`mc`: A Compiler for the “Mini” Language

CPE431 Final Report / Jacob Hladky / June 3, 2015

Mc is a compiler for the “mini” toy language, targeting the amd64 architecture on OS X and GNU/Linux. Mc is written entirely in SML, and depends on one external SML module: “SML-JSON”, which provides callbacks that assist in parsing. Mc produces either an ELF or a MachO object file that is assembled and linked by the GNU assembler and linker.

The program “mc” is in fact a bash script which handles command-line argument parsing and which calls the SML “compiler” binary along with any other required programs. The “compiler” binary is itself compiled with mlton.

Architecture

Mc is input with a source file containing a JSON-representaion of the AST for a mini program, and parses this JSON into an internal AST with the `json2Ast` function. Then it scans through the AST to collect various information about the program. This information is collected into the symbol table using the `mkSymbolTable` function.

The AST and symbol table are then passed into the static checker. The `staticCheck` function runs a series of tests on the program to make sure that it does not violate the semantics of the source language. All type checking is done in this step. The static checker can fail; upon failure the checker prints an error message and compilation stops. After the static checker finishes compilation is guaranteed.

The AST is then converted to the ILOC intermediary assembly using the `ast2Iloc` function, which also required the symbol table. The ILOC is then run through several optimizing steps, detailed below. After being optimized the ILOC is converted to amd64 assembly using virtual registers with the `iloc2Amd64` function. Register allocation is then performed by the `regAlloc` function.

This completes the compilation process. The resulting amd64 assembly is outputted using the `programToStr` function. The resulting object file is then assembled and linked.

Internal Structures

AST

Symbol Table

Control Flow Graph and Interference Graph

ILOC and Amd64

Optimizations

Code Performance