Title: Smart Machine Code Disassembler

Keywords: {comma separated value keywords}

Laboratoire, institution et université: INRIA Lille Nord Europe

Location: Lille, France

Team: Equipe RMoD – INRIA Lille Nord Europe

Internship supervisor:

Guillermo Polito guillermo.polito@univ-lille.fr

https://guillep.github.io

Context

Compilers are complex pieces of software that generate machine code from a programming language source code, and having to deal with many constraints. For example, compilers need to balance compilation time with quality/efficiency of the generated code. Such constraints have an impact on the compiler architecture, its algorithms and optimizations.

In this scenario, machine code, or object code, is as the final step encoded in binary instructions that are not human readable. Disassemblers read the binary instructions and generate a sequence of human readable *assembly* instructions. However, disassembled code lost most of the information from the original source code.

Because of these complexities, debugging compilers and the code they generate is no easy task. Generated machine code can be the result of several intermediate transformations, and tracking the source of a given problem can be a challenging task.

Objectives

The objective of this internship is to produce a smart machine code disassembler for the Cogit JIT compiler. The smart disassembler should perform the following tasks:

- provide higher-level information on top of a normal disassembler: what is this register used for? what does this offset mean? What is this jump or call referring to?

- traceability from the generated code to the original code. The disassembler should be able to answer the following questions: which step in the compiler produced this instruction? What part in the original source code does this represent?

References

The Cog Smalltalk Virtual Machine writing a JIT in a high-level dynamic language.
Miranda et al. VMIL '11

A low Overhead Per Object Write Barrier for the Cog VM.
Bera et al. IWST '16

A partial read barrier for efficient support of live object-oriented programming.
Bera et al. ISMM '15

Two decades of smalltalk VM development: live VM development through simulation tools.
Miranda et al. VMIL '18