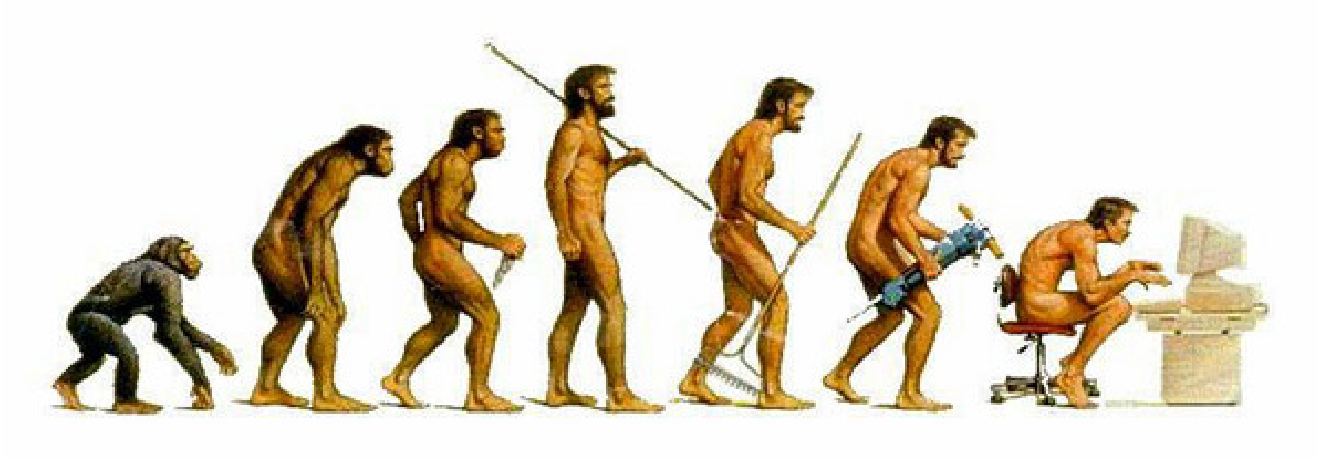
Faith, Evolution, and Programming Languages

Philip Wadler
University of Edinburgh



Evolution

Multiculturalism

Static vs. Dynamic Typing

Part I. Church: The origins of faith

Part II. Haskell: Type Classes

Part III. Java: Generics

Part IV. Links: Reconciliation

Part I

Church: The origins of faith

Gerhard Gentzen (1909–1945)

Gerhard Gentzen (1935) — Natural Deduction

$\supset$–I

$$\frac{\begin{matrix}[\mathfrak{A}] \\ \mathfrak{B}\end{matrix}}{\mathfrak{A} \supset \mathfrak{B}}$$

$\supset$–E

$$\frac{\mathfrak{A} \qquad \mathfrak{A} \supset \mathfrak{B}}{\mathfrak{B}}$$

&–I

$$\frac{\mathfrak{A} \qquad \mathfrak{B}}{\mathfrak{A} \,\&\, \mathfrak{B}}$$

&–E

$$\frac{\mathfrak{A} \,\&\, \mathfrak{B}}{\mathfrak{A}} \qquad \frac{\mathfrak{A} \,\&\, \mathfrak{B}}{\mathfrak{B}}$$

Gerhard Gentzen (1935) — Natural Deduction

$$\dfrac{\begin{array}{c}[A]^x \\ \vdots \\ B\end{array}}{A \supset B}\ \supset\text{-}\mathbf{I}^x \qquad \dfrac{A \supset B \qquad A}{B}\ \supset\text{-}\mathbf{E}$$

$$\dfrac{A \qquad B}{A \mathbin{\&} B}\ \&\text{-}\mathbf{I} \qquad \dfrac{A \mathbin{\&} B}{A}\ \&\text{-}\mathbf{E}_0 \qquad \dfrac{A \mathbin{\&} B}{B}\ \&\text{-}\mathbf{E}_1$$

Simplifying a proof

$$
\dfrac{
  \dfrac{
    \dfrac{[B \mathbin{\&} A]^z}{A}\ \&\text{-E}_1
    \qquad
    \dfrac{[B \mathbin{\&} A]^z}{B}\ \&\text{-E}_0
  }{
    \dfrac{A \mathbin{\&} B}{(B \mathbin{\&} A) \supset (A \mathbin{\&} B)}\ \supset\text{-I}^z
  }\ \&\text{-I}
  \qquad
  \dfrac{[B]^y \qquad [A]^x}{B \mathbin{\&} A}\ \&\text{-I}
}{A \mathbin{\&} B}\ \supset\text{-E}
$$

Simplifying a proof

$$\dfrac{\dfrac{\dfrac{\dfrac{[B \mathbin{\&} A]^z}{A}\,\&\text{-E}_1 \qquad \dfrac{[B \mathbin{\&} A]^z}{B}\,\&\text{-E}_0}{A \mathbin{\&} B}\,\&\text{-I}}{(B \mathbin{\&} A) \supset (A \mathbin{\&} B)}\,\supset\text{-I}^z \qquad \dfrac{[B]^y \quad [A]^x}{B \mathbin{\&} A}\,\&\text{-I}}{A \mathbin{\&} B}\,\supset\text{-E}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{[B]^y \quad [A]^x}{B \mathbin{\&} A}\,\&\text{-I}}{A}\,\&\text{-E}_1 \qquad \dfrac{\dfrac{[B]^y \quad [A]^x}{B \mathbin{\&} A}\,\&\text{-I}}{B}\,\&\text{-E}_0}{A \mathbin{\&} B}\,\&\text{-I}$$

Simplifying a proof

$$\dfrac{\dfrac{\dfrac{\dfrac{[B \mathbin{\&} A]^z}{A}\,\&\text{-E}_1 \qquad \dfrac{[B \mathbin{\&} A]^z}{B}\,\&\text{-E}_0}{A \mathbin{\&} B}\,\&\text{-I}}{(B \mathbin{\&} A) \supset (A \mathbin{\&} B)}\,\supset\text{-I}^z \qquad \dfrac{[B]^y \quad [A]^x}{B \mathbin{\&} A}\,\&\text{-I}}{A \mathbin{\&} B}\,\supset\text{-E}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{[B]^y \quad [A]^x}{B \mathbin{\&} A}\,\&\text{-I}}{A}\,\&\text{-E}_1 \qquad \dfrac{\dfrac{[B]^y \quad [A]^x}{B \mathbin{\&} A}\,\&\text{-I}}{B}\,\&\text{-E}_0}{A \mathbin{\&} B}\,\&\text{-I}$$

$$\Downarrow$$

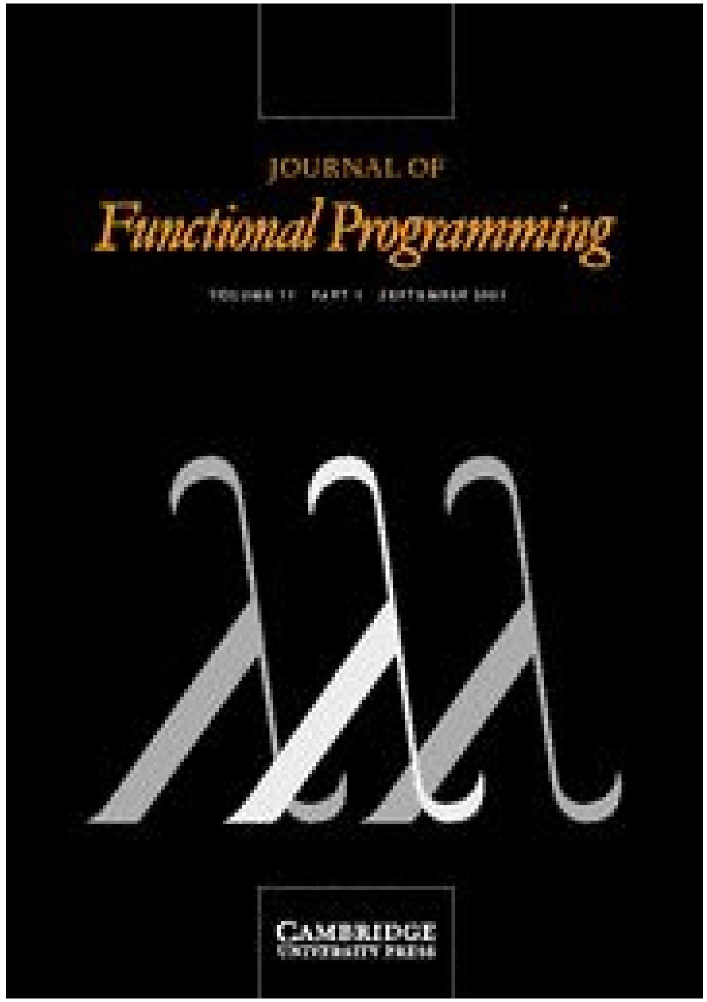
$$\dfrac{[A]^x \quad [B]^y}{A \mathbin{\&} B}\,\&\text{-I}$$

Alonzo Church (1903–1995)

Alonzo Church (1932) — Lambda calculus

An occurrence of a variable **x** in a given formula is called an occurrence of **x** as a *bound variable* in the given formula if it is an occurrence of **x** in a part of the formula of the form $\lambda \mathbf{x}[\mathbf{M}]$; that is, if there is a formula **M** such that $\lambda \mathbf{x}[\mathbf{M}]$ occurs in the given formula and the occurrence of **x** in question is an occurrence in $\lambda \mathbf{x}[\mathbf{M}]$. All other occurrences of a variable in a formula are called occurrences as a *free variable*.

A formula is said to be *well-formed* if it is a variable, or if it is one


JOURNAL OF
Functional Programming
CAMBRIDGE
UNIVERSITY PRESS

Alonzo Church (1940) — Typed λ-calculus

$$\dfrac{\begin{matrix}[x : A]^x \\ \vdots \\ u : B\end{matrix}}{\lambda x.\, u : A \supset B} \supset\text{-}\mathbf{I}^x \qquad \dfrac{s : A \supset B \qquad t : A}{s\, t : B} \supset\text{-}\mathbf{E}$$

$$\dfrac{t : A \qquad u : B}{\langle t, u\rangle : A \mathbin{\&} B} \&\text{-}\mathbf{I} \qquad \dfrac{s : A \mathbin{\&} B}{s_0 : A} \&\text{-}\mathrm{E}_0 \qquad \dfrac{s : A \mathbin{\&} B}{s_1 : B} \&\text{-}\mathrm{E}_1$$

Simplifying a program

$$
\dfrac{
  \dfrac{
    \dfrac{[z : B \,\&\, A]^z}{z_1 : A}\;\&\text{-E}_1
    \qquad
    \dfrac{[z : B \,\&\, A]^z}{z_0 : B}\;\&\text{-E}_0
  }{\langle z_1, z_0\rangle : A \,\&\, B}\;\&\text{-I}
}{\lambda z.\, \langle z_1, z_0\rangle : (B \,\&\, A) \supset (A \,\&\, B)}\;\supset\text{-I}^z
\qquad
\dfrac{[y : B]^y \quad [x : A]^x}{\langle y, x\rangle : B \,\&\, A}\;\&\text{-I}
$$

$$
\overline{(\lambda z.\, \langle z_1, z_0\rangle)\, \langle y, x\rangle : A \,\&\, B}\;\supset\text{-E}
$$

Simplifying a program

$$
\dfrac{\dfrac{\dfrac{\dfrac{[z : B \mathbin{\&} A]^z}{z_1 : A}\,\&\text{-E}_1 \qquad \dfrac{[z : B \mathbin{\&} A]^z}{z_0 : B}\,\&\text{-E}_0}{\langle z_1, z_0 \rangle : A \mathbin{\&} B}\,\&\text{-I}}{\lambda z.\, \langle z_1, z_0 \rangle : (B \mathbin{\&} A) \supset (A \mathbin{\&} B)}\,\supset\text{-I}^z \qquad \dfrac{[y : B]^y \quad [x : A]^x}{\langle y, x \rangle : B \mathbin{\&} A}\,\&\text{-I}}{(\lambda z.\, \langle z_1, z_0 \rangle)\, \langle y, x \rangle : A \mathbin{\&} B}\,\supset\text{-E}
$$

$$\Downarrow$$

$$
\dfrac{\dfrac{\dfrac{[y : B]^y \quad [x : A]^x}{\langle y, x \rangle : B \mathbin{\&} A}\,\&\text{-I}}{\langle y, x \rangle_1 : A}\,\&\text{-E}_1 \qquad \dfrac{\dfrac{[y : B]^y \quad [x : A]^x}{\langle y, x \rangle : B \mathbin{\&} A}\,\&\text{-I}}{\langle y, x \rangle_0 : B}\,\&\text{-E}_0}{\langle \langle y, x \rangle_1, \langle y, x \rangle_0 \rangle : A \mathbin{\&} B}\,\&\text{-I}
$$

Simplifying a program

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{[z : B \mathbin{\&} A]^z}{z_1 : A}\;\&\text{-E}_1
\qquad
\dfrac{[z : B \mathbin{\&} A]^z}{z_0 : B}\;\&\text{-E}_0
}{\langle z_1, z_0 \rangle : A \mathbin{\&} B}\;\&\text{-I}
}{\lambda z.\, \langle z_1, z_0 \rangle : (B \mathbin{\&} A) \supset (A \mathbin{\&} B)}\;\supset\text{-I}^z
\qquad
\dfrac{[y : B]^y \quad [x : A]^x}{\langle y, x \rangle : B \mathbin{\&} A}\;\&\text{-I}
}{(\lambda z.\, \langle z_1, z_0 \rangle)\, \langle y, x \rangle : A \mathbin{\&} B}\;\supset\text{-E}
$$

$$\Downarrow$$

$$
\dfrac{
\dfrac{
\dfrac{[y : B]^y \quad [x : A]^x}{\langle y, x \rangle : B \mathbin{\&} A}\;\&\text{-I}
}{\langle y, x \rangle_1 : A}\;\&\text{-E}_1
\qquad
\dfrac{
\dfrac{[y : B]^y \quad [x : A]^x}{\langle y, x \rangle : B \mathbin{\&} A}\;\&\text{-I}
}{\langle y, x \rangle_0 : B}\;\&\text{-E}_0
}{\langle \langle y, x \rangle_1, \langle y, x \rangle_0 \rangle : A \mathbin{\&} B}\;\&\text{-I}
$$

$$\Downarrow$$

$$
\dfrac{[x : A]^x \quad [y : B]^y}{\langle x, y \rangle : A \mathbin{\&} B}\;\&\text{-I}
$$

William Howard (1980) — Curry-Howard Isomorphism

THE FORMULAE-AS-TYPES NOTION OF CONSTRUCTION

W. A. Howard

Department of Mathematics, University of Illinois at Chicago Circle, Chicago, Illinois 60680, U.S.A.

Dedicated to H. B. Curry on the occasion of his 80th birthday.

The following consists of notes which were privately circulated in 1969. Since they have been referred to a few times in the literature, it seems worth while to publish them. They have been rearranged for easier reading, and some inessential corrections have been made.

Curry-Howard

Hindley-Milner

Girard-Reynolds

Part II

Haskell: Type Classes

Type classes

```
class Ord a where
  (<) :: a -> a -> Bool

instance Ord Int where
  (<) = primitiveLessInt

instance Ord Char where
  (<) = primitiveLessChar

max  :: (Ord a) => a -> a -> a
max x y  =  if x < y then y else x

maximum :: (Ord a) => [a] -> a
maximum [x]  =  x
maximum (x:xs)  =  max x (maximum xs)

test  :: Bool
test  =  maximum [0,1,2] == 2 &&
         maximum "abc" == 'c'
```

Translation

```
data Ord a = Ord { less :: a -> a -> Bool }

ordInt :: Ord Int
ordInt =  Ord { less = primitiveLessInt }

ordChar :: Ord Char
ordChar =  Ord { less = primitiveLessChar }

max  :: Ord a -> a -> a -> a
max d x y  =  if less d x y then y else x

maximum :: Ord a -> [a] -> a
maximum d [x]       =  x
maximum d (x:xs)    =  max d x (maximum d xs)

test  :: Bool
test  =  maximum ordInt [0,1,2] == 2 &&
         maximum ordChar "abc" == 'c'
```

Type classes, continued

```
instance Ord a => Ord [a] where
  [] < []                        =  False
  [] < y:ys                      =  True
  x:xs < []                      =  False
  x:xs < y:ys | x < y            =  True
              | y < x            =  False
              | otherwise        =  xs < ys

test' :: Bool
test' =  maximum ["zero","one","two"] == "zero" &&
         maximum [[[0],[1]],[[0,1]]]  == [[0,1]]
```

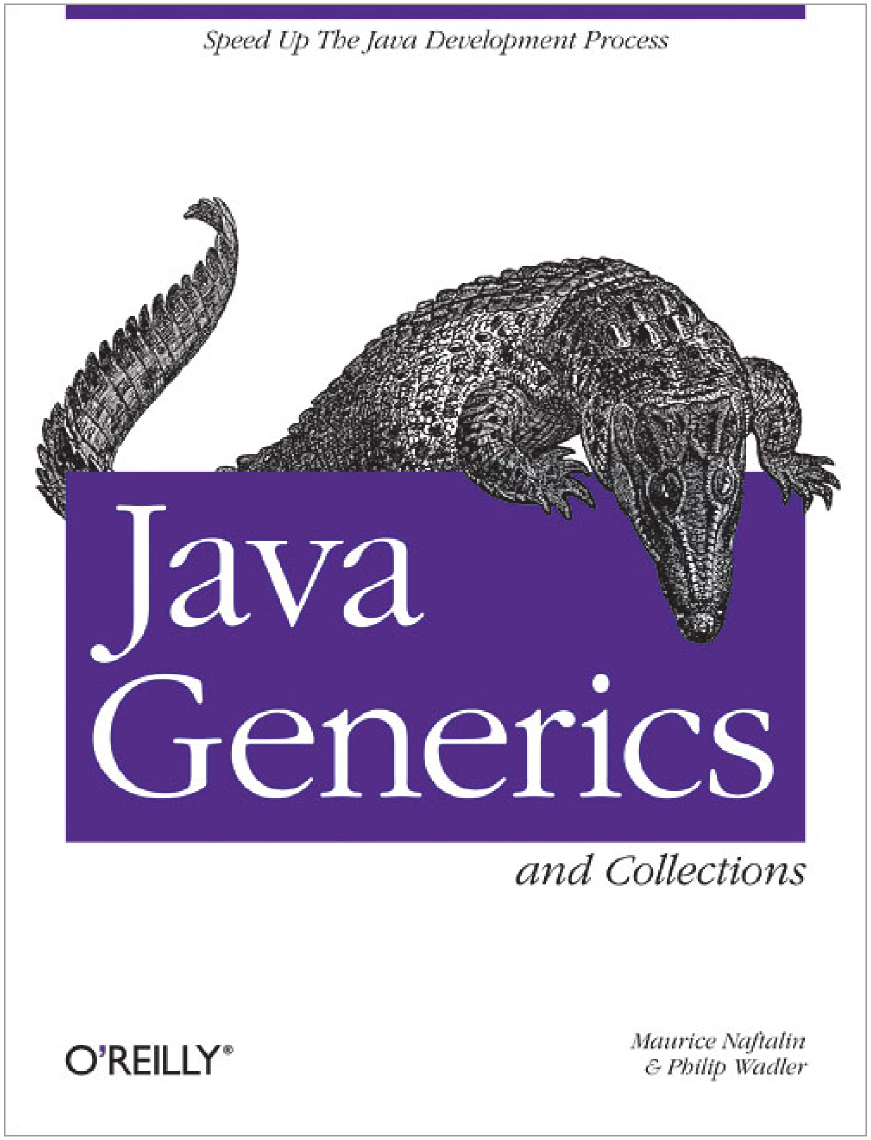
Translation, continued

```
ordList :: Ord a -> Ord [a]
ordList d  =  Ord { less = lt }
  where
  lt d [] []                         =  False
  lt d [] (y:ys)                     =  True
  lt d (x:xs) []                     =  False
  lt d (x:xs) (y:ys) | x < y         =  True
                     | y < x         =  False
                     | otherwise     =  lt d xs ys

test' :: Bool
test' =  maximum d0 ["zero","one","two"] == "zero" &&
         maximum d1 [[[0],[1]],[[0,1]]] == [[0,1]]
  where
  d0 = ordList ordChar
  d1 = ordList (ordList ordInt)
```

Part III

Java: Generics

Java generics



Lists in Java 4.0 and Java 5.0

	in Java 4.0	in Java 5.0
integer list		
string list		
string list list		

Lists in Java 4.0 and Java 5.0

	in Java 4.0	in Java 5.0
integer list	`List`	
string list		
string list list		

Lists in Java 4.0 and Java 5.0

	in Java 4.0	in Java 5.0
integer list	`List`	
string list	`List`	
string list list		

Lists in Java 4.0 and Java 5.0

	in Java 4.0	in Java 5.0
integer list	`List`	
string list	`List`	
string list list	`List`	

Lists in Java 4.0 and Java 5.0

	in Java 4.0	in Java 5.0
integer list	`List`	`List<Integer>`
string list	`List`	`List<String>`
string list list	`List`	`List<List<String>>`

Legacy library with legacy client

```
class Stack {
  private List list;
  public Stack() { list = new ArrayList(); }
  public boolean empty() { return list.size() == 0; }
  public void push(Object elt) { list.add(elt); }
  public Object pop() {
    Object elt = list.remove(list.size()-1);
    return elt;
  }
}

class Client {
  public static void main(String[] args) {
    Stack stack = new Stack();
    stack.push(new Integer(42));
    int top = ((Integer)stack.pop()).intValue();
    assert top == 42;
  }
}
```

Generic library with generic client

```
class Stack<E> {
  private List<E> list;
  public Stack() { list = new ArrayList<E>(); }
  public boolean empty() { return list.size() == 0; }
  public void push(E elt) { list.add(elt); }
  public E pop() {
    E elt = list.remove(list.size()-1);
    return elt;
  }
}

class Client {
  public static void main(String[] args) {
    Stack<Integer> stack = new Stack<Integer>();
    stack.push(42);
    int top = stack.pop();
    assert top == 42;
  }
}
```

Generic library with legacy client

```
class Stack<E> {
  private List<E> list;
  public Stack() { list = new ArrayList<E>(); }
  public boolean empty() { return list.size() == 0; }
  public void push(E elt) { list.add(elt); }
  public E pop() {
    E elt = list.remove(list.size()-1);
    return elt;
  }
}

class Client {
  public static void main(String[] args) {
    Stack stack = new Stack();  // raw type
    stack.push(new Integer(42));
    int top = ((Integer)stack.pop()).intValue();
    assert top == 42;
  }
}
```

Legacy library with generic client—minimal changes

```
class Stack<E> {
  private List list;  // raw type
  public Stack() { list = new ArrayList(); }
  public boolean empty() { return list.size() == 0; }
  public void push(E elt) { list.add(elt); }
  public E pop() {
    Object elt = list.remove(list.size()-1);
    return (E)elt;  // unchecked cast
  }
}

class Client {
  public static void main(String[] args) {
    Stack<Integer> stack = new Stack<Integer>();
    stack.push(42);
    int top = stack.pop();
    assert top == 42;
  }
}
```

Legacy library with generic client—stubs

```
class Stack<E> {
  public Stack() { throw new StubException(); }
  public boolean empty() { throw new StubException(); }
  public void push(E elt) { throw new StubException(); }
  public E pop() { throw new StubException(); }
}

class Client {
  public static void main(String[] args) {
    Stack<Integer> stack = new Stack<Integer>();
    stack.push(42);
    int top = stack.pop();
    assert top == 42;
  }
}

// compile with stub library, execute with legacy library
% javac -classpath stubs Client.java
% java -ea -classpath legacy Client
```

Comparison

```
interface Comparable<T> {
  public int compareTo(T o);
}

Integer int0 = 0;
Integer int1 = 1;
assert int0.compareTo(int1) < 0;

String str0 = "zero";
String str1 = "one";
assert str0.compareTo(str1) > 0;

Number num0 = 0;
Number num1 = 1.0;
assert num0.compareTo(num1) < 0;  // compile-time error
```

Maximum of a list

```
public static <T extends Comparable<T>>
  T max(List<T> elts)
{
  T candidate = elts.get(0);
  for (T elt : elts) {
    if (candidate.compareTo(elt) < 0) candidate = elt;
  }
  return candidate;
}

List<Integer> ints = Arrays.asList(0,1,2);
assert max(ints) == 2;

List<String> strs = Arrays.asList("zero","one","two");
assert max(strs).equals("zero");

List<Number> nums = Arrays.asList(0,1,2,3.14);
assert max(nums) == 3.14;  // compile-time error
```

A fruity example

Permit comparison of apples with oranges

```
class Fruit implements Comparable<Fruit> {
  ... }
class Apple extends Fruit {
  ... }
class Orange extends Fruit {
  ... }
```

Prohibit comparison of apples with oranges

```
class Fruit {
  ... }
class Apple extends Fruit implements Comparable<Apple> {
  ... }
class Orange extends Fruit implements Comparable<Orange> {
  ... }
```

Comparing lists

```
class ComparableList<E extends Comparable<E>>
  extends ArrayList<E> implements Comparable<List<E>>
{
  public ComparableList(Iterable<E> list) {super(list)}
  public int compareTo(List<E> that) {
    int n1 = this.size();
    int n2 = that.size();
    for (int i = 0; i < Math.min(n1,n2); i++) {
      int k = list1.get(i).compareTo(list2.get(i));
      if (k != 0) return k;
    }
    return (n1 < n2) ? -1 : (n1 == n2) ? 0 : 1;
  }
}
```

Cumbersome to use—probably better to use comparators

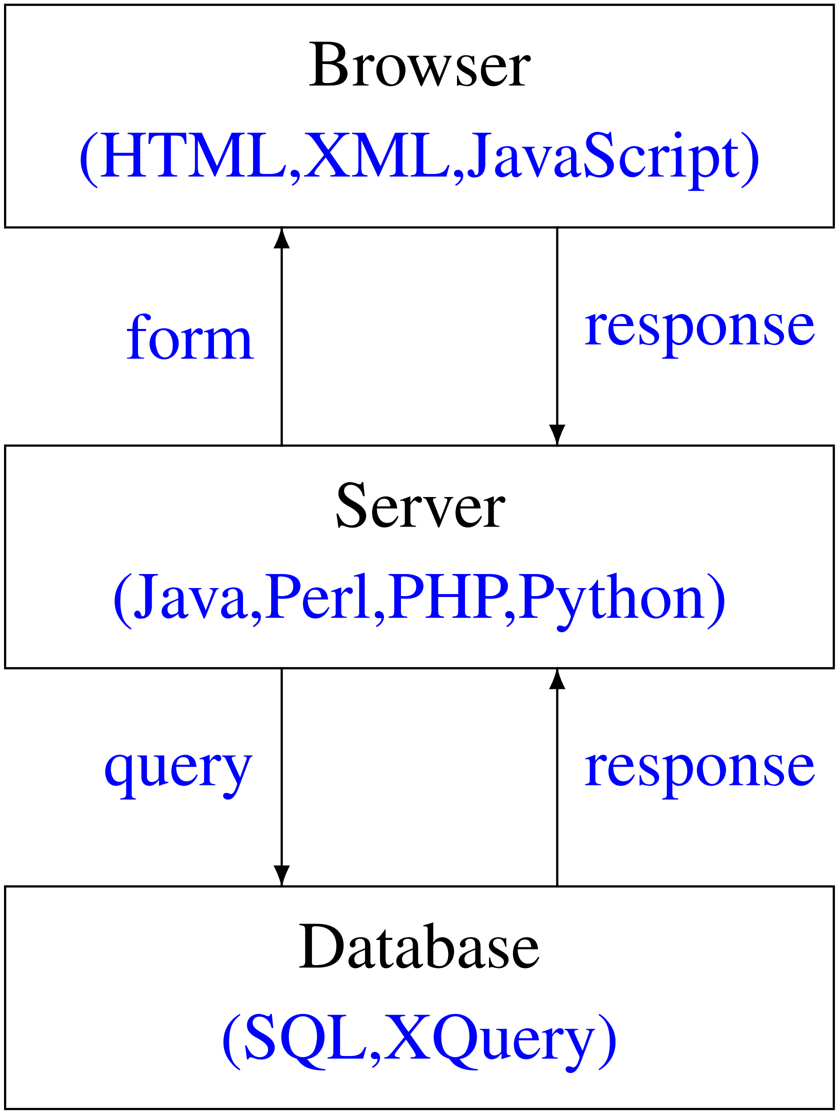
Maximum of a list of lists of lists of integers

```
ComparableList<ComparableList<Integer>> xss =
  new ComparableList<ComparableList<Integer>>{{
    add(new ComparableList<Integer>{{add(0)}});
    add(new ComparableList<Integer>{{add(1)}});}};
ComparableList<ComparableList<Integer>> yss =
  new ComparableList<ComparableList<Integer>>{{
    add(new ComparableList<Integer>{{add(0);add(1);}})}};
List<ComparableList<ComparableList<Integer>>> xsss =
  new List<ComparableList<ComparableList<Integer>>>{{
    add(xss); add(yss);}};
assert max(xsss).equals(yss);
```

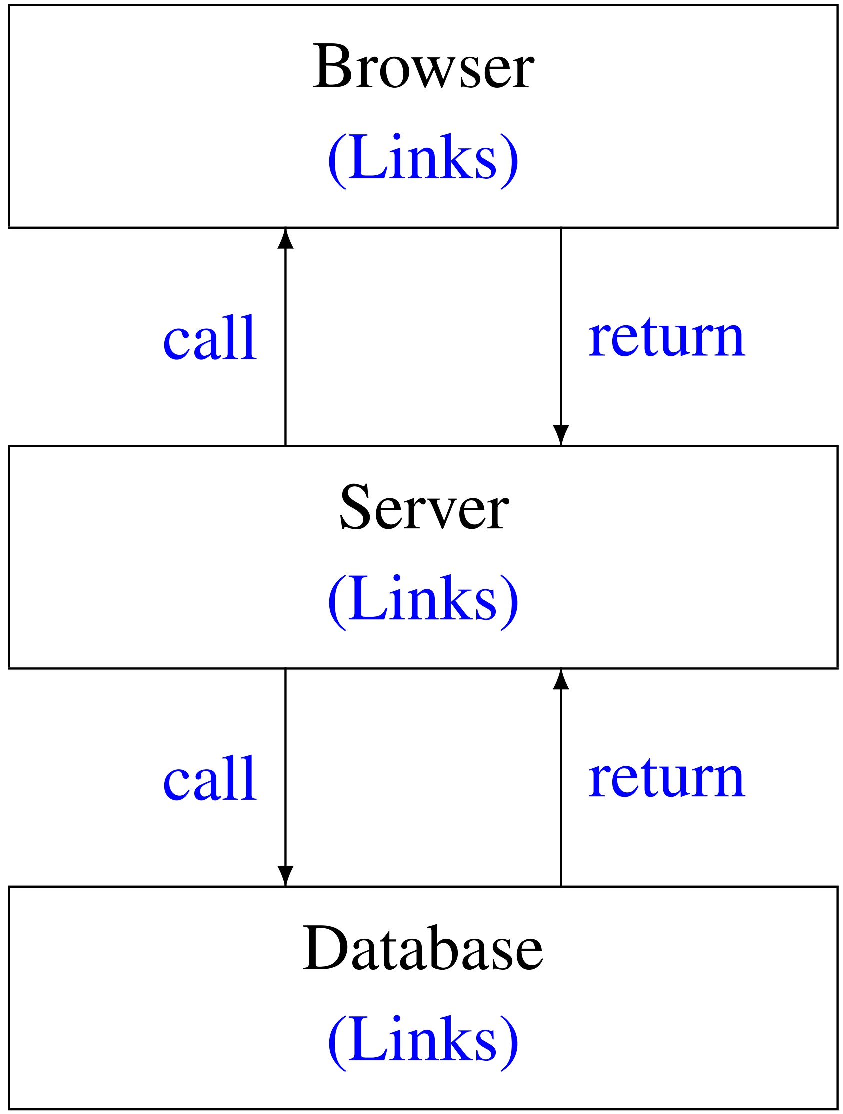
Part IV

Links: Reconciliation

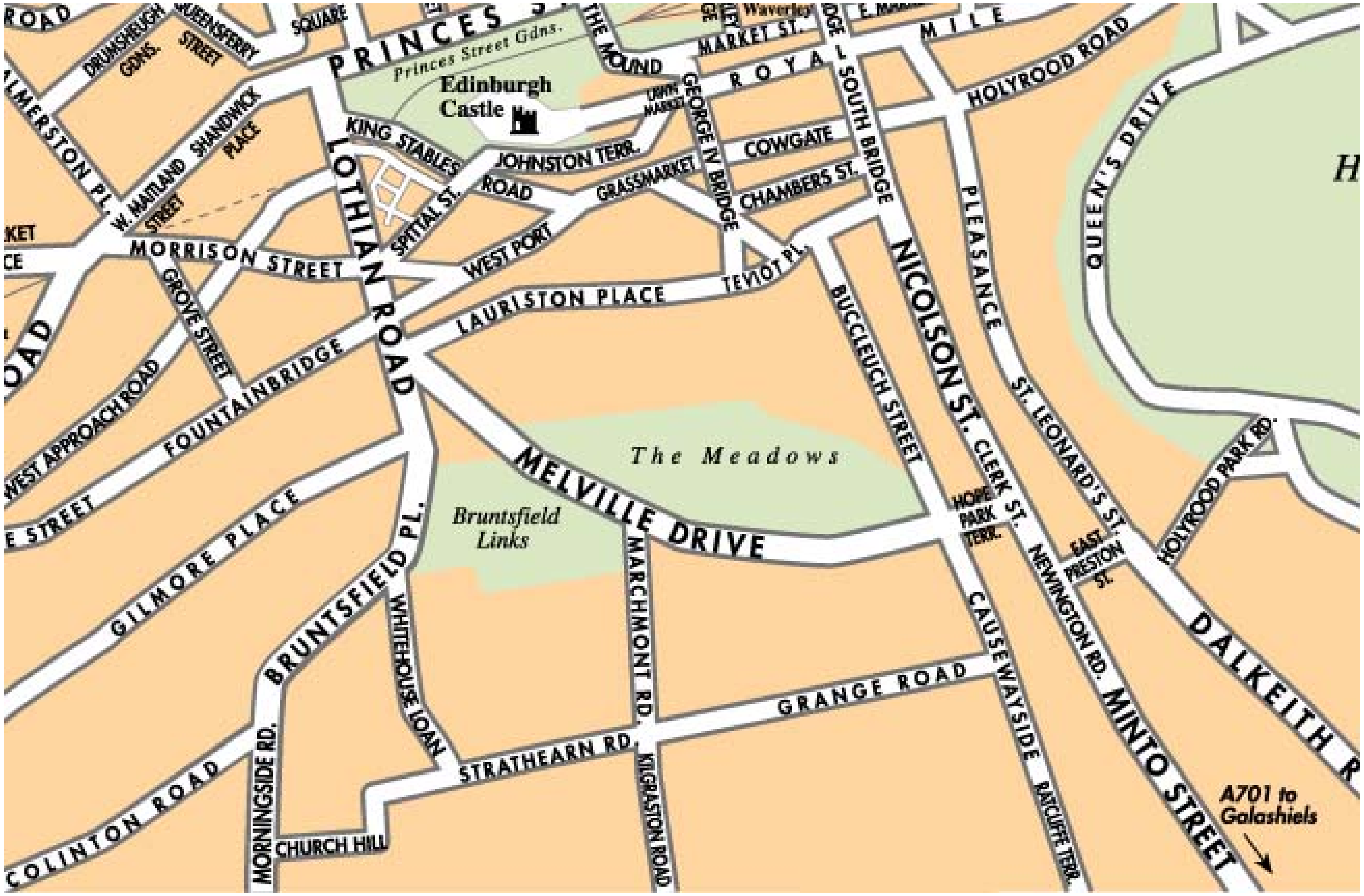
Links: Web Programming without Tiers



Links: Web Programming without Tiers



Hope — Burstall, MacQueen, and Sannella (1980)



Links — Cooper, Lindley, Wadler, and Yallop (2007)

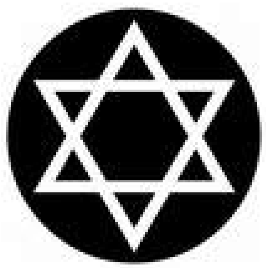
Type classes, revisited

```
(+)  :: Num a => a -> a -> a
1    :: Num a => a
2    :: Num a => a

1 + 1 == 2  ::  Num a => Bool
-- ambiguous!


read :: Read a => String -> a
show :: Show a => a -> String

show . read  ::  (Read a, Show a) => String -> String
-- ambiguous!
```

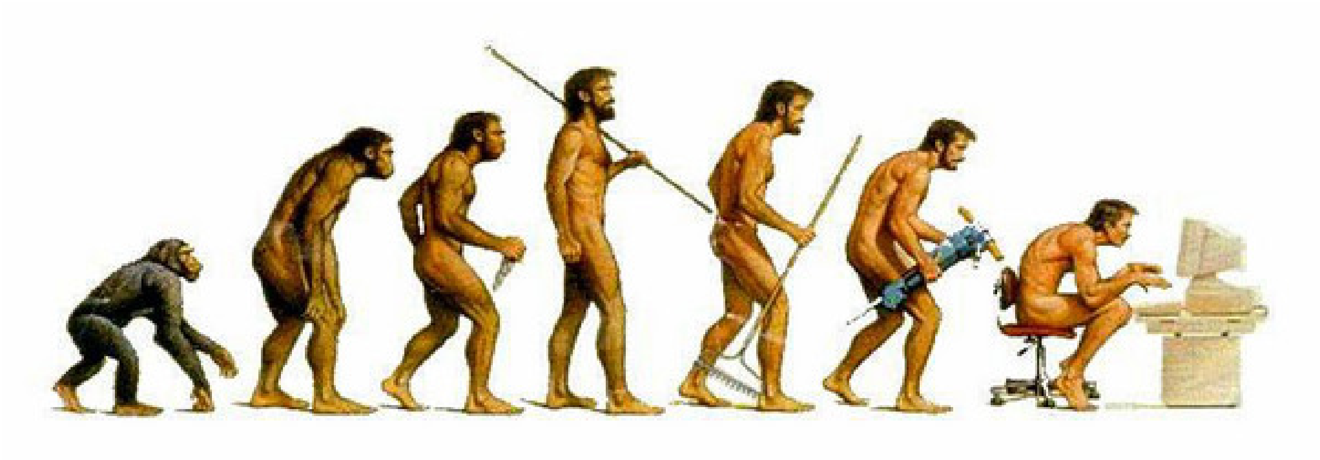
Type classes, revisited

```
(+) :: Num a => a -> a -> a
1   :: Int
2   :: Int

1 + 1 == 2  ::  Bool
-- unambiguous!


readInt :: String -> Int
show    :: Show a => a -> String

show . readInt  ::  String -> String
-- unambiguous!
```

Static vs. Dynamic Typing

Multiculturalism



Evolution

Faith, Evolution, and Programming Languages

Philip Wadler
University of Edinburgh