboxing framework or an operating system.

Additional tasks

It is expected that at least some of these tasks are completed during the project:

- Design a small standard-library to allow programs written in the designed TAL to interact with the world in a meaningful manner.
- Extend the TAL to a real-world assembly language such as MIPS or possibly x86.
- Build a prototype compiler, which is able to translate into the designed TAL.
- Actually implement (a small part of) the above mentioned sandbox or operating system.

Learning objectives

1. Analyze and identify problems related to memory corruption.
2. Understand and explain the theory of affine/linear types.
3. Understand and explain the theory of TAL.
4. Explain how the theory from learning objective (2) or (3) can help both help avoid the issues presented in (1).
5. Explain how the theory from learning objectives (2) and (3) can be combined.
6. Evaluate the designed TAL with respect to usability, restrictions and overhead in terms of size, compilation time, verification time and runtime.
7. In the context of the evaluation, compare the designed TAL with other solutions to the same problem, such as e.g. other TALs, other proof-carrying code schemes or the JVM.