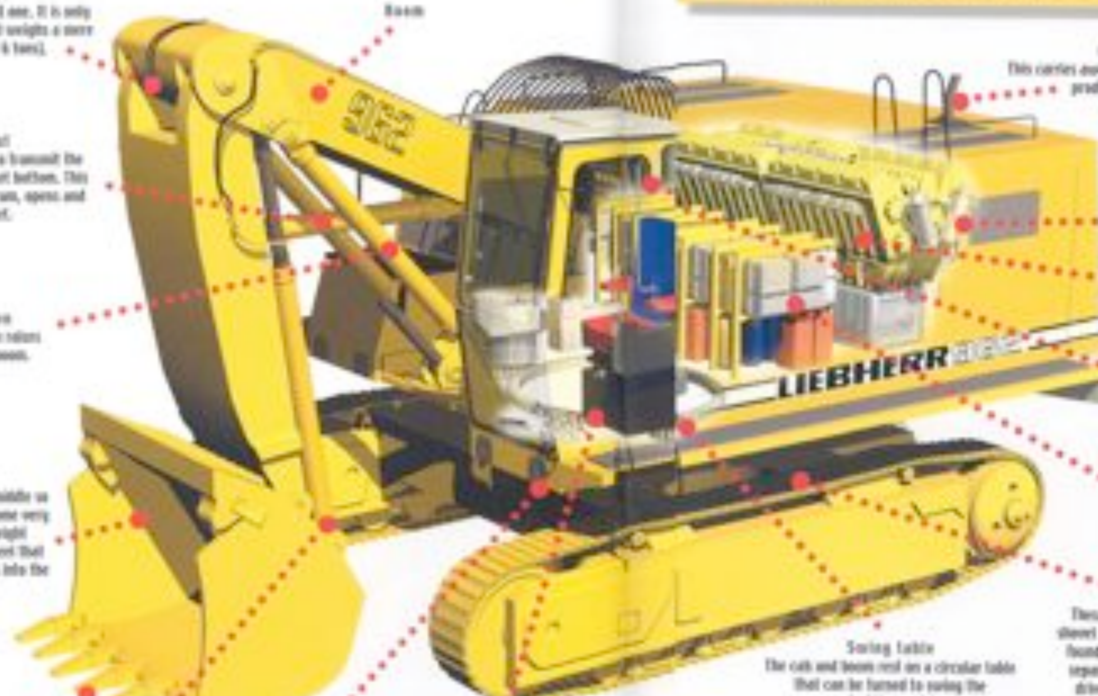
The Unreasonable Effectiveness of Logic

Philip Wadler
University of Edinburgh
wadler@inf.ed.ac.uk

Is computing a deep subject?

POWER SHOVEL



COMPUTER CONTROLS

Monster machines such as the power shovel need to be carefully controlled, or they could do a lot of damage. If they are overloaded or break down they are extremely expensive to repair. So the power shovel has built-in computer systems that automatically shut down the engine if there is a danger of overload. Sensors fitted around the shovel monitor engine performance, temperature and oil pressure. The computer gives a warning if the engine is not operating properly.

BIG DIGGER

A power shovel digs coal out of the walls of an open-cast coal mine. This monster machine has a huge bucket at the end of a long arm or boom – it carries up to 140 cubic metres (1,937 cubic feet) of coal. The boom stretches up the coal face to scrape out coal with the bucket. When the bucket is full, the driver swings the arm round and dumps the coal on to a waiting lorry. A power shovel works fast – it can fill a large lorry with 120 tonnes (118 tons) of coal in just two minutes. Power shovels are driven by petrol or diesel engines, or by electric motors.

The Marion 6360 power shovel has a boom length of 67 metres (220 feet) and a reach of 72 metres (236 feet). It weighs 1,100 tonnes (1,082 tons) and uses 20 electric motors to power the boom and bucket. It works in an open-cast coal mine near Percy in Illinois, USA.

Theoretical computer science is unnatural ...

... but is it unnatural like Ikebana?

... or is it unnatural like Judo?


WHANG!
BOOM!
CRASH!
BUMP!
BANG!
THWACK!
THUMP!

More than a coincidence?

second-order logic	polymorphism	Java
modal logic	monads	XML
classical logic	continuations	Links

Part I

A remarkable coincidence

Gerhard Gentzen (1909–1945)

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*

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

∨–*E*

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

∀–*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

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

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

Simplifying a proof

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

Simplifying a proof

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

$$\Downarrow$$

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

Simplifying a proof

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

$$\Downarrow$$

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

$$\Downarrow$$

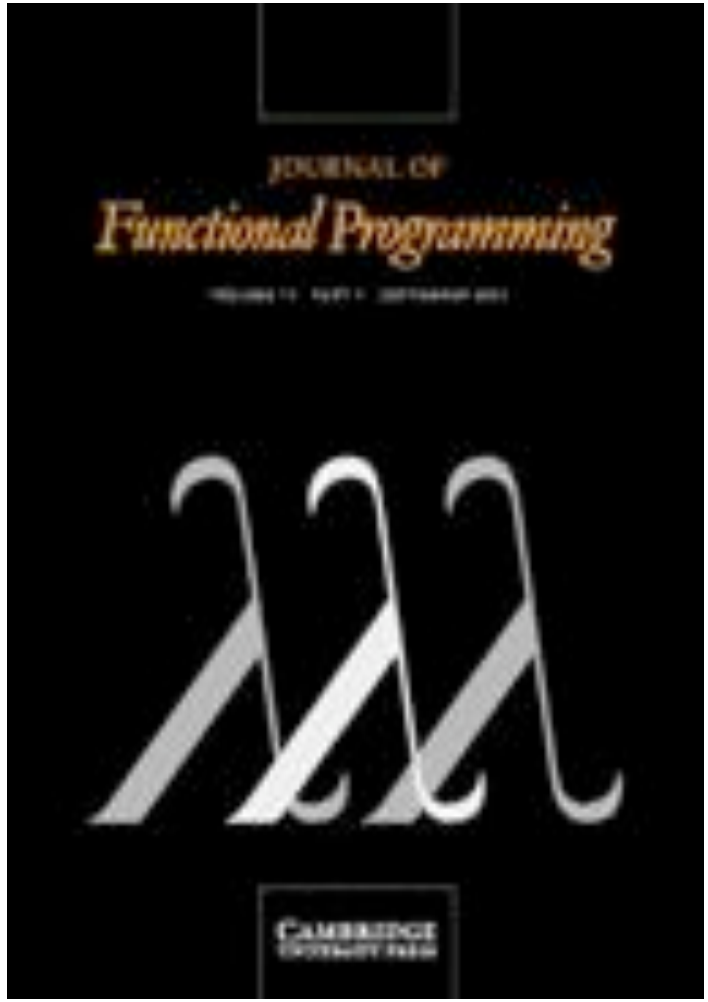
$$\dfrac{[A]^x \quad [B]^y}{A \,\&\, B}\,\&\text{-I}$$

Alonzo Church (1903–1995)

Alonzo Church (1932) — Lambda calculus

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

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

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

Simplifying a program

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

$$\Downarrow$$

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

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.

More than a coincidence?

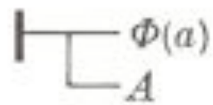
second-order logic	polymorphism	Java
modal logic	monads	XML
classical logic	continuations	Links

Part II

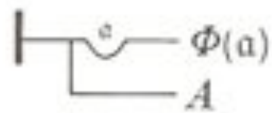
Second-order logic, Polymorphism, and Java

Gottlob Frege (1879) — Quantifiers (∀)

It is clear also that from



we can derive

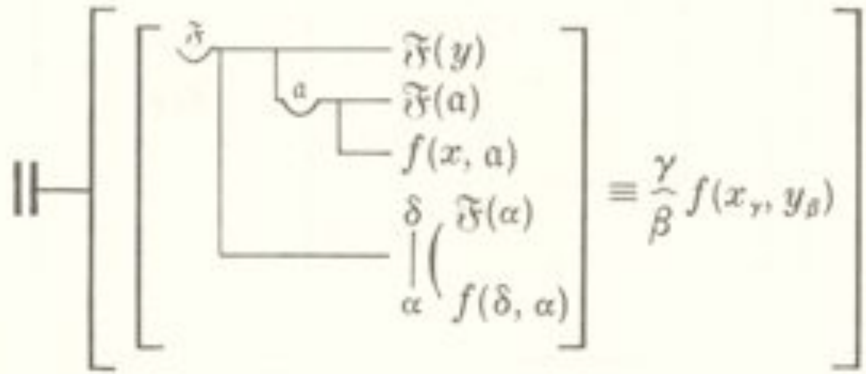


if A is an expression in which a does not occur and if a stands only in the argument places of Φ(a).[14] If $-\!\overset{\mathfrak{a}}{\smile}\!-\Phi(\mathfrak{a})$ is denied, we must be able to specify a meaning for *a* such that $\Phi(a)$ will be denied. If, therefore, $-\!\overset{\mathfrak{a}}{\smile}\!-\Phi(\mathfrak{a})$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

Gottlob Frege (1879) — Quantifiers (∀)

If from the proposition that 𝔡 *has property F, whatever* 𝔡 *may be, it can be inferred that every result of an application of the procedure f to* 𝔡 *has property F, then property F is hereditary in the f-sequence.*

§ 26.



(76).

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:

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

Jean-Yves Girard (1972) — Polymorphism

UNE EXTENSION DE L'INTERPRETATION DE GÖDEL A L'ANALYSE, ET SON APPLICATION A L'ELIMINATION DES COUPURES DANS L'ANALYSE ET LA THEORIE DES TYPES

Jean-Yves GIRARD

(8, Rue du Moulin d'Amboile, 94-Sucy en Brie, France)

Ce travail comprend (Ch. 1–5) une interprétation de l'Analyse, exprimée dans la logique intuitionniste, dans un système de fonctionnelles Y, décrit Ch. 1, et qui est une extension du système connu de Gödel [Gd]. En gros, le système est obtenu par l'adjonction de deux sortes de types (respectivement existentiels et universels, si les types construits avec → sont considérés comme implicationnels) et de quatre schémas de construction de fonctionelles correspondant à l'introduction et à l'élimination de chacun de ces types, ainsi que par la donnée des règles de calcul (réductions) correspondantes.

Robin Milner (1975) — Polymorphism

A Theory of Type Polymorphism in Programming

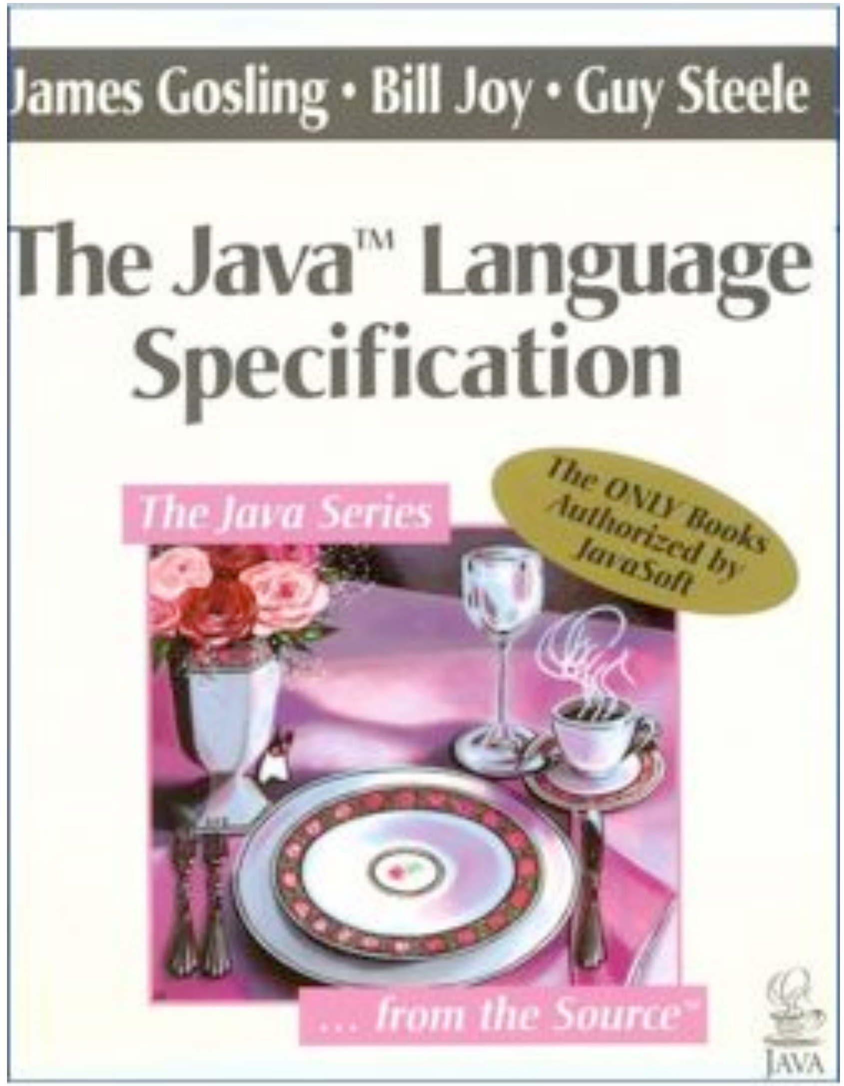
ROBIN MILNER

Computer Science Department, University of Edinburgh, Edinburgh, Scotland

Received October 10, 1977; revised April 19, 1978

The aim of this work is largely a practical one. A widely employed style of programming, particularly in structure-processing languages which impose no discipline of types, entails defining procedures which work well on objects of a wide variety. We present a formal type discipline for such polymorphic procedures in the context of a simple programming language, and a compile time type-checking algorithm $\mathscr{W}$ which enforces the discipline. A Semantic Soundness Theorem (based on a formal semantics for the language) states that well-type programs cannot "go wrong" and a Syntactic Soundness Theorem states that if $\mathscr{W}$ accepts a program then it is well typed. We also discuss extending these results to richer languages; a type-checking algorithm based on $\mathscr{W}$ is in fact already implemented and working, for the metalanguage ML in the Edinburgh LCF system.

Gosling, Joy, Steele (1996) — Java



Odersky and Wadler (1997) — Pizza

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);
```

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

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

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

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

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

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

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

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

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

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

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

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

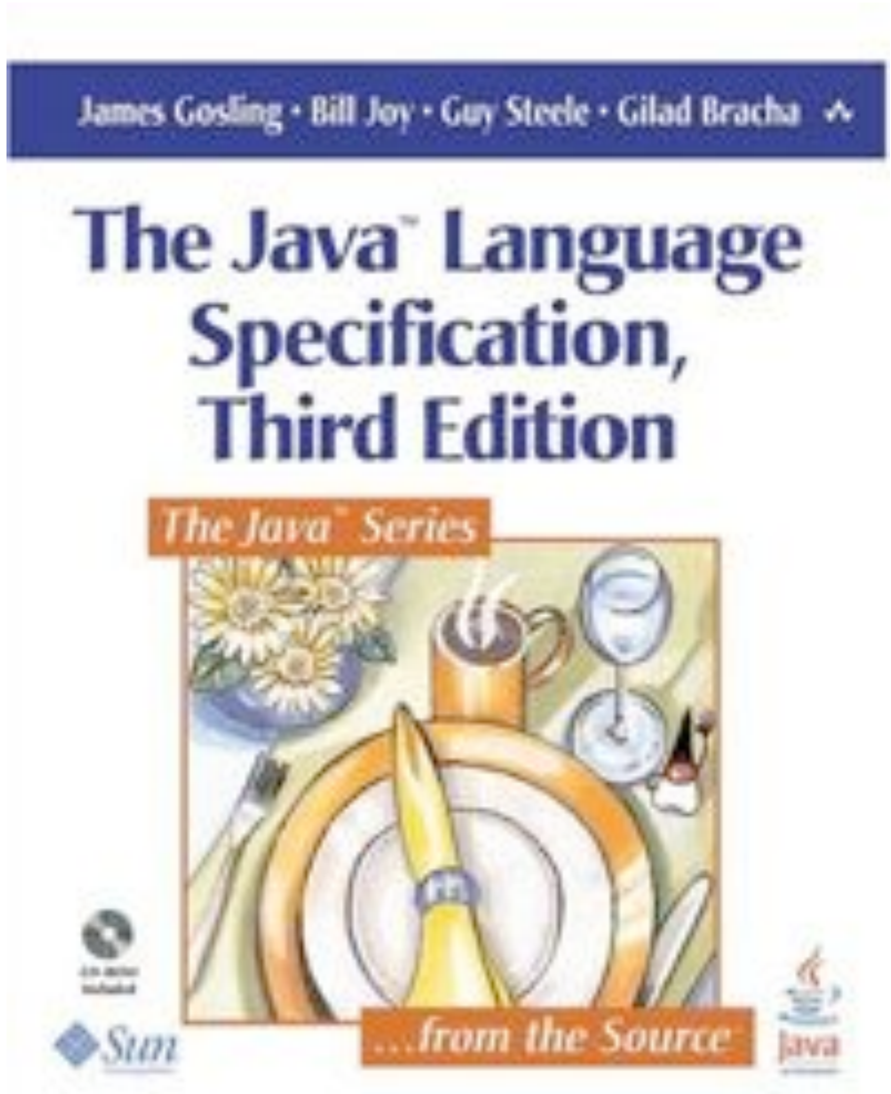
$$\frac{\Delta \vdash \mathtt{N}\ \mathrm{ok} \qquad \mathit{fields}(\mathtt{N}) = \bar{\mathtt{T}}\ \bar{\mathtt{f}} \qquad \Delta;\Gamma \vdash \bar{\mathtt{e}} : \bar{\mathtt{S}} \qquad \Delta \vdash \bar{\mathtt{S}} <: \bar{\mathtt{T}}}{\Delta;\Gamma \vdash \mathtt{new\ N}(\bar{\mathtt{e}}) : \mathtt{N}}$$

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

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

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

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



Part III

Modality, monads, and XML

Clarence Lewis (1918) — Modal Logic

Systems previously developed, except MacColl's, have only two truth-values, "true" and "false". The addition of the idea of impossibility gives us five truth-values, all of which are familiar logical ideas:

(1) p, "p is true".

(2) $-p$, "p is false".

(3) $\sim p$, "p is impossible".

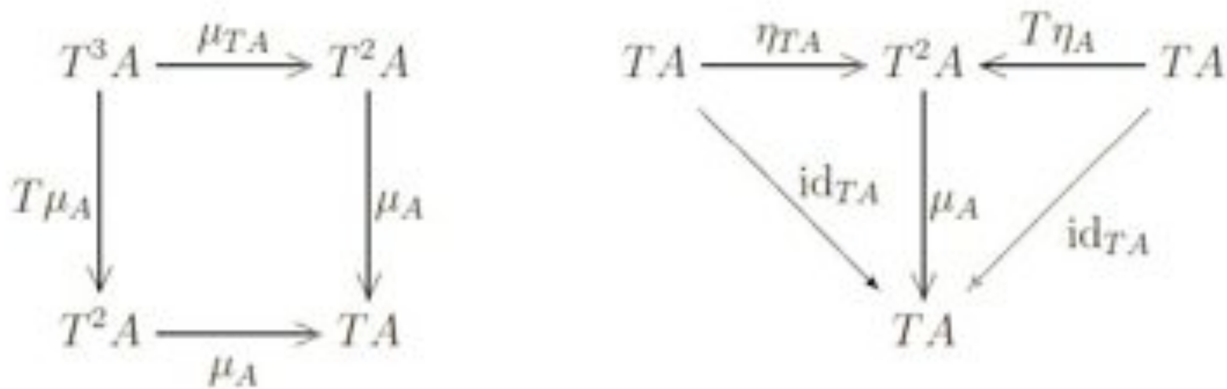
(4) $-\sim p$, "It is false that p is impossible"—i. e., "p is possible".

(5) $\sim - p$, "It is impossible that p be false"—i. e., "p is necessarily true".

Strictly, the last two should be written $-(\sim p)$ and $\sim(-p)$: the parentheses are regularly omitted for typographical reasons.

Eugenio Moggi (1988) — Monads

Definition 2.1
A **computational model** *is a monad* (T, η, μ) *over a category* $\mathcal{C}$, *i.e. a functor* $T{:}\,\mathcal{C} \to \mathcal{C}$ *and two natural transformations* $\eta{:}\,\mathrm{Id}_{\mathcal{C}} \stackrel{.}{\to} T$ *and* $\mu{:}\,T^2 \stackrel{.}{\to} T$ *s.t.*



which satisfies also an extra **equalizing requirement**: $\eta_A{:}\,A \to TA$ *is an equalizer of* $\eta_T A$ *and* $T(\eta_A)$, *i.e. for any* $f{:}\,B \to TA$ *s.t.* $f;\eta_{TA} = f;T(\eta_A)$ *there exists a unique* $m{:}\,B \to A$ *s.t.* $f = m;\eta_A$[3].

Philip Wadler (1990) — Comprehensions

2.2 Comprehensions

Many functional languages provide a form of *list comprehension* analogous to set comprehension. For example,

$$[(x, y) \mid x \leftarrow [1, 2], y \leftarrow [3, 4]] \;=\; [(1, 3), (1, 4), (2, 3), (2, 4)].$$

In general, a comprehension has the form $[\, t \mid q \,]$, where t is a term and q is a qualifier. We use the letters t, u, v to range over terms, and p, q, r to range over qualifiers. A qualifier is either empty, Λ; or a generator, $x \leftarrow u$, where x is a variable and u is a list-valued term; or a composition of qualifiers, (p, q). Comprehensions are defined by the following rules:

$$
\begin{array}{lllll}
(1) & [\, t \mid \Lambda \,] & = & \mathit{unit}\; t, \\
(2) & [\, t \mid x \leftarrow u \,] & = & \mathit{map}\,(\lambda x \rightarrow t)\; u, \\
(3) & [\, t \mid (p, q) \,] & = & \mathit{join}\,[\,[t|q] \mid p\,].
\end{array}
$$

Peter Buneman *et al* (1991) — Comprehensions

A more verbose version of this query can also be written in SQL

```
SELECT   Name = p.Name, Mgr = d.Mgr
FROM     Emp p, Dept d
WHERE    p.D# = d.D#
```

We can put a different interpretation on the syntax of this query. In SQL, the symbols p and d are simply aliases for the relation names Emp and Dept respectively.

interesting connections with what we shall develop. In our syntax this query is written:

```
{[Name = p.Name, Mgr = d.Mgr] |
     \p <- Emp,
     \d <- Dept,
     p.DNum = d.DNum}
```

The syntactic form $\{e \mid c_1, c_2, \ldots, c_n\}$ is a *comprehension*. It is an expression that denotes a collection – in

XQuery (2004) — FLWOR

W3C Working Draft

3.8 FLWOR Expressions

XQuery provides a feature called a FLWOR expression that supports iteration and binding of variables to intermediate results. This kind of expression is often useful for computing joins between two or more documents and for restructuring data. The name FLWOR, pronounced "flower", is suggested by the keywords `for`, `let`, `where`, `order by`, and `return`.

```
<authlist>
 {
   for $a in fn:distinct-values($bib/book/author)
   order by $a
   return
     <author>
        <name> {$a} </name>
        <books>
          {
            for $b in $bib/book[author = $a]
            order by $b/title
            return $b/title
          }
        </books>
     </author>
 }
</authlist>
```

XQuery (2004) — Formal Semantics

W3C Working Draft

4.8.2 For expression

Static Type Analysis

A single `for` expression is typed as follows: First $Type_1$ of the iteration expression $Expr_1$ is inferred. Then the prime type of $Type_1$, prime($Type_1$), is computed. This is a union over all item types in $Type_1$ (See **[8.4 Judgments for FLWOR and other expressions on sequences]**). With the variable component of the static environment statEnv extended with $VarRef_1$ as type prime($Type_1$), the type $Type_2$ of $Expr_2$ is inferred. Because the `for` expression iterates over the result of $Expr_1$, the final type of the iteration is $Type_2$ multiplied with the possible number of items in $Type_1$ (one, `?`, `*`, or `+`). This number is determined by the auxiliary type-function quantifier($Type_1$).

$$\frac{\begin{array}{c}\text{statEnv} \vdash Expr_1 : Type_1 \\ \text{statEnv} + \text{varType}(VarRef_1 : \text{prime}(Type_1)) \vdash Expr_2 : Type_2\end{array}}{\text{statEnv} \vdash \text{for } VarRef_1 \text{ in } Expr_1 \text{ return } Expr_2 : Type_2 \cdot \text{quantifier}(Type_1)}$$

Part IV

Classical logic, continuations, and the Web

Andrei Kolmogorov (1925)

To an elementary formula $\mathfrak{S}$ there corresponds in pseudomathematics the formula $\mathfrak{S}^*$, which expresses the double negation of $\mathfrak{S}$:

$$\mathfrak{S}^* \equiv \overline{\overline{\mathfrak{S}}}. \tag{48}$$

In what follows we shall, for convenience, denote the double negation of $\mathfrak{S}$ by $n\mathfrak{S}$.

To the formula of the nth order $F(\mathfrak{S}_1, \mathfrak{S}_2, \ldots, \mathfrak{S}_k)$, where $\mathfrak{S}_1, \mathfrak{S}_2, \ldots, \mathfrak{S}_k$ are formulas of the $(n-1)$th order at most, there corresponds in pseudomathematics the formula $F(\mathfrak{S}_1, \mathfrak{S}_2, \ldots, \mathfrak{S}_k)^*$ such that

$$F(\mathfrak{S}_1, \mathfrak{S}_2, \ldots, \mathfrak{S}_k)^* \equiv nF(\mathfrak{S}_1^*, \mathfrak{S}_2^*, \ldots, \mathfrak{S}_k^*), \tag{49}$$

$\mathfrak{S}_1^*, \mathfrak{S}_2^*, \ldots, \mathfrak{S}_k^*$ being regarded as already determined. For example, to the formula

$$a = b \rightarrow \{A(a) \rightarrow B(a)\}$$

there corresponds in pseudomathematics the formula

$$n[n(a = b) \rightarrow n\{nA(a) \rightarrow nB(a)\}].$$

Gordon Plotkin (1975)

We begin with a simulation of call-by-value by call-by-name. Given a call-by-value language with its Constapply_V, Eval_V and λ_V, we consider the call-by-name language whose variables are those of the given language together with three others, $\varkappa$, α and β say, and whose list of variables for the substitution prefix is that of the given language. Its Constapply will be given in a little while. First the term simulation map $M \mapsto \overline{M}$ sending terms in the call-by-value language to the call-by-name language is given by the recursive definition:

$$\bar{a} = \lambda\varkappa\,(\varkappa a)$$

$$\bar{x} = \lambda\varkappa\,(\varkappa x)$$

$$\overline{\lambda x M} = \lambda\varkappa\,(\varkappa\,(\lambda x \overline{M}))$$

$$\overline{MN} = \lambda\varkappa\,(\overline{M}\,(\lambda\alpha\overline{N}\,(\lambda\beta\alpha\beta\varkappa))).$$

Constapply_N is given by:

$$\text{Constapply}_N(a, b) = \overline{\text{Constapply}_V(a, b)}$$

Philip Wadler (2000)

$(\beta\&)$	$\langle V, W\rangle \bullet \mathrm{fst}[K]$	$\longrightarrow_v$	$V \bullet K$
$(\beta\&)$	$\langle V, W\rangle \bullet \mathrm{snd}[L]$	$\longrightarrow_v$	$W \bullet L$
$(\beta\vee)$	$\langle V\rangle\mathrm{inl} \bullet [K, L]$	$\longrightarrow_v$	$V \bullet K$
$(\beta\vee)$	$\langle W\rangle\mathrm{inr} \bullet [K, L]$	$\longrightarrow_v$	$W \bullet L$
$(\beta\neg)$	$[K]\mathrm{not} \bullet \mathrm{not}\langle M\rangle$	$\longrightarrow_v$	$M \bullet K$
$(\beta\supset)$	$\lambda x.\, N \bullet V \mathbin{@} L$	$\longrightarrow_v$	$V \bullet x.(N \bullet L)$
$(\beta\mathrm{L})$	$V \bullet x.(S)$	$\longrightarrow_v$	$S\{V/x\}$
$(\beta\mathrm{R})$	$(S).\alpha \bullet K$	$\longrightarrow_v$	$S\{K/\alpha\}$

$(\beta\&)$	$\langle M, N\rangle \bullet \mathrm{fst}[P]$	$\longrightarrow_n$	$M \bullet P$
$(\beta\&)$	$\langle M, N\rangle \bullet \mathrm{snd}[Q]$	$\longrightarrow_n$	$N \bullet Q$
$(\beta\vee)$	$\langle M\rangle\mathrm{inl} \bullet [P, Q]$	$\longrightarrow_n$	$M \bullet P$
$(\beta\vee)$	$\langle N\rangle\mathrm{inr} \bullet [P, Q]$	$\longrightarrow_n$	$N \bullet Q$
$(\beta\neg)$	$[K]\mathrm{not} \bullet \mathrm{not}\langle M\rangle$	$\longrightarrow_n$	$M \bullet K$
$(\beta\supset)$	$\lambda x.\, N \bullet M \mathbin{@} Q$	$\longrightarrow_n$	$M \bullet x.(N \bullet Q)$
$(\beta\mathrm{L})$	$M \bullet x.(S)$	$\longrightarrow_n$	$S\{M/x\}$
$(\beta\mathrm{R})$	$(S).\alpha \bullet P$	$\longrightarrow_n$	$S\{P/\alpha\}$

Philip Wadler (2000)

$$\frac{}{x : A \to \mid x : A}\,\text{IdR} \qquad \frac{}{\alpha : A \mid \to \alpha : A}\,\text{IdL}$$

$$\frac{\Gamma \to \Theta \mid M : A \qquad \Gamma \to \Theta \mid N : B}{\Gamma \to \Theta \mid \langle M, N \rangle : A \,\&\, B}\,\&\text{R} \qquad \frac{K : A \mid \Gamma \to \Theta}{\text{fst}[K] : A \,\&\, B \mid \Gamma \to \Theta} \qquad \frac{L : B \mid \Gamma \to \Theta}{\text{snd}[L] : A \,\&\, B \mid \Gamma \to \Theta}\,\&\text{L}$$

$$\frac{\Gamma \to \Theta \mid M : A}{\Gamma \to \Theta \mid \langle M \rangle\text{inl} : A \vee B} \qquad \frac{\Gamma \to \Theta \mid N : B}{\Gamma \to \Theta \mid \langle N \rangle\text{inr} : A \vee B}\,\vee\text{R} \qquad \frac{K : A \mid \Gamma \to \Theta \qquad L : B \mid \Gamma \to \Theta}{[K, L] : A \vee B \mid \Gamma \to \Theta}\,\vee\text{L}$$

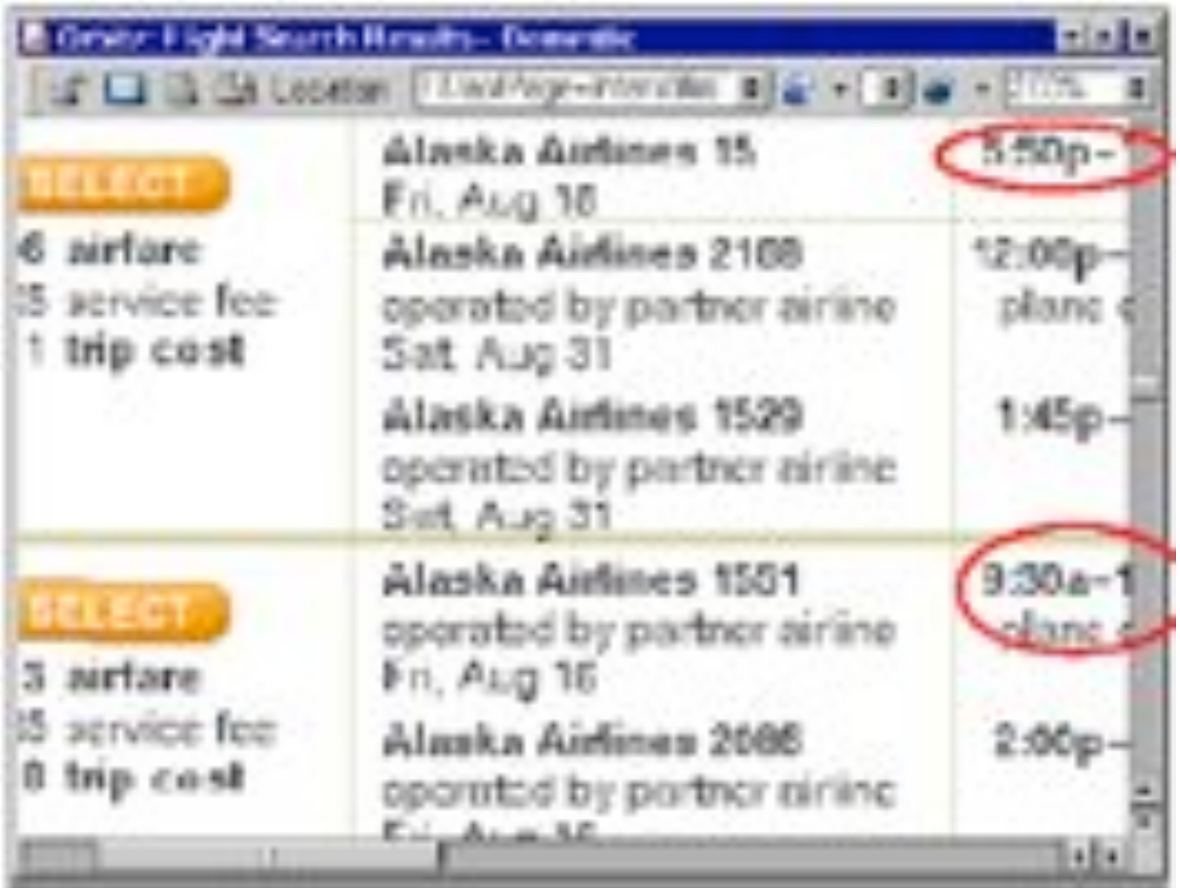
$$\frac{K : A \mid \Gamma \to \Theta}{\Gamma \to \Theta \mid [K]\text{not} : \neg A}\,\neg\text{R} \qquad \frac{\Gamma \to \Theta \mid M : A}{\text{not}\langle M \rangle : \neg A \mid \Gamma \to \Theta}\,\neg\text{L}$$

$$\frac{x : A, \Gamma \to \Theta \mid N : B}{\Gamma \to \Theta \mid \lambda x.N : A \supset B}\,\supset\text{R} \qquad \frac{\Gamma \to \Theta \mid M : A \qquad L : B \mid \Delta \to \Lambda}{M \mathbin{@} L : A \supset B \mid \Gamma, \Delta \to \Theta, \Lambda}\,\supset\text{L}$$

$$\frac{\Gamma \mid S \mapsto \Theta, \alpha : A}{\Gamma \to \Theta \mid (S).\alpha : A}\,\text{RI} \qquad \frac{x : A, \Gamma \mid S \mapsto \Theta}{x.(S) : A \mid \Gamma \to \Theta}\,\text{LI}$$

$$\frac{\Gamma \to \Theta \mid M : A \qquad K : A \mid \Delta \to \Lambda}{\Gamma, \Delta \mid M \bullet K \mapsto \Theta, \Lambda}\,\text{Cut}$$

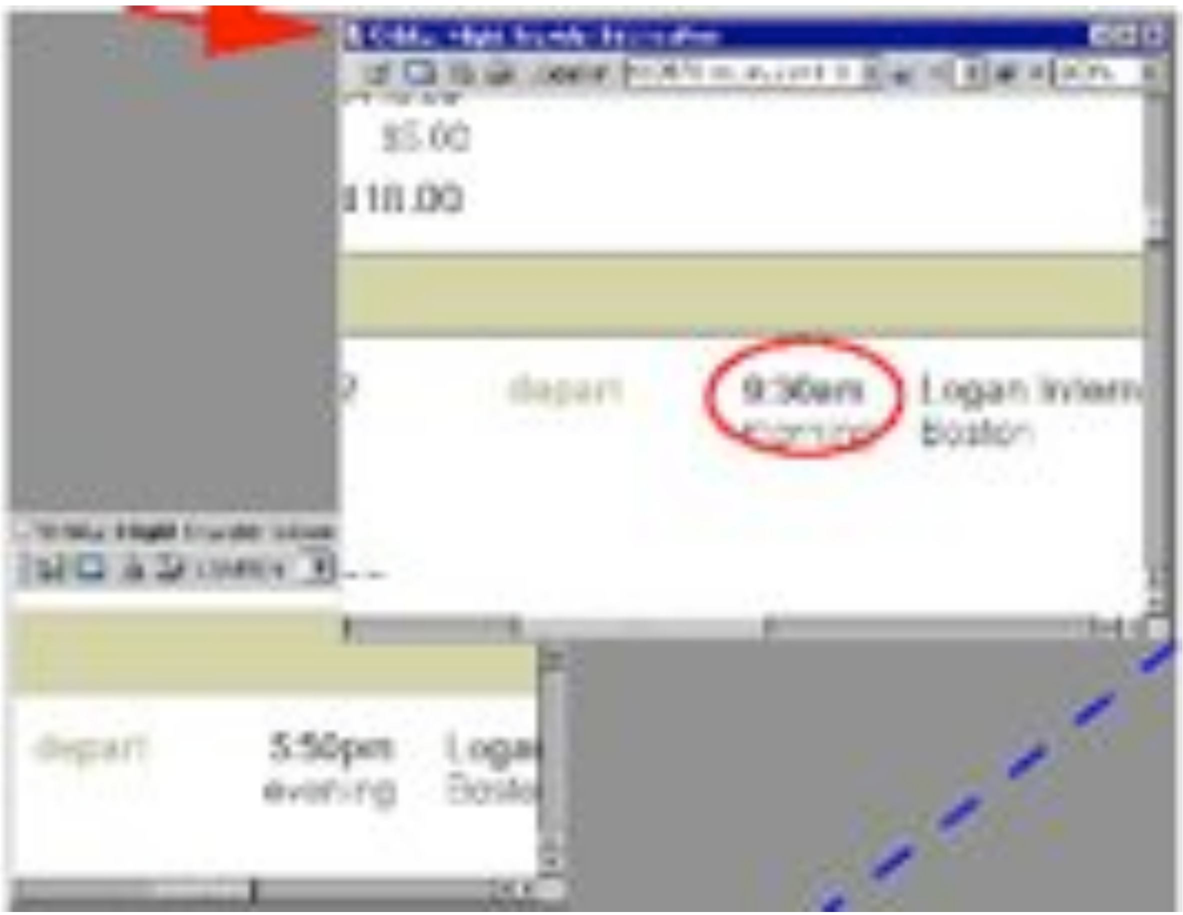
Orbitz: Two flights



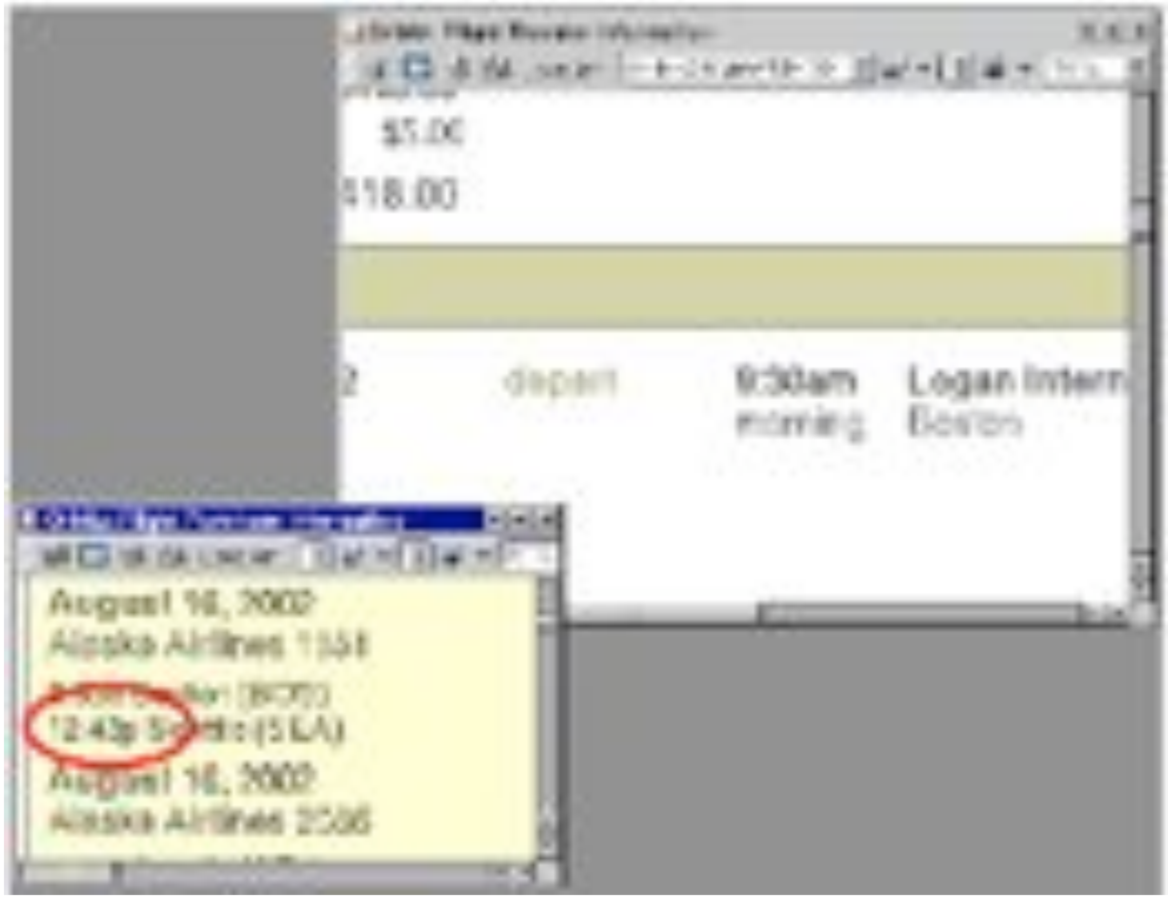
Graunke, Findler, Krishnamurthi, Felleisen (ESOP 2003)

Orbitz: Clone and submit first

Orbitz: Submit second



Orbitz: Select first – problem!

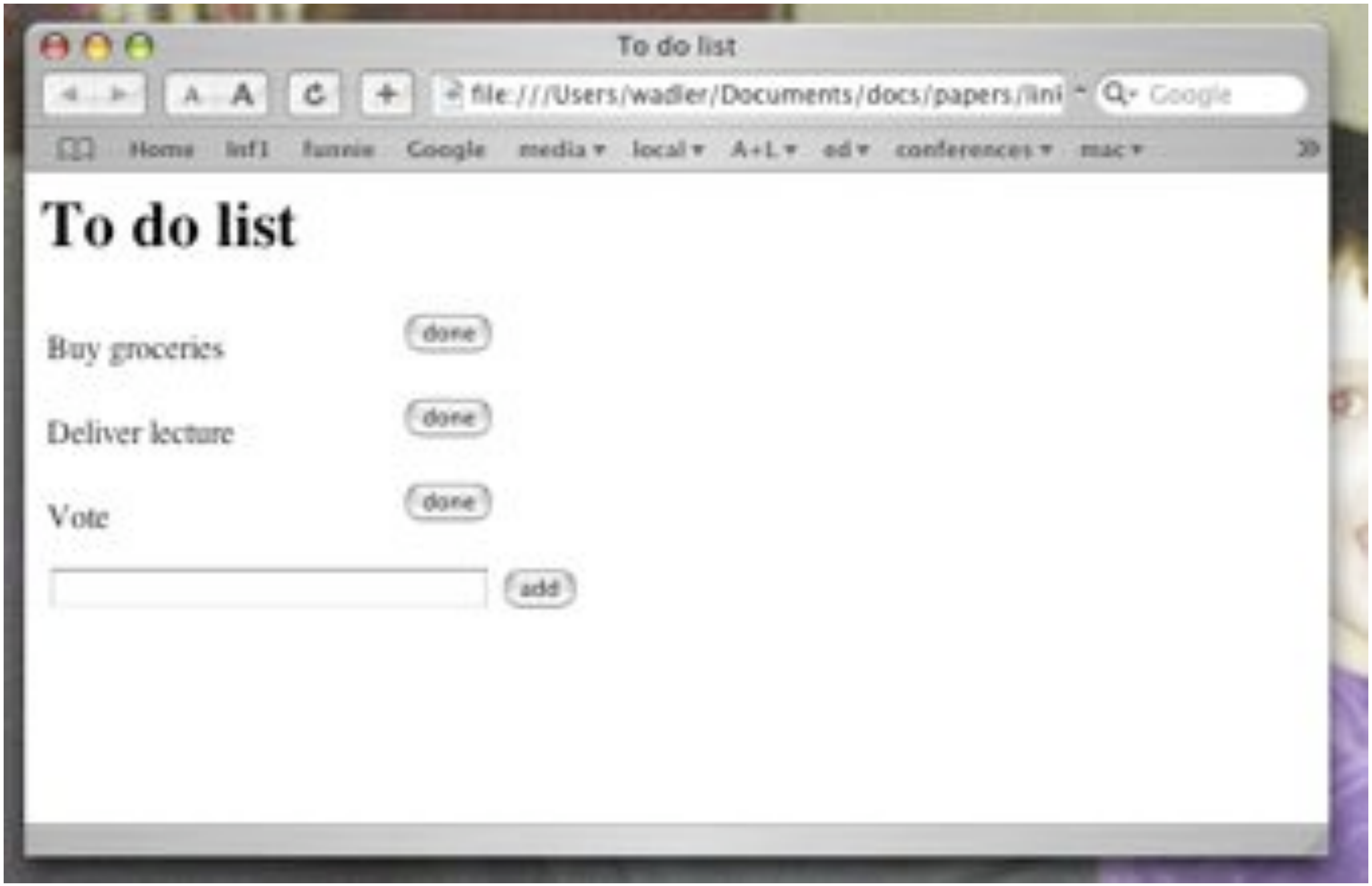


Burstall, MacQueen, and Sannella (1980) — Hope

Burstall, MacQueen, and Sannella (1980) — Hope

Wadler and Yallop (2005) — Links

```
main() { todo([]) }
todo(items) {
  <html><body>
    <h1>Items to do</h1>
    <table>{
      for item in items return
        <tr>
          <td>{item}</td>
          <td>
            <form l:action="{todo(items\\[item])}">
              <input type="submit" value="done"/>
            </form>
          </td>
        </tr>
    }</table>
    <form l:action="{todo(items++[new])}">
      <input l:name="{new}" type="text" size="40">
      <input type="submit" value="add"/>
    </form>
  </body></html>
}
```

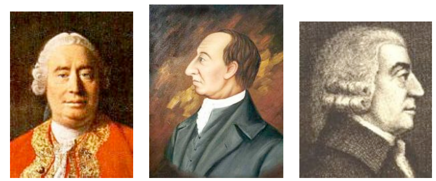

To do list
file:///Users/wadler/Documents/docs/papers/lin
Google
Home
inf1
funnie
Google
media
local
A+L
ed
conferences
mac
To do list
Buy groceries
done
Deliver lecture
done
Vote
done
add

Part V

Conclusions

Kinds of coincidence

Historical confluence of great minds — Hume, Hutton, Smith



Geographical shape of continents

Astronomical size of sun and moon from earth

More than a coincidence?

second-order logic	polymorphism	Java
modal logic	monads	XML
classical logic	continuations	Links

More than a coincidence?

second-order logic	polymorphism	Java
	Milner	
modal logic	monads	XML
	Moggi,Buneman	
classical logic	continuations	Links
	Plotkin	

Scottish Programming Language Seminar

Dec 2004 University of Glasgow

Mar 2005 University of Edinburgh

Jun 2005 Heriot-Watt University

Sep 2005 *University of St Andrews?*

Special thanks to

My colleagues for their ideas

Martina Sharp, Avaya Labs, and Diana Sisu, Informatics, for scanning

Adam and Leora for their books

Catherine for the tie

You for listening