Frege

purely functional programming
on the JVM

GOTO Berlin 2015
Dreaming of code
Why do we care?

\[
\begin{align*}
\text{time}_1 & : & a &= 1 & & b &= 2 & & c &= b \\
\text{time}_2 & : & b &= a \\
\text{time}_3 & : & a &= c \\
\end{align*}
\]
Operational Reasoning

\[
\begin{align*}
a &= 1 & 1 \\
b &= 2 & 1 \quad 2 \\
c &= b & 1 \quad 2 \quad 2 \\
time_1 \\
b &= a & 1 \quad 1 \quad 2 \\
a &= c & 2 \quad 1 \quad 2 \\
time_2 \\
\text{We need a debugger!}
\end{align*}
\]
Using functions

\[ a = 1 \]
\[ b = 2 \]
Using functions

\[
\begin{align*}
  a &= 1 \\
  b &= 2 \\
\end{align*}
\]

\[
\text{swap}(a,b) = (b,a)
\]
Let’s just program without assignments or statements!
Developer Discipline

Pure Functional Language
Online REPL
try.frege-lang.org
Define a Function

\[ \text{Define a Function} \]

\[
\text{frege}\> \text{times } a \ b = a \ * \ b
\]

\[
\text{frege}\> \text{times } 2 \ 3
\]

6

\[
\text{frege}\> :\text{type times}
\]

\[
\text{Num } \alpha \Rightarrow \alpha \to \alpha \to \alpha
\]
Define a Function

frege> times a b = a * b

frege> (times 2)3

6

frege> :type times

Num α => α -> (α -> α)

no types declared

function appl.
left associative

no comma

typeclass constraint
only 1 parameter!
return type is a function!

thumb: „two params of same numeric type returning that type“
Reference a Function

frege> twotimes = times 2
frege> twotimes 3
6
frege> :t twotimes
Int -> Int
Reference a Function

```plaintext
frege> twotimes x = times 2 x
frege> twotimes 3
6
frege> :t twotimes
Int -> Int
```

- No second arg!
- „Currying“, „schönfinkeling“, or „partial function application“.
  Concept invented by Gottlob Frege.

Inferred types are more specific
Function Composition

frege> twotimes (threetimes 2)

12

frege> sixtimes = twotimes . threetimes

frege> sixtimes 2

frege> :t sixtimes

Int -> Int
Function Composition

\[
(f \circ g)(x) = f(g(x))
\]

\[
\begin{align*}
\text{frege}\> & \quad \text{twotimes \ (threetimes \ 2)} \\
& \quad \text{12} \\
\text{frege}\> & \quad \text{sixtimes = twotimes . threetimes} \\
\text{frege}\> & \quad \text{sixtimes \ 2} \\
\text{frege}\> & \quad :t \ \text{sixtimes} \\
\text{Int} & \rightarrow \text{Int}
\end{align*}
\]
Pure Functions

Java

```java
T foo(Pair<T,U> p) {...}
```

Frege

```frege
foo :: (α,β) -> α
```

What could possibly happen?

What could possibly happen?
Pure Functions

Java

T foo(Pair<T,U> p) {...}

Frege

foo :: (α,β) -> α

Everything!
State changes, file or db access, missile launch,...

a is returned
Pure Functions

can be **cached** (memoized)

can be evaluated **lazily**

can be evaluated **in advance**

can be evaluated **concurrently**

can be **eliminated**

in common subexpressions

can be **optimized**
Is my method pure?

Let the **type system** find out!
Java Interoperability

Do not mix OO and FP, combine them!
Java -> Frege

Frege compiles Haskell to Java source and byte code.

Just call that.

You can get help by using the :java command in the REPL.
Frege -> Java

pure native encode java.net.URLEncoder.encode :: String -> String
encode “Dierk König”

native millis java.lang.System.currentTimeMillis :: () -> IO Long
millis ()
millis ()
past = millis () - 1000

Does not compile!

This is a key distinction between Frege and other JVM languages!

even Java can be pure
Frege allows calling Java but never unprotected!

is explicit about effects just like Haskell
Type System

Global type inference

More safety and less work for the programmer

You don’t need to specify any types at all!

But sometimes you do anyway...
Keep the mess out!

Mutable I/O

Mutable

Pure Computation

Mutable

Pure Computation

Pure Computation

Pure Computation
Ok, these are Monads. Be brave. Think of them as contexts that the type system propagates and makes un-escapable.
Fizzbuzz

http://c2.com/cgi/wiki?FizzBuzzTest

https://dierk.gitbooks.io/fregegoodness/
  chapter 8 „FizzBuzz“
public class FizzBuzz{
    public static void main(String[] args){
        for(int i= 1; i <= 100; i++){
            if(i % 15 == 0){
                System.out.println(„FizzBuzz”);
            }else if(i % 3 == 0){
                System.out.println(“Fizz”);
            }else if(i % 5 == 0){
                System.out.println(“Buzz”);
            }else{
                System.out.println(i);
            }
        }
    }
}
Fizzbuzz Logical

fizzes = cycle ['','','fizz']
buzzes = cycle ['','','','','buzz']
pattern = zipWith (++) fizzes buzzes
numbers = map show [1..]
fizzbuzz = zipWith max pattern numbers

main _ = for (take 100 fizzbuzz) println
# Fizzbuzz Comparison

<table>
<thead>
<tr>
<th></th>
<th>Imperative</th>
<th>Logical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditionals</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Operators</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Nesting level</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Sequencing</td>
<td>sensitive</td>
<td>transparent</td>
</tr>
<tr>
<td>Maintainability</td>
<td>- - -</td>
<td>+</td>
</tr>
<tr>
<td>Incremental development</td>
<td>-</td>
<td>+++</td>
</tr>
</tbody>
</table>
Unique in Frege

Global type inference requires a purely functional language

(only expressions and parametric polymorphism)

Purity by default

effects are explicit in the type system

Laziness by default

Values are always immutable

Guarantees extend into Java calls
Why Frege

Robustness under parallel execution
Robustness under composition
Robustness under increments
Robustness under refactoring

Enables local and equational reasoning

Best way to learn FP
Why Frege

it is just a pleasure to work with
How?

http://www.frege-lang.org
@fregelang
stackoverflow „frege“ tag
edX FP101 MOOC
Dierk König
canoo

Please give feedback!
mittie
FGA

Language level is Haskell Report 2010. Yes, performance is roughly ~ Java. Yes, the compiler is reasonably fast. Yes, we have an Eclipse Plugin. Yes, Maven/Gradle/etc. integration. Yes, we have HAMT (aka HashMap). Yes, we have QuickCheck (+shrinking) No, but STM is in the works.