THC

THE Haskell Compiler

Terrible, Horrible, Evil Haskell Compiler

By: Hans Shi, Alexandru Marcu

Overview

A haskell-like language

Pure functions. Function does nothing but give output, and is dependent only on input.

Most functions can’t do IO or have states.

Predictable, well defined and safe program behaviour

Will not kill your dog and summon Yog-Sothoth, keeper of the gate

Program structure

Algebraic datatype definitions

Every datatype is a labelled union of structs

```
-- data definition
data List a = Node a (List a) | Empty;
```

Function definitions

Body is composition of function definitions and cases

```
-- function definition
map :: (a -> b) -> List a -> List b;
map f list = case list of {
    Empty -> Empty;
    Node h t -> Node (f h) (map f t);
};
```

Frontend

Very simplified subset of haskell

Type definitions are required

Semicolons and braces for easier parsing

Support for 10 levels of left, right and non-associative infix operators, although no way to define infix operator level right now.

Generated AST is very optimizable via inlining functions and constant folding.

Type checking

Currently very limited, user must define all types as there is no type inference.

Recursively evaluates the type taken and produced by function applications and cases to ensure type of function matches signature.

If types are validated at compile time, one can guarantee that they will be correct at runtime, allowing us to throw away all type information and do no type checking at runtime. (Type erasure)

IR/Backend

Continuation passing style. Functions never return, but call the code which should run next.

CPS allows for various optimizations, and aids in garbage collection.

CPS representation is compiled into C functions

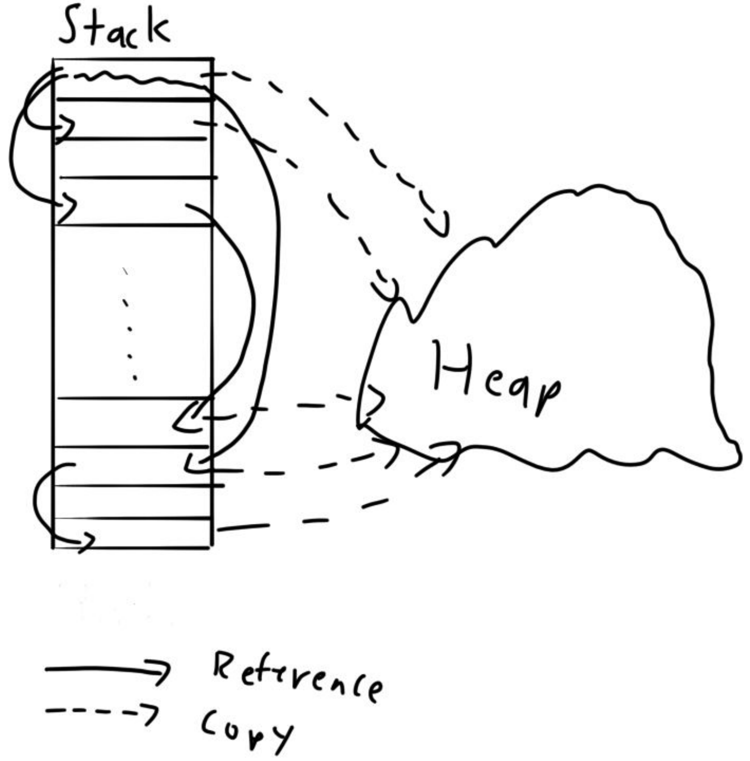
Runtime/GC

When stack gets too large, GC must run.

Only thing we need is top frame, since functions don't return, due to CPS.

Trace all references on the stack and copy to heap.

GC heap using similar tracing if heap gets too big.



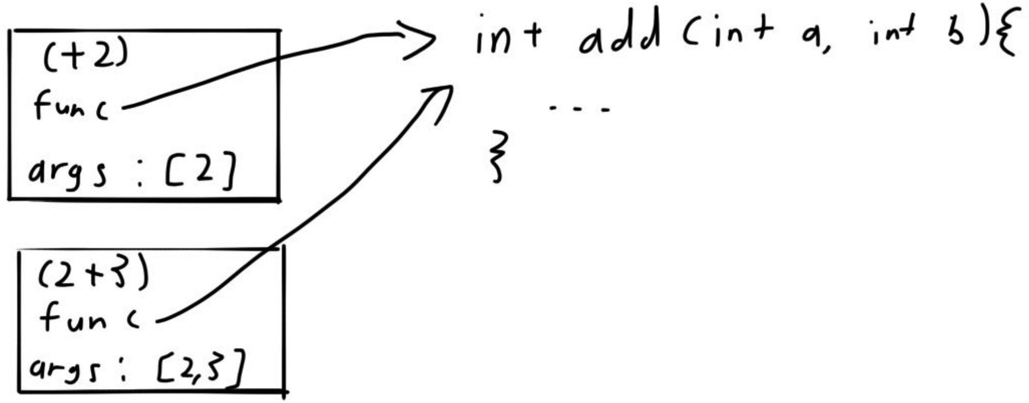
Currying and Lazy evaluation

Partially applied functions are functions

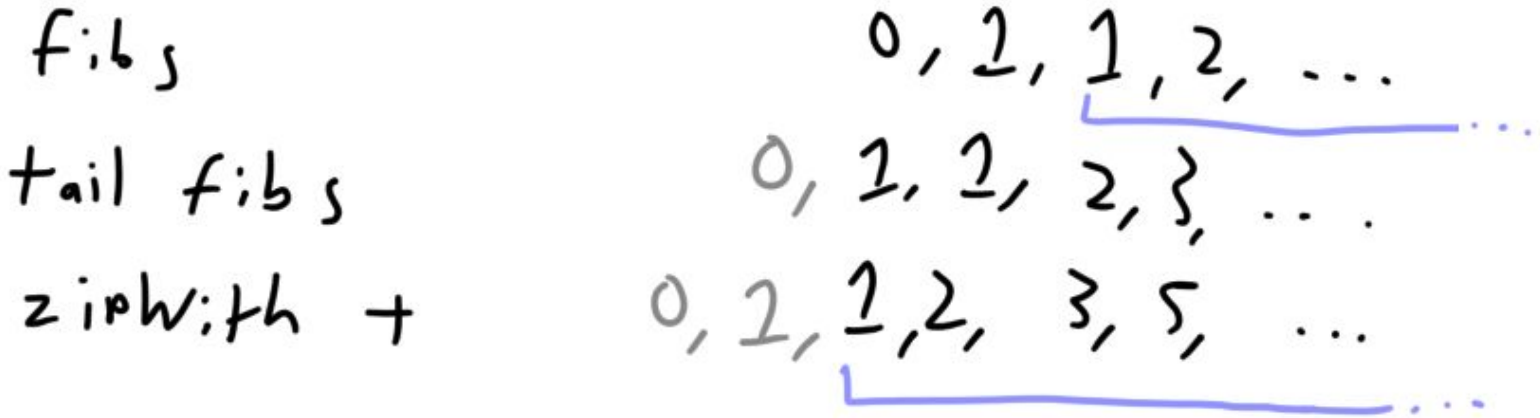
Nothing is evaluated until it needs to

Function objects store function pointer and argument list.

Argument list may have same size as function arity, but function is not called until needed.



Demo



Moving forward

Type inference, no more need to specify every single type

Typeclasses, lets us have monads, which lets us have proper IO

Self bootstrapping compiler. Implement THC in THC.

Lots of internal hacks and inefficiencies to fix!

WHY??!??!??!???

The design of the language prevents you from making terrible hacks.

Very easy to debug, and reason about. Also very easy for the compiler to make optimizations, as there are many consistency assumptions that can be made

The ideas in haskell are being implemented in other languages like rust, which has very functional error handling, and C++, which is getting traits, a system very similar to typeclasses.

JS and Python have functional constructs, like map and reduce.

JS callbacks are just continuation passing style! (except you have to write it by hand)