Rhine

Overview of Rhine

- Lisp style language inspired by Clojure
- S-Expressions
- Built on top of LLVM
- Dynamic typing
- Functional features
- Automatic type conversion
- First class functions
- External C bindings
- Types

Example: Fibonacci

```
(defn fib [n]
   (if (= n 0) 0
     (if (= n 1) 1
       (+ (fib (- n 1)) (fib (- n 2))))))
```

Example: Map!

```
(defn map
  [f coll]
  (if (not (= [] coll))
    (cons (f (first coll))
          (map f (rest coll)))
    []))
```

Implementation

- OCaml LLVM bindings
- LLVM arrays/vectors are fixed length, not used
- Arrays can contain any type
- Variable length arrays supported

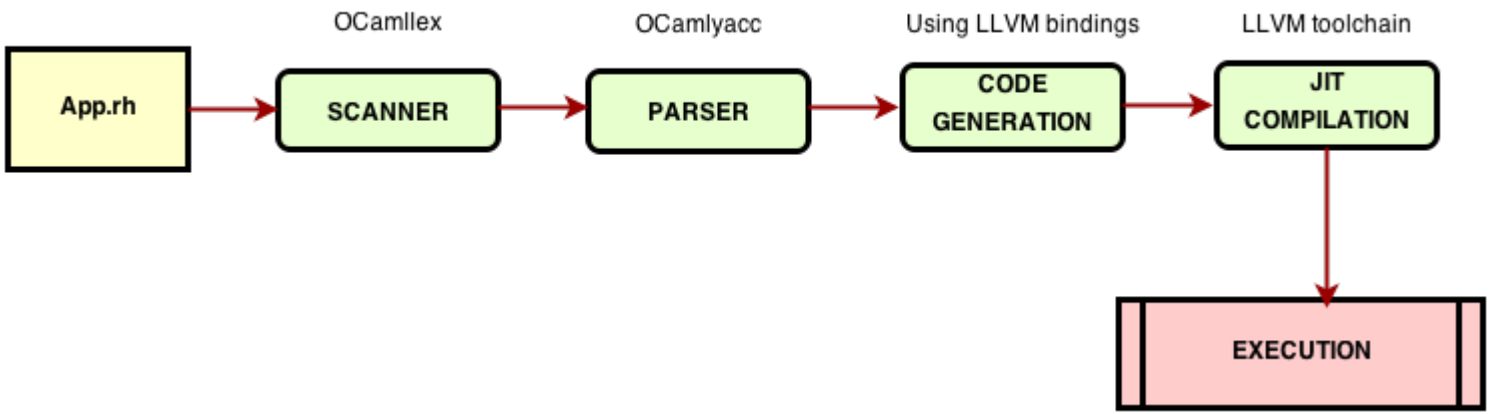
Implementation

- value_t is the structure behind dynamic typing
- Contains Integers, Bools, Strings, Arrays, Array length, Doubles
- Function pointers are stored in value_t
- Type conversion

Implementation

- **defn** generates LLVM functions directly
- Nested and recursive **let** is supported
- **def** uses global constant + initializer function
- Top level statements are generated as functions with zero arguments and are always run, like main

Pipeline


OCamllex
OCamlyacc
Using LLVM bindings
LLVM toolchain
App.rh
SCANNER
PARSER
CODE GENERATION
JIT COMPILATION
EXECUTION

Summary and lessons learned

- Summer class is really short
- LLVM is really hard
- Features of Lisp look nice abstractly but are difficult to implement

Future additions

- Garbage collection, value_t is malloc'd but never free'd currently
- Variable number of arguments for functions being passed around
- Functions that support varargs
- Connection to text editor

Demo!