A Compiler for the CAMLE language

January 8, 2015

Contents

1 Introduction

1.1 Language choice

I had initially started the coursework using `Java` and `ANTLRv4`. Unfortunately `ANTLRv4` deprecated the use of abstract syntax trees in favour of node listeners that run a procedure when a certain node is created[1], this was a choice by the author *Terence Parr* to make the use of `ANTLR` easier for most users (who aren't building compilers). I then decided I'd switch back to `ANTLRv3` as was suggested. I implemented the lexer and parser, and spent a while trying to create an IR tree from the AST in `Java`, this was a deeply unpleasant experience. After around 10 hours of battling with the `Java` code I decided I would instead switch to a language with pattern matching. I had

[1]`https://theantlrguy.atlassian.net/wiki/display/~admin/2012/12/08/Tree+rewriting+in+ANTLR+v4`

read good things about `Haskell` for compiler writing, and I was keen to get more experience with some of the more advanced concepts in the language such as `applicative functors` and `monads`.

2 Parser Design

There are a variety of libraries for parser construction in `Haskell`, `alex` and `happy` are a pair of libraries often used in conjunction for lexical analysis and parser generatation, however since I had already gone down the route of parser generation in `Java` I decided that instead I would try a different style of parsing utilising *parser combinators*, `parsec` is one of the more well known libraries to implement this idiom, and seemed to have reasonable documentation so I chose this.

"In functional programming, a parser combinator is a higher-order function that accepts several parsers as input and returns a new parser as its output"[2].

The `parsec` library is so well written, the source code acts as a beautiful example reference.

3 Intermediate Representation Design

3.1 Stack machine based design

At one stage I had considered generating instructions for a stack machine from the AST, however I couldn't figure out how to allocate registers for the variables that were stored via pushing to the stack.

3.2 Three address code

4 Backend Design

5 Conclusion

[2]Parser Combinators - `http://en.wikipedia.org/wiki/Parser_combinators`