COMP 520 - Final Report (GoLite)

Alexandre St-Louis Fortier (260720127)
Stefan Knudsen (260678259)
Cheuk Chuen Siow (260660584)

April 15, 2016

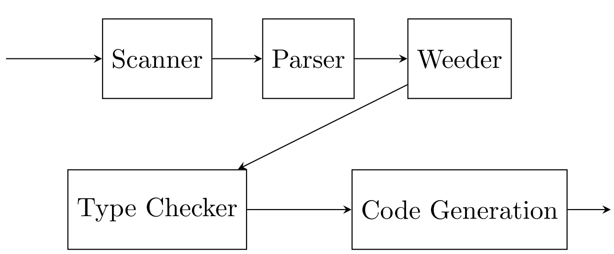
1 Introduction

New programming languages are being conceived every other year to tackle various kinds of problems. The structure of a computer program are dependent on the syntax and semantics of a given programming language, and the correctness of the program has to be exact for a compiler to translate it to a machine language. Since the number of programming languages is on the rise, the study on compiler design is imperative to actually understand the tools and how the whole compilation process works. Our group decided to work on the GoLite project as the source-to-source compiler.

We chose OCaml [1] as our implementation language for compiler construction because the recursive nature of the language allows for easy manipulation of abstract syntax tree (AST). In addition, the pattern matching feature that OCaml provides is particularly helpful in constructing the compiler and has been applied to most parts of the compiler. The tools that we used for building the scanner and parser are `ocamllex` and `Menhir` [2,3,4]. The maturity of the parsing and scanning tools were another reason for the choice of OCaml. What's more, the lead TA had pointed out that OCaml is a good language for such a project.

As for the target language, we have chosen to generate WebAssembly (or `wasm`) [5], a low-level programming language that attempts to be more efficient than JavaScript for the web browser. In particular, our compiler generates WebAssembly TextFormat (`.wast`) code which uses an AST representation in S-expression syntax [5,6]. Since WebAssembly is currently in the experimental stage, only a handful of documentations about the language exist and even some of them are not complete. So far only 2 compilers exist that translates a particular language to WebAssembly: emscripten [8] from LLVM to JavaScript (`asm.js`, the predecessor of `wasm`), and ilwasm [9] from C# to `wasm`. As such, our compiler that takes GoLite programs and generates WebAssembly TextFormat is a timely project.

2 Compiler Structure



2.1 Scanner

2.2 Parser

2.3 Weeder

2.4 Symbol Table

2.5 Typechecker

2.6 Code Generation

2.7 Webassembly primer

2.7.1 Declaring locals

3 Examples

4 Conclusions

5 Contributions

Scanner Alexandre laid the ground work for defining the tokens. Cheuk Chuen and Alexandre defined the regular expressions for the literals.

Parser Cheuk Chuen laid the ground work for defining the grammar and AST. Alexandre made important changes and thorough bug checks.

Pretty printer Stefan worked extensively on the pretty printer.

Weeder Alexandre wrote the weeder.

AST Alexandre modified the AST so as to add annotations.

Symbol table Cheuk Chuen wrote the symbol table. Alexandre made changes according to the modified type checker.

Type checker Stefan laid the ground work for the type checker. Alexandre made changes according to the modified AST.

Pretty printer for type Stefan modified the pretty printer to work for the new AST and to print the types of typed expressions.

Testing Alexandre wrote shell scripts to automate testing of the existing valid and invalid programs. Cheuk Chuen wrote the majority of the tests for type checking.

Code generation All three of us contribute to various parts of the code generation.

6 References

1. `https://ocaml.org/`
2. `http://caml.inria.fr/pub/docs/manual-ocaml-4.00/manual026.html`

3. https://realworldocaml.org/v1/en/html/parsing-with-ocamllex-and-menhir.html
4. http://pauillac.inria.fr/~fpottier/menhir/manual.pdf
5. https://webassembly.github.io/
6. https://github.com/WebAssembly/spec/blob/master/ml-proto/README.md
7. https://github.com/WebAssembly/design/blob/master/AstSemantics.md
8. https://github.com/kripken/emscripten
9. https://github.com/WebAssembly/ilwasm