FP in OCaml Report

A Short and Sweet Dive into Interpreters with OCaml

Jul 7th 2019

Written by:

Khinshan Khan
Using emacs org-mode and latex, as well as embed languages.

Note: Table of Contents is hyperlinked to pages within the document, and mentioned of within context, "here" should hyperlinked to external links. Some PDF viewers may not support this, but will render the text fine. Most should support it, as even opening this in a browser like Google Chrome works.

Enjoy.

Contents

1 Introduction

This project is a Lisp interpreter implemented in OCaml to evaluate basic Lisp code. This Lisp resembles ELisp the most closely, as I'm more familiar with it. Although this means I'll forgo rational types, a way to represent any rational number without precision errors. However, this is not completely ELisp, as it never was meant to be. This project more so explored how to utilize OCaml to evaluate Lisp code. It exposed me to ocamllex, menhir, sexp, and a deeper appreciation for pattern matching.

2 Parsing and Lexing

The most critical part of the project is probably the lexer and parser. Lisp is a nice language to use as it consists of S-expressions (symbolic expressions, I will use the shorthand sexp). I decided to take a very list oriented approach. I shall gloss over various things such as prog in `parser.mly`. The important sexp I defined was either an atom or cons. A cons would simply be a list of sexp, delimited by surrounding `()`. An atom is much more complex, it's a single "unit" of either an int, float, bool, string, char, or a sexp list. To be exact, the type is

```
type t =
  | Atom of [ `Int of int | `Float of float | `Bool of bool | `String of string | `Char of char
| Cons of t list
```

where units are delimited by at least one space (extra non important spaces are ignored).