A LLVM Back-end for MLton

by

Rahul Dhawan
rdhawan201455@gmail.com

Report submitted to
IIT Palakkad

in partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY
in Computer Science

SUPERVISED BY
Dr. Piyush P. Kurur

Department of CSE
Ahalia Integrated Campus,
Palakkad Dist., Kozhippara, Kerala 678557, INDIA

April 2019

Abstract

An LLVM Back-end for MLton

Rahul Dhawan

Supervising Professor: Dr. Piyush P. Kurur

This report presents the design and implementation of a new LLVM back-end for the ML-ton Standard ML compiler and converting of LLVM Ast to LLVM IR. The motivation of this project is to utilize the features that an LLVM back-end can provide to a compiler, and compare its implementation to the existing back-ends that ML-ton has for C and native assembly (x86 and amd64). The LLVM back-end was found to offer a greatly simpler implementation compared to the existing back-ends, along with comparable compile times and performance of generated executable, with the LLVM-compiled version performing the best on many of the benchmarks. The LLVM IR can used for optimization of source code and JIT(Just In Time).

Contents

Chapter 1

Introduction

Compilers are some of the most important tools in the software ecosystem, being the tool that turns code written in high-level programming languages to executables that can run directly on hardware. They are also complex software systems, having to face the challenges of transforming high-level language constructs and abstractions to efficient lower-level representations, and handle the portability aspects of supporting compilation to multiple platforms.

Compilers are generally architected in a three-phase design: The front-end, optimizer and back-end, shown in figure 1.1. The front-end is responsible for the lexing, parsing, and type checking of the source code, transforming it into an abstract syntax tree (AST), which acts as an intermediate representation (IR) in the compiler. The optimizer improves the efficiency and performance of this intermediate representation by transforming the code to simpler yet semantically equivalent versions, possibly using different representations if it is useful to do so. In the back-end, the code is emitted to an executable form, usually as either machine code that can be directly run on hardware, or byte code that can be run on a virtual machine.

Because of the split between the different components of a compiler, it is possible for multiple compilers for different languages to share the same back-end system. One of the long-standing

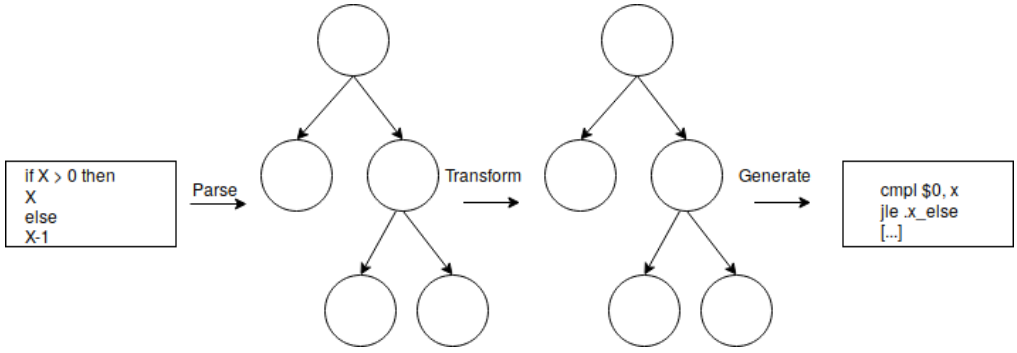

Figure 1.1: General compiler pipeline

issues in compiler design is how to best solve the challenges in utilizing a common back-end technology. Compiler developers want to ensure the compiled programs perform the best that they can, but they also want to leverage language-agnostic tools and libraries that free them from being concerned with the low level details required to make the compiler generate high-performance executables. One such project that solves this issue effectively is LLVM, which defines a high-level, target independent assembly language that can be aggressively optimized and compiled to several different architectures

This report examines an LLVM back-end for MLton and conversion of LLVM AST into the LLVM IR source code. MLton5 is written primarily in SML with a runtime system written in C, and is able to self-host. MLton's features include support for a large variety of platforms and architectures, the ability to handle large, resource intensive programs, and aggressive compiler optimizations that lead to efficient programs with fast running times. The MLton compiler currently has three back-ends: C, x86, and amd64. The C back-end emits the compiled program as C code, and uses an external C compiler to compile to native code. The x86 and amd64 back- ends, known together as the native back-ends, emit assembly language directly which is then assembled by the system assembler into native code. The native back-ends offer better performance and compile times, but have

a limited set of supported platforms.

In this report, we will be looking at the design and implementation of the new back- end for the MLton Standard ML compiler, using the LLVM IR as a target and Design and implementation of an algorithm to convert the LLVM ast into LLVM IR.

The rest of the report is organized as follows. Chapter 2 goes over the design of MLton and the techniques it uses to compile Standard ML programs. It also goes over the design of LLVM, and how it can be used as a compiler back-end. Chapter 3 describes the design and implementation of the LLVM back-end, going over the design choices made and the strategy used for translating to LLVM. Chapter 4 goes over the design and implementation of the algorithm to convert the LLVM AST into LLVM IR. Chapter 5 gives concluding remarks for this project and ideas for further work

Chapter 2

Background