Hoki

We are lying to you :)

What is Hoki?

>\<> A pseudo stack-based, postfix ordered, repl based, functional programming language.

>\<> In other words, what if a language was really silly?

>\<> Wahoo!

(Fl)operations!

```
> 1 2 + //Addition
3
> 2 1 - //Subtraction
¯1
> 1 2 × //Multiplication
3
> 2 6 ÷ //Division
3
```

Co(d)mparisons!

```
> 1 1 = //Equality
true
> 1 2 = //Equality
false
> 1 2 > //Greater than
true
> 1 2 < //Less than
false
```

(F)Lis(h)ts!

```
> [2, 3, 4] [1] >< //Concat operator
[1, 2, 3, 4]
>
> 1 <> //Encapsulate operator
[1]
>
> [2, 3, 4] 1 <>< //Cons operator
[1, 2, 3, 4]
>
```

(F)Lis(h)ts Cont!

```
>
>
> [1, 2, 3, 4] <>]< //Head operator
1
>
> [1, 2, 3, 4] <>[< //Tail operator
[2, 3, 4]
>
>
```

(F)Lis(h)ts Cont!

```
>
> 0 [1, 2, 3, 4] + <<      //Fold
> 0 (1 +(2 +(3 +(4 +)))) //Equiv
10
>
>
> [1, 2, 3, 4] (2+) >> //Map
[3, 4, 5, 6]
>
```

Functions

```
> Integral Integral add -> Integral
> a b add <- a b +
> Integral Integral mult -> Integral
> 0 ~ mult <- 0 //case evaluation
> ~ 0 mult <- 0
> a b mult <- b a 1 - b mult +
> // recursive definitions
> 2 3 mult
6
```

Lambda Expressions

```
>
> 1 2 3 (c b a ~ <- c b a + -)
0
>
> // Length using fold
> 0 [1, 2, 3, 4] (~ x ~ <- x 1 +) <<
4
>
```

Implementation Split

Frontend Language

- 🐠 For the user
- 🐠 Feature-rich syntax
- 🐠 Reduces/compiles down to core language

Core Language

- λ For the computer
- λ Type system and evaluation happen here
- λ Relatively small language (easier to work with)
- λ Minimal tooling/syntax for testing

Core Language

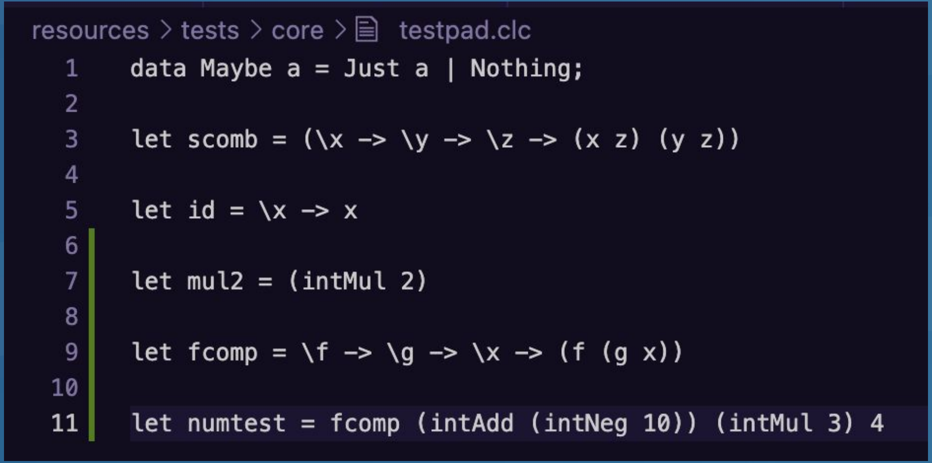
Simply Typed Lambda Calc extended with:

- λ Basic Primitives
- λ Data Constructors
- λ Pattern Matching
- λ Let Polymorphism
- λ Typeclass sorta things if time permits

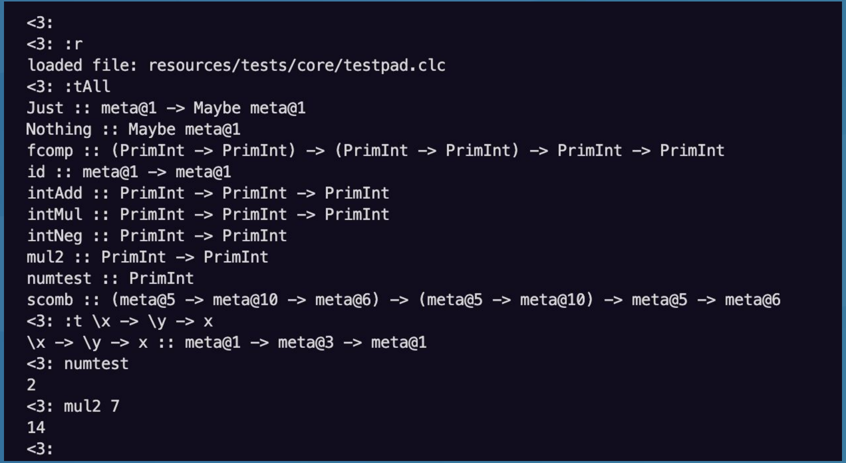
Bidirectional Type System

- λ InferType & CheckType operations
- λ Lends itself well to subtyping (won't be doing that here though)
- λ Based on *Practical type inference for arbitrary-rank types* paper & its structure

Core Lang Progress/Demo


resources > tests > core > testpad.clc

```
data Maybe a = Just a | Nothing;

let scomb = (\x -> \y -> \z -> (x z) (y z))

let id = \x -> x

let mul2 = (intMul 2)

let fcomp = \f -> \g -> \x -> (f (g x))

let numtest = fcomp (intAdd (intNeg 10)) (intMul 3) 4
```


```
<3:
<3: :r
loaded file: resources/tests/core/testpad.clc
<3: :tAll
Just :: meta@1 -> Maybe meta@1
Nothing :: Maybe meta@1
fcomp :: (PrimInt -> PrimInt) -> (PrimInt -> PrimInt) -> PrimInt -> PrimInt
id :: meta@1 -> meta@1
intAdd :: PrimInt -> PrimInt -> PrimInt
intMul :: PrimInt -> PrimInt -> PrimInt
intNeg :: PrimInt -> PrimInt
mul2 :: PrimInt -> PrimInt
numtest :: PrimInt
scomb :: (meta@5 -> meta@10 -> meta@6) -> (meta@5 -> meta@10) -> meta@5 -> meta@6
<3: :t \x -> \y -> x
\x -> \y -> x :: meta@1 -> meta@3 -> meta@1
<3: numtest
2
<3: mul2 7
14
<3:
```

Project Structure

- >\<> Broken into two namespaces
    - >\<> CoreLang contains the backend
        - >\<> CoreLang.CoreRepl, CoreLang.CoreLoader, CoreLang.CoreTyping...
    - >\<> Hoki contains the frontend
        - >\<> Hoki.FrontParse, Hoki.FrontSorts, Hoki.Macros, Hoki.Translate...
- >\<> Uses Parsec and Haskeline

Github

Hoki Github

More Hoki

Viewer Beware

Some *Advanced* Hoki‡

‡ Advanced hoki is only available in some regions with time permitting
purchase not necessary

Advanced Hoki

```
> 0.5 ∡ //arccos of 0.5
0.5235…
> Decimal sin -> Decimal
> a sin <- 2 π ÷ a - ∠
> π sin
0.0
> 2 5 ⁿ //power function
25   //5 to the 2nd power
```

Wide Hoki

```
>
> Decimal atan -> Decimal
> x atan <- 2 π ÷ 0.5 1 2 x ⁿ + ⁿ x ÷ ≦ +
> 1 atan
0.7853…
> 1 atan 4 π ÷ =
true
>
```

Typed Hoki

```
Data_Cons       |
type Data_Cons  |
   TypeVar List ^
//Alignment is a style guide, not
enforced.
```

Typed Hoki Examples

```
False   |
True    |
MyBool ^
//Boolean
Definition
```

```
               Empty |
(a List) a Cons |
              a List ^
//List Definition
```

Type Deconstructor

```
> a a  MyBool if -> a
> ~ t  (True) if <- t
> f ~ (False) if <- f
> 0 1  True if
1
> 0 1 False if
0
```

Thank You!