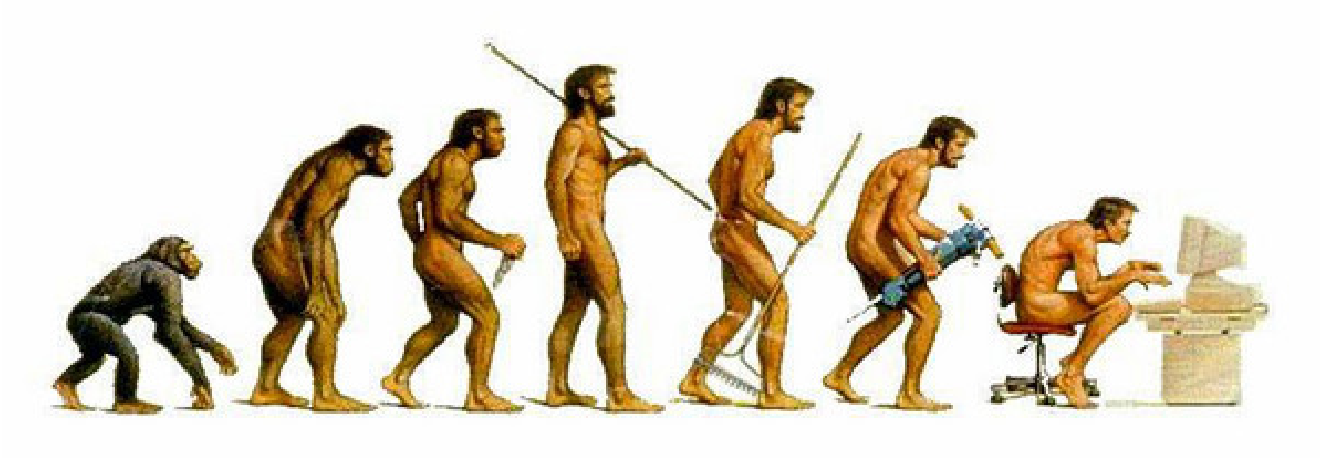
Faith, Evolution, and Programming Languages

Philip Wadler
University of Edinburgh

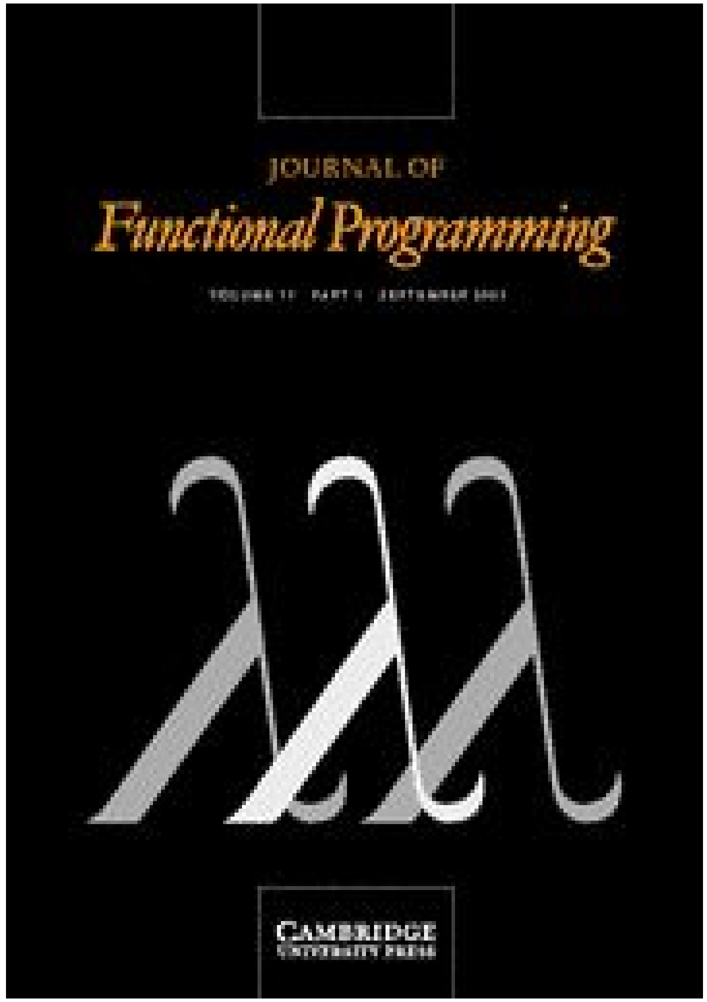


Evolution

Multiculturalism

Part I

Church: The origins of faith


JOURNAL OF
Functional Programming
CAMBRIDGE
UNIVERSITY PRESS

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

&–*I*

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

&–*E*

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

∨–*I*

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

∨–*E*

$$\frac{\mathfrak{A} \vee \mathfrak{B} \quad \begin{matrix}[\mathfrak{A}] \\ \mathfrak{C}\end{matrix} \quad \begin{matrix}[\mathfrak{B}] \\ \mathfrak{C}\end{matrix}}{\mathfrak{C}}$$

∀–*I*

$$\frac{\mathfrak{F}a}{\forall \mathfrak{x}\, \mathfrak{F}\mathfrak{x}}$$

∀–*E*

$$\frac{\forall \mathfrak{x}\, \mathfrak{F}\mathfrak{x}}{\mathfrak{F}a}$$

∃–*I*

$$\frac{\mathfrak{F}a}{\exists \mathfrak{x}\, \mathfrak{F}\mathfrak{x}}$$

∃–*E*

$$\frac{\exists \mathfrak{x}\, \mathfrak{F}\mathfrak{x} \quad \begin{matrix}[\mathfrak{F}a] \\ \mathfrak{C}\end{matrix}}{\mathfrak{C}}$$

⊃–*I*

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

⊃–*E*

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

¬–*I*

$$\frac{\begin{matrix}[\mathfrak{A}] \\ \curlywedge\end{matrix}}{\neg \mathfrak{A}}$$

¬–*E*

$$\frac{\mathfrak{A} \quad \neg \mathfrak{A}}{\curlywedge} \qquad \frac{\curlywedge}{\mathfrak{D}} \;.$$

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

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

Simplifying a proof

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

$$\Downarrow$$

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

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

Alonzo Church (1940) — Typed λ-calculus

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

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

Simplifying a program

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

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

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

$$\Downarrow$$

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

$$\Downarrow$$

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

William Howard (1980) — Curry-Howard Isomorphism

THE FORMULAE-AS-TYPES NOTION OF CONSTRUCTION

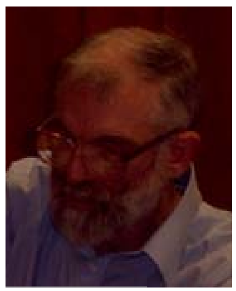
W. A. Howard

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

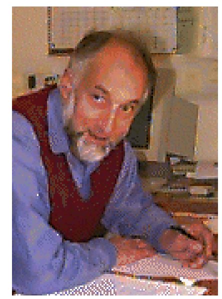
Dedicated to H. B. Curry on the occasion of his 80th birthday.

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



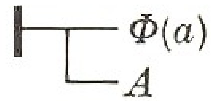
Girard-Reynolds

Part II

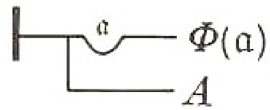
Second-order logic,
Polymorphism,
and Java

Gottlob Frege (1879) — Quantifiers (∀)

It is clear also that from



we can derive



if A is an expression in which a does not occur and if a stands only in the argument places of $\Phi(a)$.[14] If $-\!\overset{\mathfrak{a}}{\smile}\!-\Phi(\mathfrak{a})$ is denied, we must be able to specify a meaning for a such that $\Phi(a)$ will be denied. If, therefore, $-\!\overset{\mathfrak{a}}{\smile}\!-\Phi(\mathfrak{a})$ were to be denied and

John Reynolds (1974) — Polymorphism

TOWARDS A THEORY OF TYPE STRUCTURE †

John C. Reynolds

Syracuse University

Syracuse, New York 13210, U.S.A.

Introduction

The type structure of programming languages has been the subject of an active development characterized by continued controversy over basic principles.[1-7] In this paper, we formalize a view of these principles somewhat similar to that of J. H. Morris.[5] We introduce an extension of the typed lambda calculus which permits user-defined types and polymorphic functions, and show that the semantics of this language satisfies a representation theorem which embodies our notion of a "correct" type structure.

Syntax

To formalize the syntax of our language, we begin with two disjoint, countably infinite sets: the set T of type variables and the set V of normal variables. Then W, the set of type expressions, is the minimal set satisfying:

(1a) If $t \in T$ then:

$t \in W$.

(1b) If $w_1, w_2 \in W$ then:

$(w_1 \rightarrow w_2) \in W$.

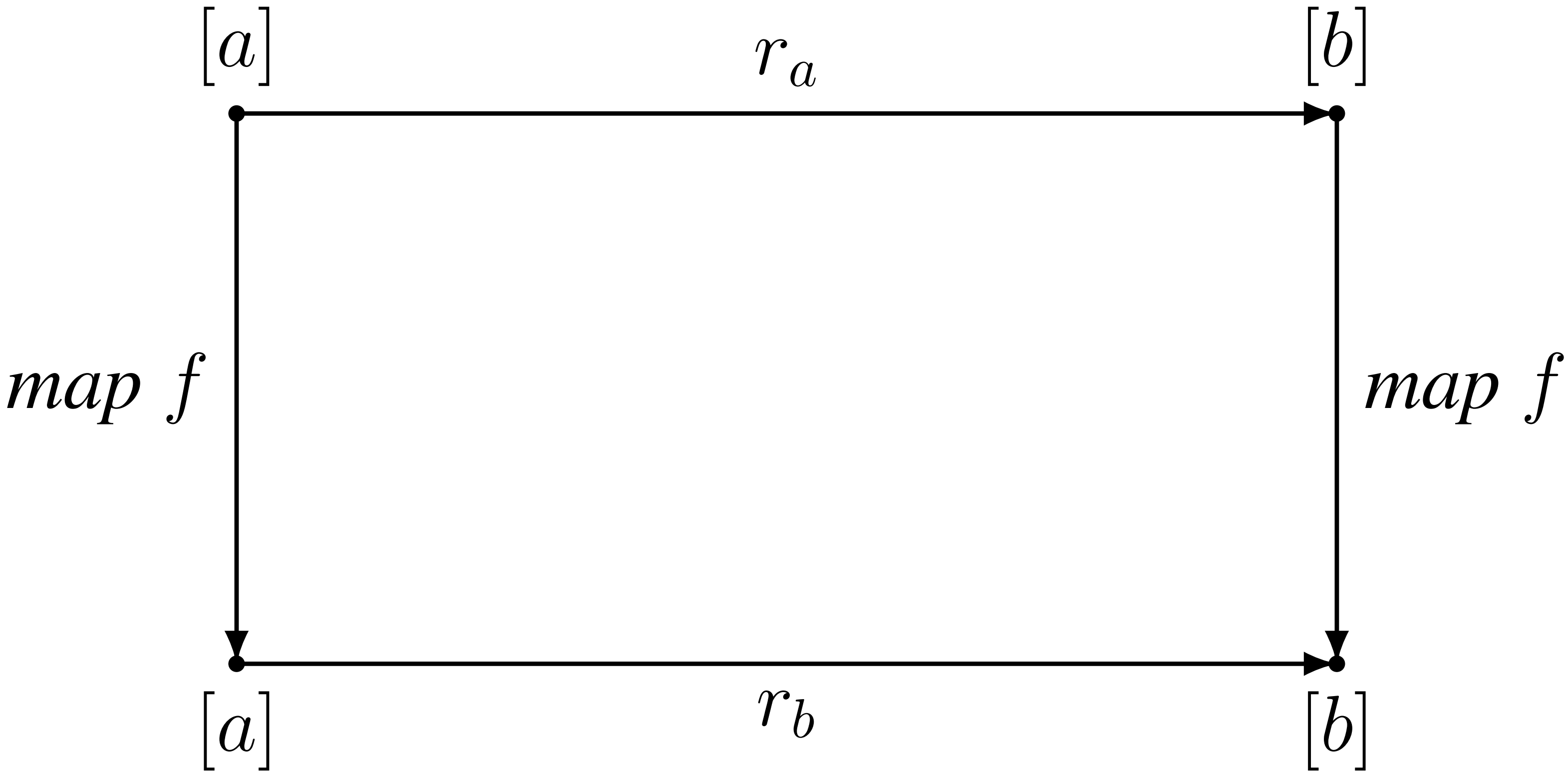
(1c) If $t \in T$ and $w \in W$ then:

$(\Delta t.\ w) \in W$.

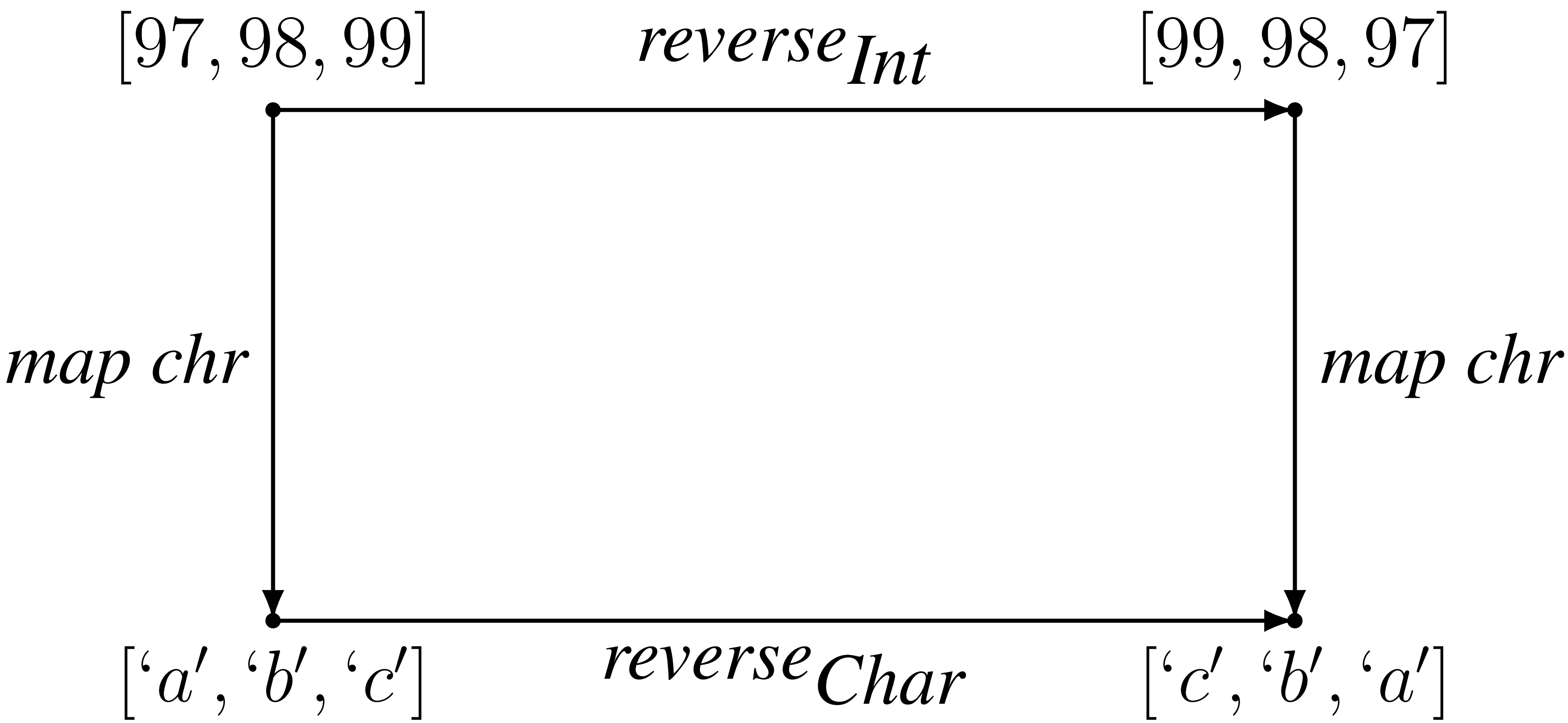
A magic trick

$$r :: [a] \rightarrow [a]$$

Theorems for Free!



Theorems for Free!



Odersky and Wadler (1997) — Pizza

Pizza into Java: Translating theory into practice

Martin Odersky
University of Karlsruhe

Philip Wadler
University of Glasgow

Example 2.1 Polymorphism in Pizza

```
class Pair<elem> {
    elem x;  elem y;
    Pair (elem x, elem y) {this.x = x; this.y = y;}
    void swap () {elem t = x; x = y; y = t;}
}

Pair<String> p = new Pair("world!", "Hello,");
p.swap();
System.out.println(p.x + p.y);

Pair<int> q = new Pair(22, 64);
q.swap();
System.out.println(q.x − q.y);
```

Example 2.3 Homogenous translation of polymorphism into Java

```
class Pair {
    Object x;  Object y;
    Pair (Object x, Object y) {this.x = x; this.y = y;}
    void swap () {Object t = x; x = y; y = t;}
}

class Integer {
    int i;
    Integer (int i) { this.i = i; }
    int intValue() { return i; }
}

Pair p = new Pair((Object)"world!", (Object)"Hello,");
p.swap();
System.out.println((String)p.x + (String)p.y);

Pair q = new Pair((Object)new Integer(22),
                  (Object)new Integer(64));
q.swap();
System.out.println(((Integer)(q.x)).intValue() −
                  ((Integer)(q.y)).intValue());
```

Igarashi, Pierce, and Wadler (1999) — Featherweight Java

$$\Gamma \vdash \mathtt{x} : \Gamma(\mathtt{x})$$

$$\frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{C}_0 \qquad \mathit{fields}(\mathtt{C}_0) = \overline{\mathtt{C}}\ \overline{\mathtt{f}}}{\Gamma \vdash \mathtt{e}_0.\mathtt{f}_i : \mathtt{C}_i}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{C}_0 \qquad \mathit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{D}} \rightarrow \mathtt{C} \qquad \Gamma \vdash \overline{\mathtt{e}} : \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} <: \overline{\mathtt{D}}}{\Gamma \vdash \mathtt{e}_0.\mathtt{m}(\overline{\mathtt{e}}) : \mathtt{C}}$$

$$\frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{D}}\ \overline{\mathtt{f}} \qquad \Gamma \vdash \overline{\mathtt{e}} : \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} <: \overline{\mathtt{D}}}{\Gamma \vdash \mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}}) : \mathtt{C}}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{D} \qquad \mathtt{D} <: \mathtt{C}}{\Gamma \vdash (\mathtt{C})\mathtt{e}_0 : \mathtt{C}}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{D} \qquad \mathtt{C} <: \mathtt{D} \qquad \mathtt{C} \neq \mathtt{D}}{\Gamma \vdash (\mathtt{C})\mathtt{e}_0 : \mathtt{C}}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{D} \qquad \mathtt{C} \not<: \mathtt{D} \qquad \mathtt{D} \not<: \mathtt{C} \qquad \mathit{stupid\ warning}}{\Gamma \vdash (\mathtt{C})\mathtt{e}_0 : \mathtt{C}}$$

Igarashi, Pierce, and Wadler (1999)
— Featherweight Generic Java

$$\Delta;\Gamma \vdash \mathtt{x} : \Gamma(\mathtt{x})$$

$$\frac{\Delta;\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0 \qquad \mathit{fields}(\mathit{bound}_\Delta(\mathtt{T}_0)) = \bar{\mathtt{T}}\ \bar{\mathtt{f}}}{\Delta;\Gamma \vdash \mathtt{e}_0.\mathtt{f}_i : \mathtt{T}_i}$$

$$\frac{\begin{array}{c}\Delta;\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0 \qquad \mathit{mtype}(\mathtt{m}, \mathit{bound}_\Delta(\mathtt{T}_0)) = \mathtt{<}\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{P}}\mathtt{>}\bar{\mathtt{U}} \to \mathtt{U} \\ \Delta \vdash \bar{\mathtt{V}}\ \mathrm{ok} \qquad \Delta \vdash \bar{\mathtt{V}} <: [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{P}} \qquad \Delta;\Gamma \vdash \bar{\mathtt{e}} : \bar{\mathtt{S}} \qquad \Delta \vdash \bar{\mathtt{S}} <: [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{U}}\end{array}}{\Delta;\Gamma \vdash \mathtt{e}_0.\mathtt{m}\mathtt{<}\bar{\mathtt{V}}\mathtt{>}(\bar{\mathtt{e}}) : [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}}$$

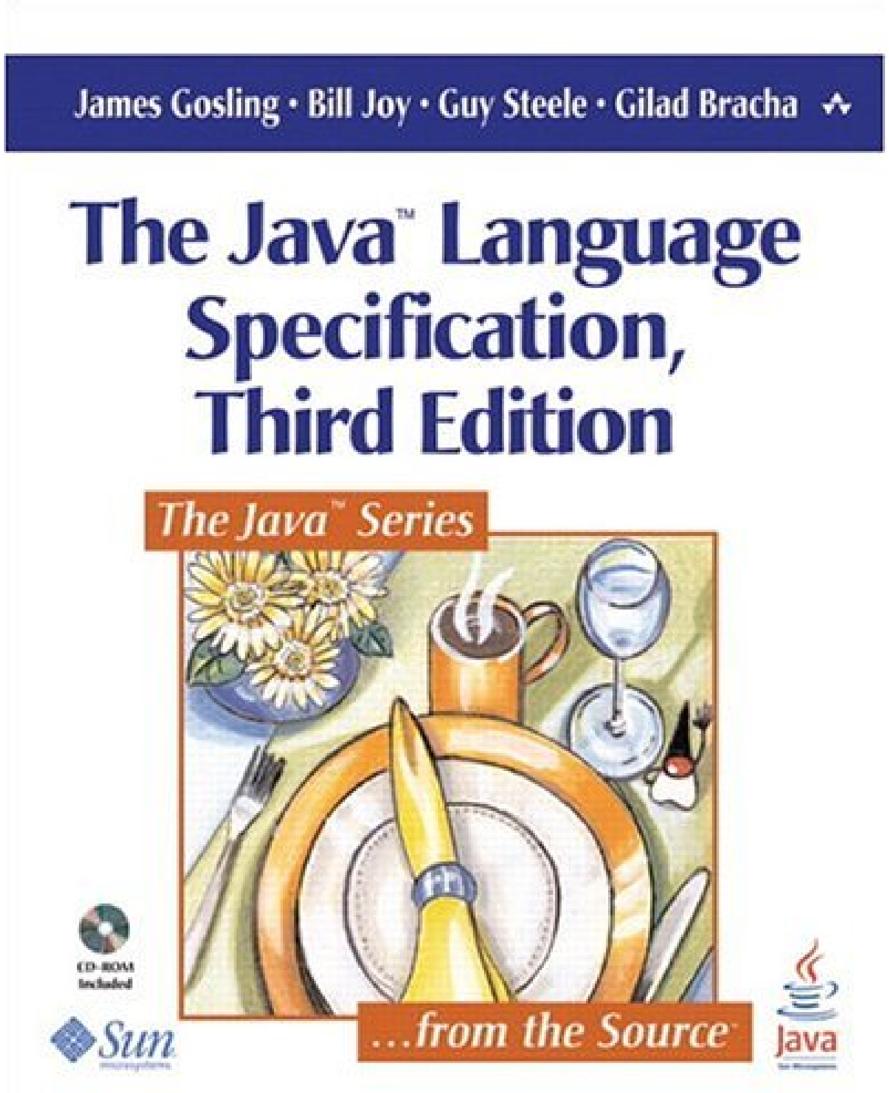
$$\frac{\Delta \vdash \mathtt{N}\ \mathrm{ok} \qquad \mathit{fields}(\mathtt{N}) = \bar{\mathtt{T}}\ \bar{\mathtt{f}} \qquad \Delta;\Gamma \vdash \bar{\mathtt{e}} : \bar{\mathtt{S}} \qquad \Delta \vdash \bar{\mathtt{S}} <: \bar{\mathtt{T}}}{\Delta;\Gamma \vdash \mathtt{new}\ \mathtt{N}(\bar{\mathtt{e}}) : \mathtt{N}}$$

$$\frac{\Delta;\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0 \qquad \Delta \vdash \mathit{bound}_\Delta(\mathtt{T}_0) <: \mathtt{N}}{\Delta;\Gamma \vdash (\mathtt{N})\mathtt{e}_0 : \mathtt{N}}$$

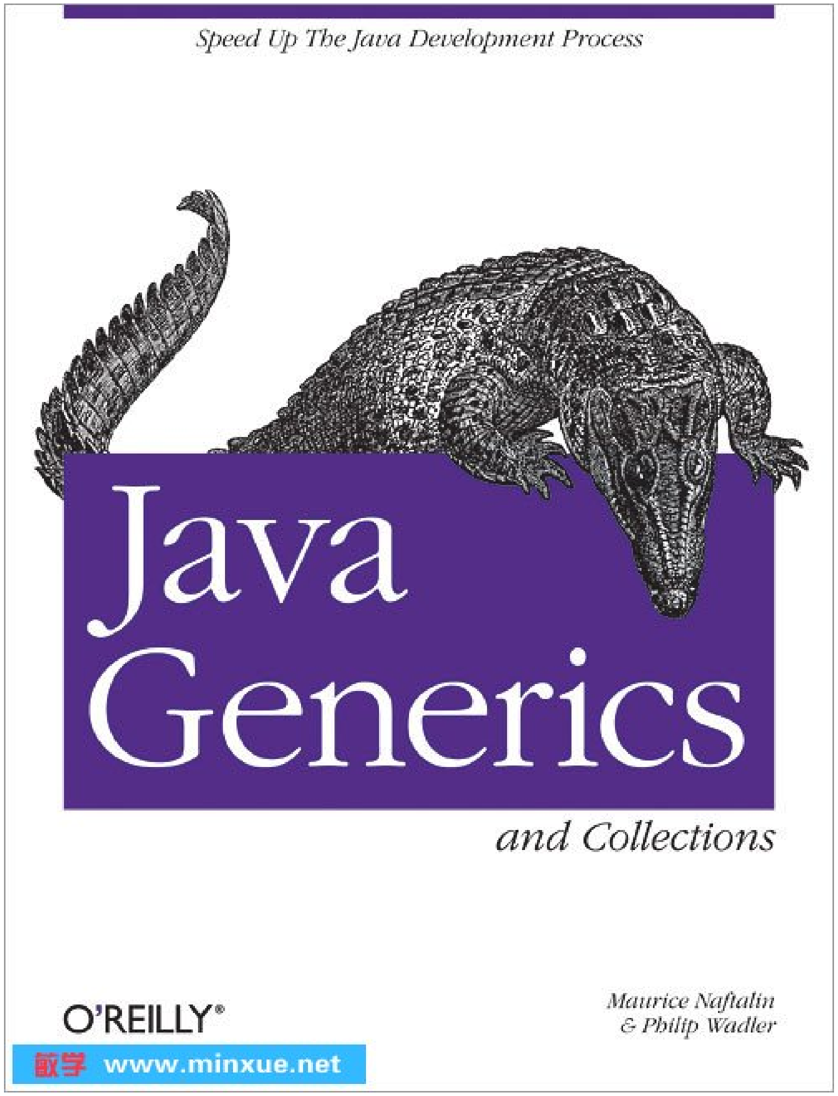
$$\frac{\begin{array}{c}\Delta;\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0 \qquad \Delta \vdash \mathtt{N}\ \mathrm{ok} \qquad \Delta \vdash \mathtt{N} <: \mathit{bound}_\Delta(\mathtt{T}_0) \\ \mathtt{N} = \mathtt{C}\mathtt{<}\bar{\mathtt{T}}\mathtt{>} \qquad \mathit{bound}_\Delta(\mathtt{T}_0) = \mathtt{D}\mathtt{<}\bar{\mathtt{U}}\mathtt{>} \qquad \mathit{dcast}(\mathtt{C}, \mathtt{D})\end{array}}{\Delta;\Gamma \vdash (\mathtt{N})\mathtt{e}_0 : \mathtt{N}}$$

$$\frac{\begin{array}{c}\Delta;\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0 \qquad \Delta \vdash \mathtt{N}\ \mathrm{ok} \qquad \mathtt{N} = \mathtt{C}\mathtt{<}\bar{\mathtt{T}}\mathtt{>} \qquad \mathit{bound}_\Delta(\mathtt{T}_0) = \mathtt{D}\mathtt{<}\bar{\mathtt{U}}\mathtt{>} \\ \mathtt{C} \not\trianglelefteq \mathtt{D} \qquad \mathtt{D} \not\trianglelefteq \mathtt{C} \qquad \mathit{stupid\ warning}\end{array}}{\Delta;\Gamma \vdash (\mathtt{N})\mathtt{e}_0 : \mathtt{N}}$$

Gosling, Joy, Steele, Bracha (2004) — Java 5



Naftalin and Wadler (2006)



Part III

Haskell: Type Classes

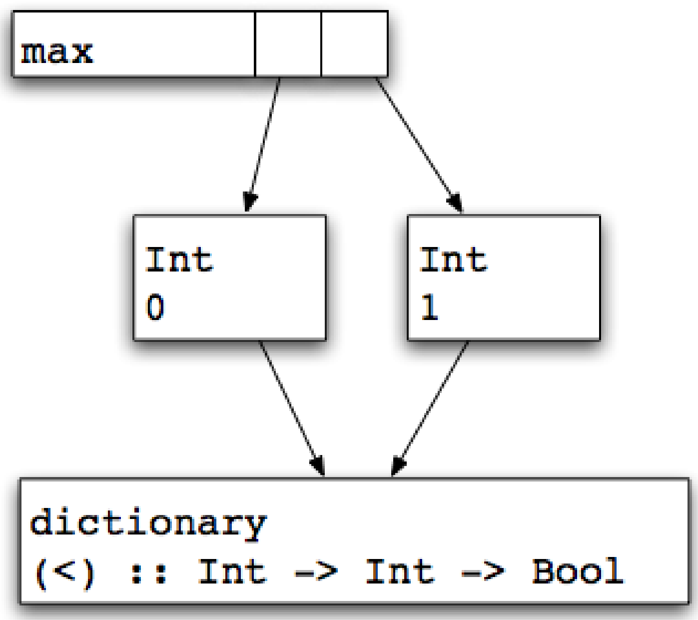
Type classes

```
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 | x < y       =  y
        | otherwise   =  x

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

maximum [0,1,2] == 2
maximum "abc" == 'c'
```

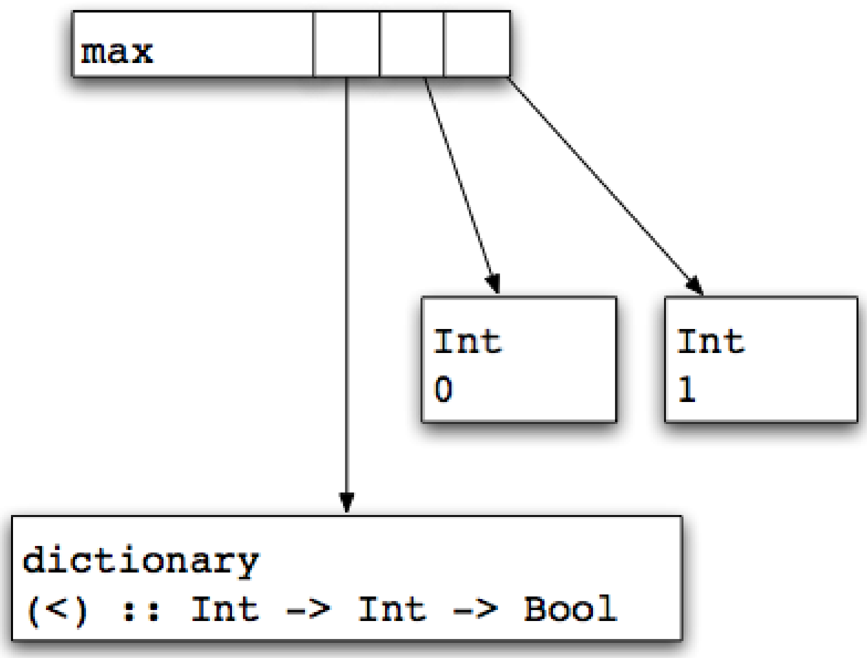
Translation

```
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 | less d x y  =  x
          | otherwise   =  y

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

maximum ordInt [0,1,2] == 2
maximum ordChar "abc" == 'c'
```

Object-oriented



Type classes



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

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) | less d x y  =  True
                     | less d y x  =  False
                     | otherwise   =  lt d xs ys

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

Maximum of a list, in Java

```
public static <T extends Comparable<T>>
  T maximum(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 maximum(ints) == 2;

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

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

Part IV

Three recent ideas

Idea I: Blame calculus

$$
\begin{array}{rcll}
v : A{\to}B \Rightarrow^{p} A'{\to}B' & \longrightarrow & \lambda x'{:}A'.\,(v\;(x' : A' \Rightarrow^{\overline{p}} A) : B \Rightarrow^{p} B') & \\
v : G \Rightarrow^{p} G & \longrightarrow & v & \text{if } G \neq \star{\to}\star \\
v : A \Rightarrow^{p} \star & \longrightarrow & v : A \Rightarrow^{p} G \Rightarrow \star & \text{if } A \prec G \text{ and } A \neq \star \\
v : G \Rightarrow \star \Rightarrow^{p} A & \longrightarrow & v : G \Rightarrow^{p} A & \text{if } G \prec A \\
v : G \Rightarrow \star \Rightarrow^{p} A & \longrightarrow & \texttt{blame}\; p & \text{if } G \not\prec A \\
(v : G \Rightarrow \star)\;\texttt{is}\; G & \longrightarrow & \texttt{true} & \\
(v : H \Rightarrow \star)\;\texttt{is}\; G & \longrightarrow & \texttt{false} & \text{if } G \neq H \\
E[\texttt{blame}\; p] & \longmapsto & \texttt{blame}\; p & \text{if } E \neq [\cdot]
\end{array}
$$

Idea II: Propositions as Sessions

$$\frac{}{w \leftrightarrow x \vdash w : A^{\perp}, x : A}\ \mathsf{Ax} \qquad \frac{P \vdash \Gamma, x : A \quad Q \vdash \Delta, x : A^{\perp}}{\nu x : A.(P \mid Q) \vdash \Gamma, \Delta}\ \mathsf{Cut}$$

$$\frac{P \vdash \Gamma, y : A \quad Q \vdash \Delta, x : B}{x[y].(P \mid Q) \vdash \Gamma, \Delta, x : A \otimes B}\ \otimes \qquad \frac{R \vdash \Theta, y : A, x : B}{x(y).R \vdash \Theta, x : A \parr B}\ \parr$$

$$\frac{P \vdash \Gamma, x : A}{x[\mathsf{inl}].P \vdash \Gamma, x : A \oplus B}\ \oplus_1 \qquad \frac{P \vdash \Gamma, x : B}{x[\mathsf{inr}].P \vdash \Gamma, x : A \oplus B}\ \oplus_2 \qquad \frac{Q \vdash \Delta, x : A \quad R \vdash \Delta, x : B}{x.\mathsf{case}(Q, R) \vdash \Delta, x : A \,\&\, B}\ \&$$

$$\frac{P \vdash ?\Gamma, y : A}{!x(y).P \vdash ?\Gamma, x : !A}\ ! \qquad \frac{Q \vdash \Delta, y : A}{?x[y].Q \vdash \Delta, x : ?A}\ ?$$

$$\frac{Q \vdash \Delta}{Q \vdash \Delta, x : ?A}\ \mathsf{Weaken} \qquad \frac{Q \vdash \Delta, x : ?A, x' : ?A}{Q\{x/x'\} \vdash \Delta, x : ?A}\ \mathsf{Contract}$$

Idea III: Object to Object

- Object vitiates parametricity

```
class Object {
  Bool eq(Object that) {...}
  String show() {...}
}
```

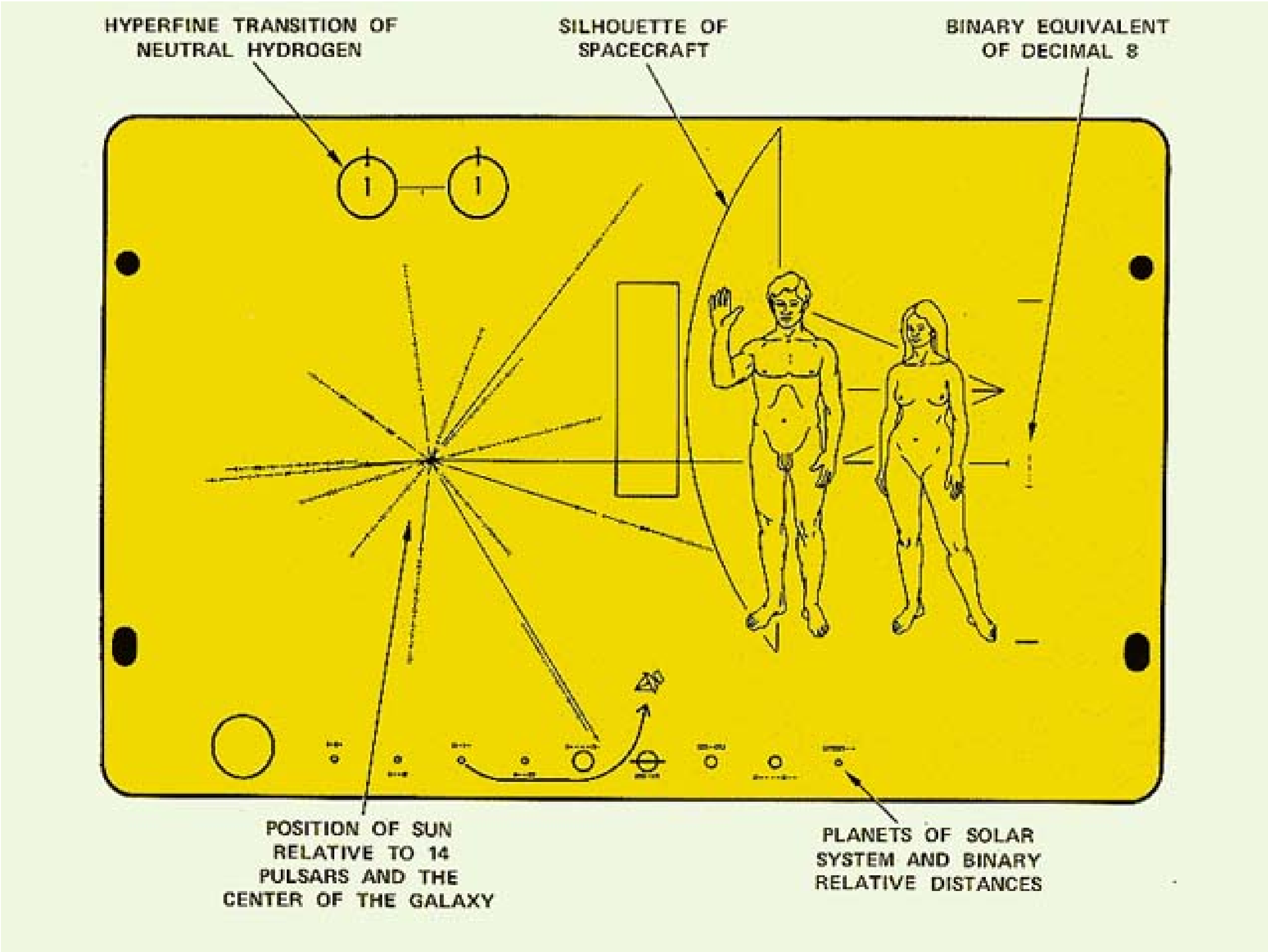
- Top preserves parametricty

```
class Object {
  // no methods!
}
```

Part V

Aliens

How to talk to aliens



Independence Day

A universal programming language?

Lambda is Omniversal