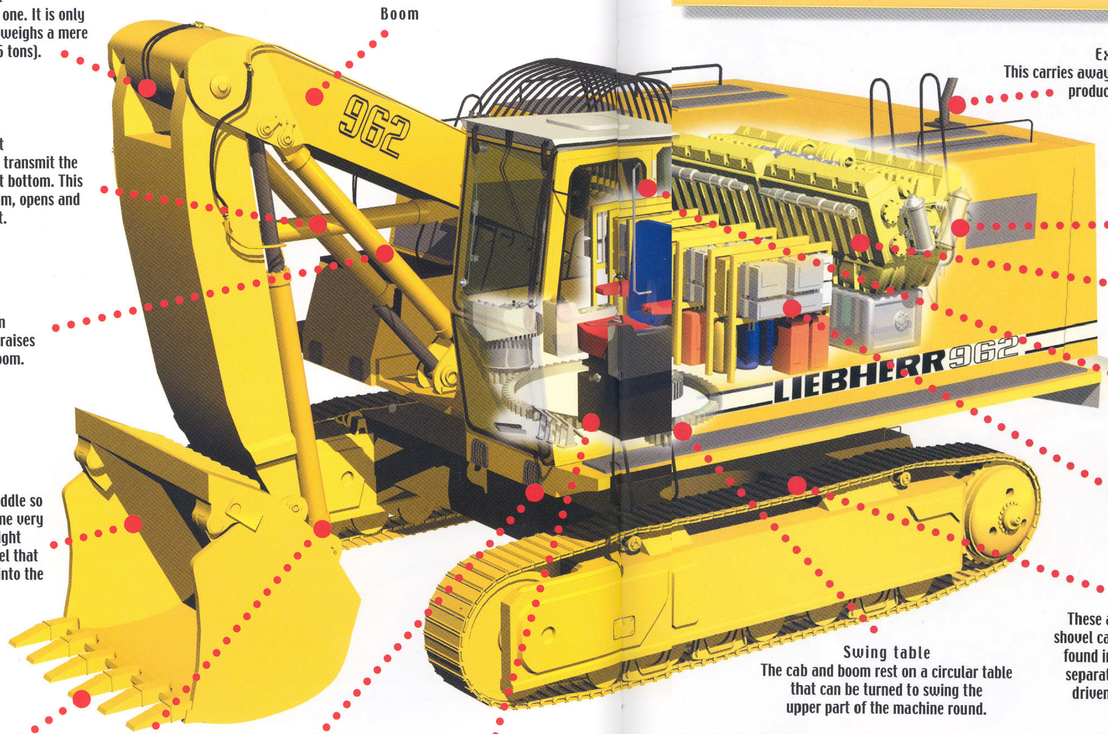
The Unreasonable Effectiveness of Logic

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

Is computing a deep subject?

POWER SHOVEL



Small shovel
This power shovel is a small one. It is only 13 metres (43 feet) long and weighs a mere 6 tonnes (just under 6 tons).

Boom

Open and shut
This ram uses oil pressure to transmit the force of a piston to the bucket bottom. This system, called a hydraulic ram, opens and closes the bucket.

Up and down
This hydraulic ram raises and lowers the boom.

Bucket
The bucket is hinged in the middle so it can open to drop a load. Some very big mining shovels have eight buckets fitted to a large wheel that revolves as the machine cuts into the coal-face.

Replaceable teeth
The teeth on the bucket are designed to sharpen themselves as they cut into the coal-face. They can be replaced when they eventually wear out.

Bucket hinge
The bucket opens and shuts here.

Headlight
Powerful headlights shine on the coal-face for working at night.

Swing motor
The swing table is turned by an electric or hydraulic motor. This swings the boom round for unloading.

Exhaust pipe
This carries away the waste gases and fumes produced by the engine.

Air filter
This filters (or separates) out the dust in the air, ensuring that only clean air goes to the engine.

Engines
The shovel has two powerful diesel engines. If one engine breaks down, the other takes over.

Driver's cab
The cab is 6 metres (18 feet) from the ground. It is sound- and vibration-proofed. The operator pulls levers to move the boom, and to open and close the bucket.

Oil tank
This holds the oil used in the hydraulic systems.

Crawler tracks
These are driven by the diesel engine. The shovel can move around safely on the soft soil found in open-cast mines. The tracks work separately; to turn the shovel, one track is driven forward while the other is driven backwards or kept still.

Swing table
The cab and boom rest on a circular table that can be turned to swing the upper part of the machine round.

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

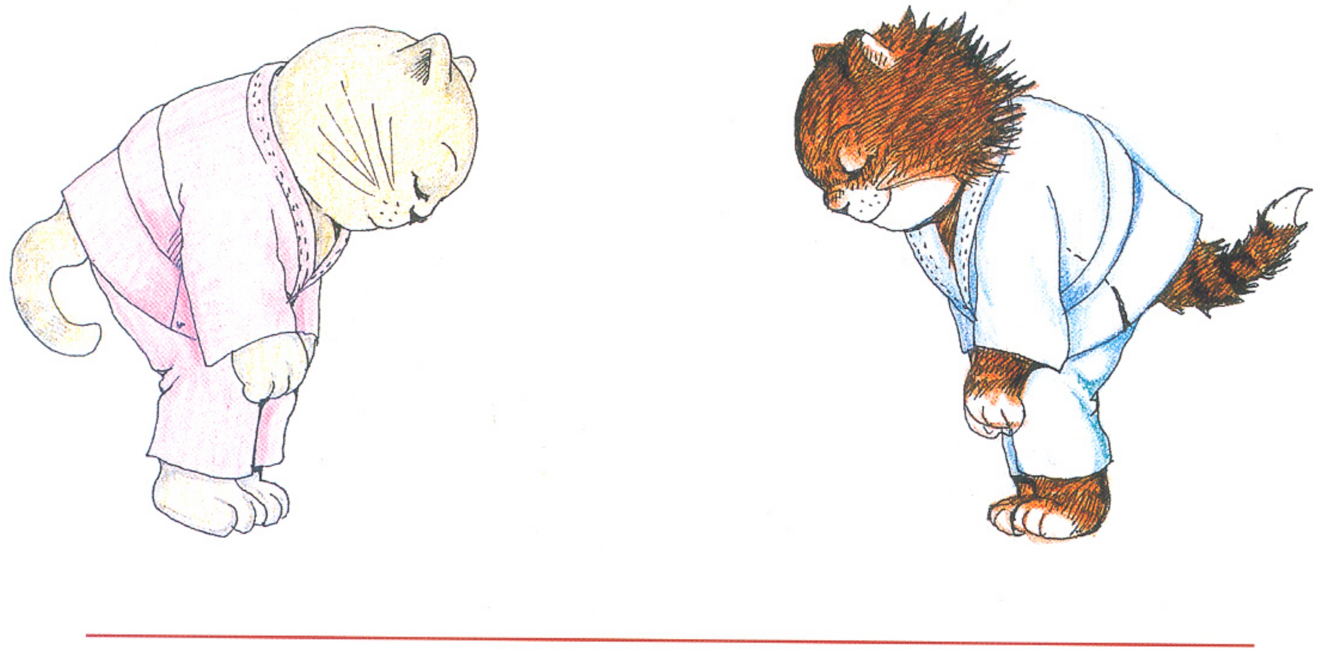
A power shovel digs coal out of the walls of an open-cast coal mine. This monster machine has a huge bucket at the end of a long arm or boom – it carries up to 140 cubic metres (1,507 cubic feet) of coal. The boom stretches up the coal face to scrape out coal with the bucket. When the bucket is full, 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!
BUMP!
CRASH!
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

$$
\begin{array}{cccc}
\&\text{–}I & \&\text{–}E & \vee\text{–}I & \vee\text{–}E \\[1ex]
\dfrac{\mathfrak{A} \quad \mathfrak{B}}{\mathfrak{A} \,\&\, \mathfrak{B}} &
\dfrac{\mathfrak{A} \,\&\, \mathfrak{B}}{\mathfrak{A}} \quad \dfrac{\mathfrak{A} \,\&\, \mathfrak{B}}{\mathfrak{B}} &
\dfrac{\mathfrak{A}}{\mathfrak{A} \vee \mathfrak{B}} \quad \dfrac{\mathfrak{B}}{\mathfrak{A} \vee \mathfrak{B}} &
\dfrac{\mathfrak{A} \vee \mathfrak{B} \quad \overset{[\mathfrak{A}]}{\mathfrak{C}} \quad \overset{[\mathfrak{B}]}{\mathfrak{C}}}{\mathfrak{C}} \\[4ex]
\forall\text{–}I & \forall\text{–}E & \exists\text{–}I & \exists\text{–}E \\[1ex]
\dfrac{\mathfrak{F}a}{\forall \mathfrak{x}\, \mathfrak{F}\mathfrak{x}} &
\dfrac{\forall \mathfrak{x}\, \mathfrak{F}\mathfrak{x}}{\mathfrak{F}a} &
\dfrac{\mathfrak{F}a}{\exists \mathfrak{x}\, \mathfrak{F}\mathfrak{x}} &
\dfrac{\exists \mathfrak{x}\, \mathfrak{F}\mathfrak{x} \quad \overset{[\mathfrak{F}a]}{\mathfrak{C}}}{\mathfrak{C}} \\[4ex]
\supset\text{–}I & \supset\text{–}E & \neg\text{–}I & \neg\text{–}E \\[1ex]
\dfrac{\overset{[\mathfrak{A}]}{\mathfrak{B}}}{\mathfrak{A} \supset \mathfrak{B}} &
\dfrac{\mathfrak{A} \quad \mathfrak{A} \supset \mathfrak{B}}{\mathfrak{B}} &
\dfrac{\overset{[\mathfrak{A}]}{\curlywedge}}{\neg \mathfrak{A}} &
\dfrac{\mathfrak{A} \quad \neg \mathfrak{A}}{\curlywedge} \quad \dfrac{\curlywedge}{\mathfrak{D}} \;.
\end{array}
$$

Gerhard Gentzen (1935) — Natural Deduction

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

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

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

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

$$\Downarrow$$

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

Simplifying a proof

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

$$\Downarrow$$

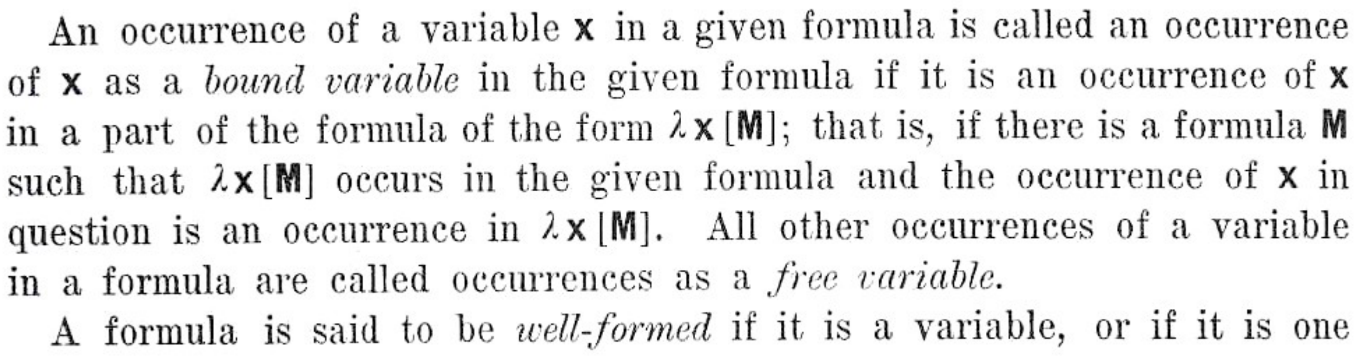
$$\dfrac{\dfrac{\dfrac{[B]^y \quad [A]^x}{B \mathbin{\&} A}\,\&\text{-I}}{A}\,\&\text{-E}_1 \qquad \dfrac{\dfrac{[B]^y \quad [A]^x}{B \mathbin{\&} A}\,\&\text{-I}}{B}\,\&\text{-E}_0}{A \mathbin{\&} B}\,\&\text{-I}$$

$$\Downarrow$$

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

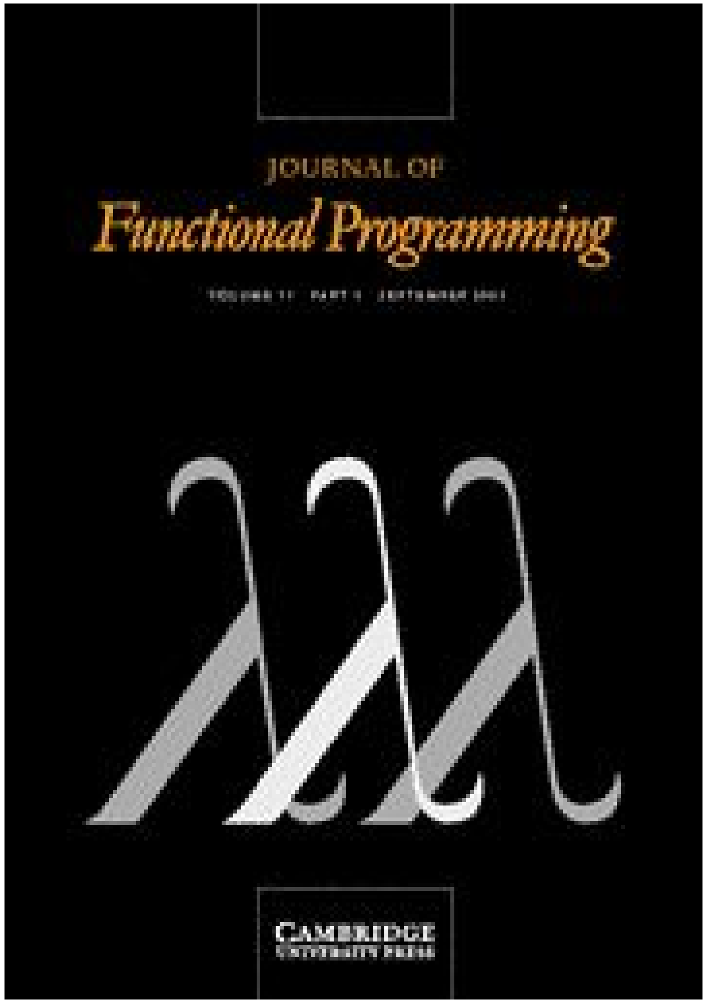
Alonzo Church (1903–1995)

Alonzo Church (1932) — Lambda calculus



An occurrence of a variable $\mathbf{x}$ 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

VOLUME 11 · PART 5 · SEPTEMBER 2001

CAMBRIDGE UNIVERSITY PRESS

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

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

$$\Downarrow$$

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

Simplifying a program

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

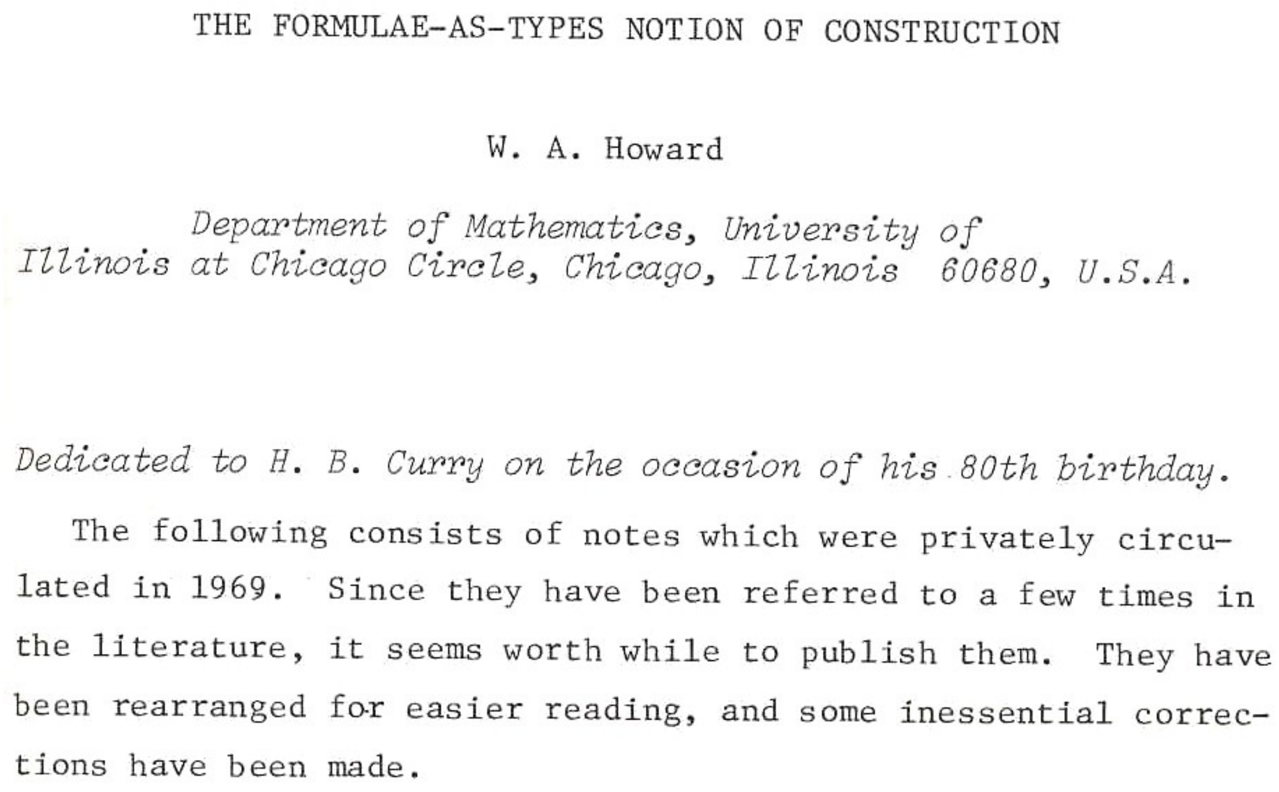
$$\Downarrow$$

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

$$\Downarrow$$

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

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?

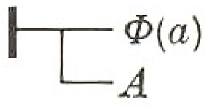
second-order logic	polymorphism	Java
modal logic	monads	XML
classical logic	continuations	Links

Part II

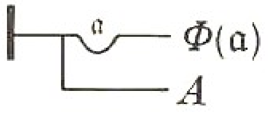
Second-order logic, Polymorphism, and Java

Gottlob Frege (1879) — Quantifiers (∀)

It is clear also that from

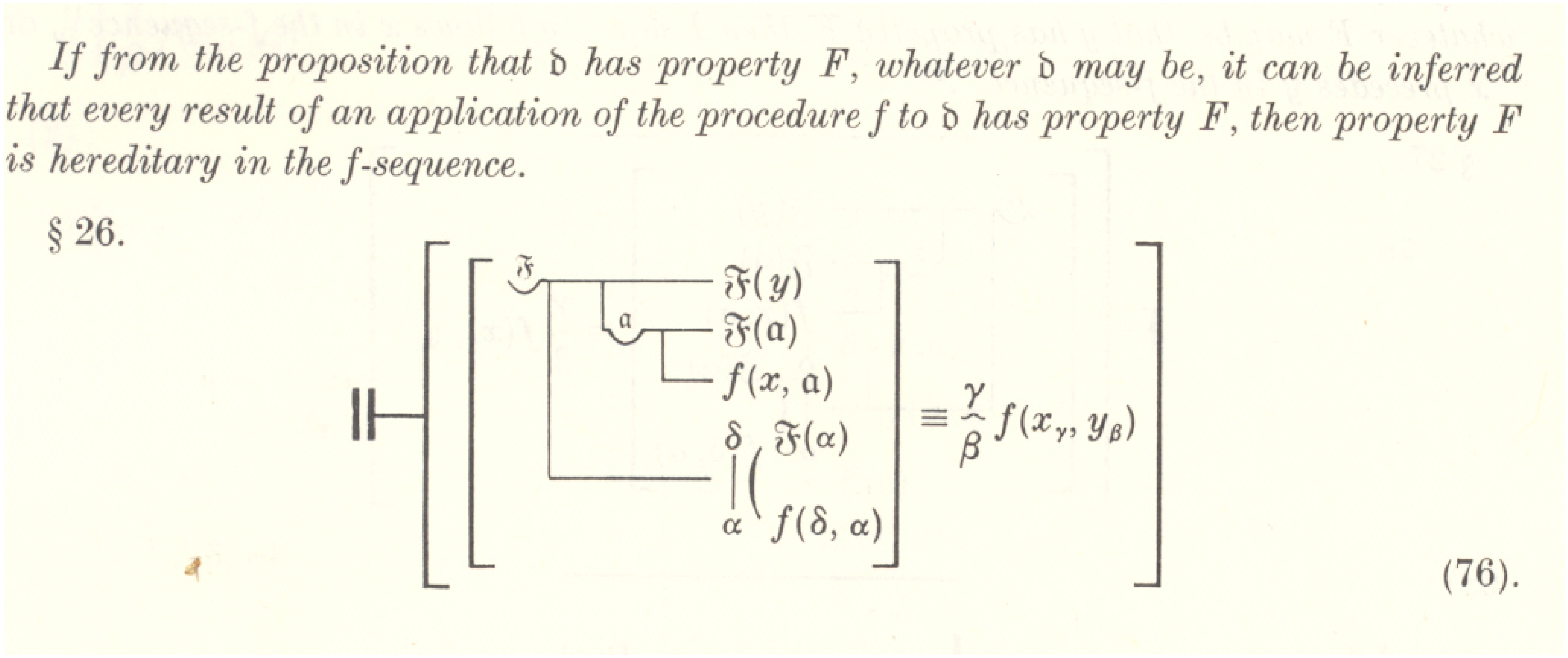


we can derive



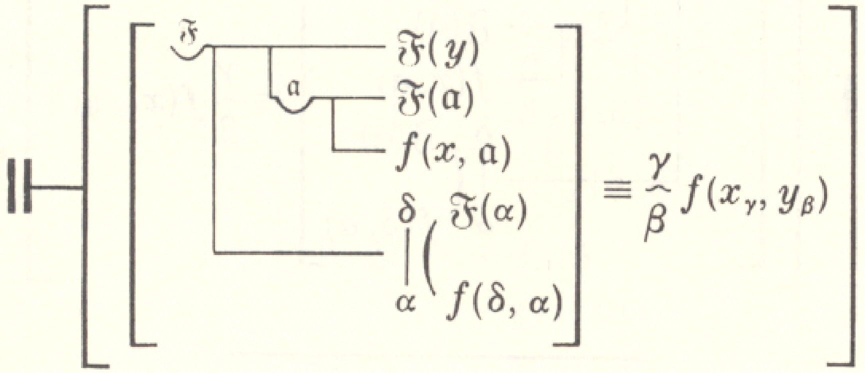
if A is an expression in which a does not occur and if a stands only in the argument places of $\Phi(a)$.[14] If $—\overset{\mathfrak{a}}{\smile}—\Phi(\mathfrak{a})$ is denied, we must be able to specify a meaning for a such that $\Phi(a)$ will be denied. If, therefore, $—\overset{\mathfrak{a}}{\smile}—\Phi(\mathfrak{a})$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

Gottlob Frege (1879) — Quantifiers (∀)



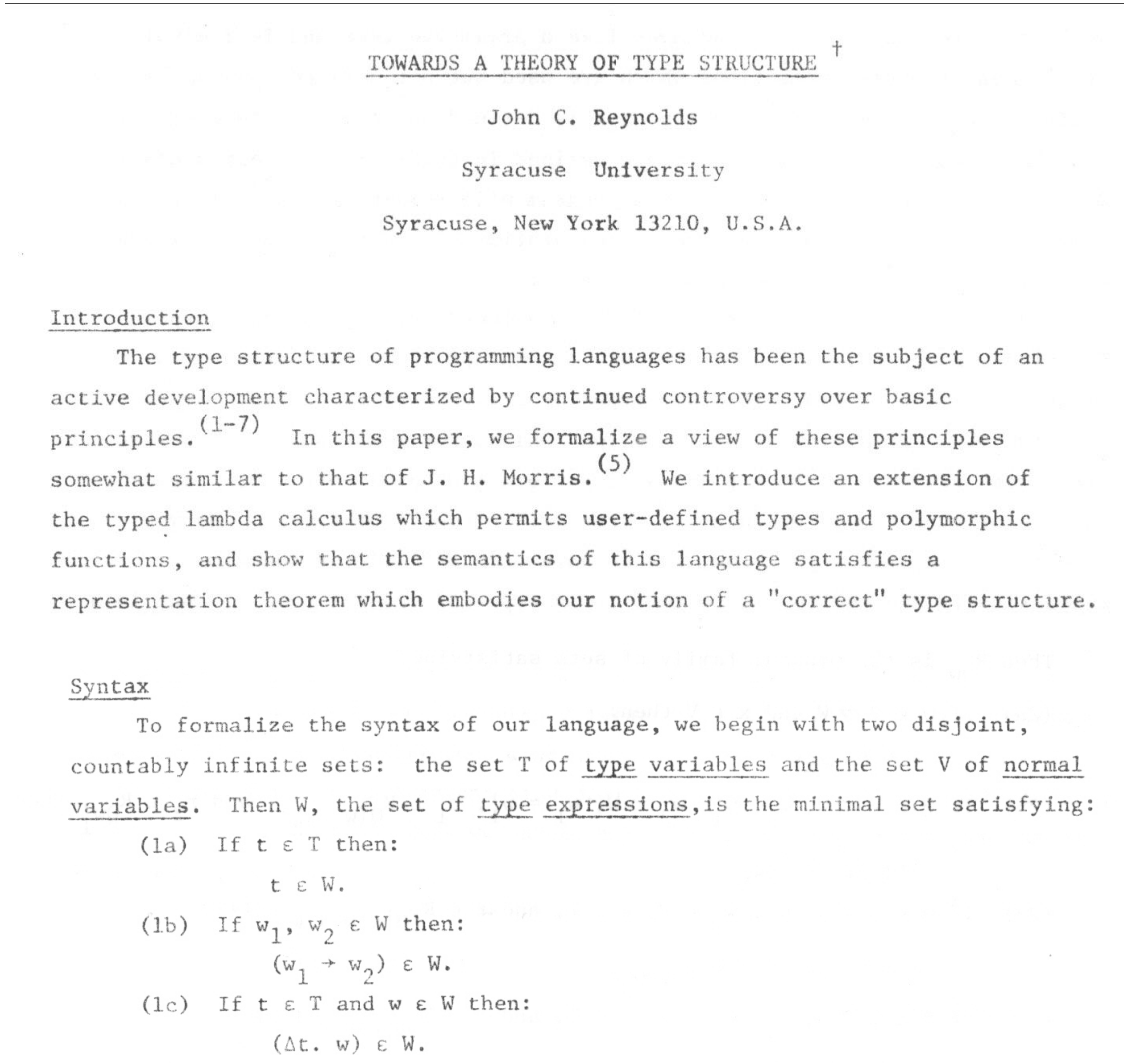
If from the proposition that 𝔡 *has property F, whatever* 𝔡 *may be, it can be inferred that every result of an application of the procedure f to* 𝔡 *has property F, then property F is hereditary in the f-sequence.*

§ 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:

$t \in W$.

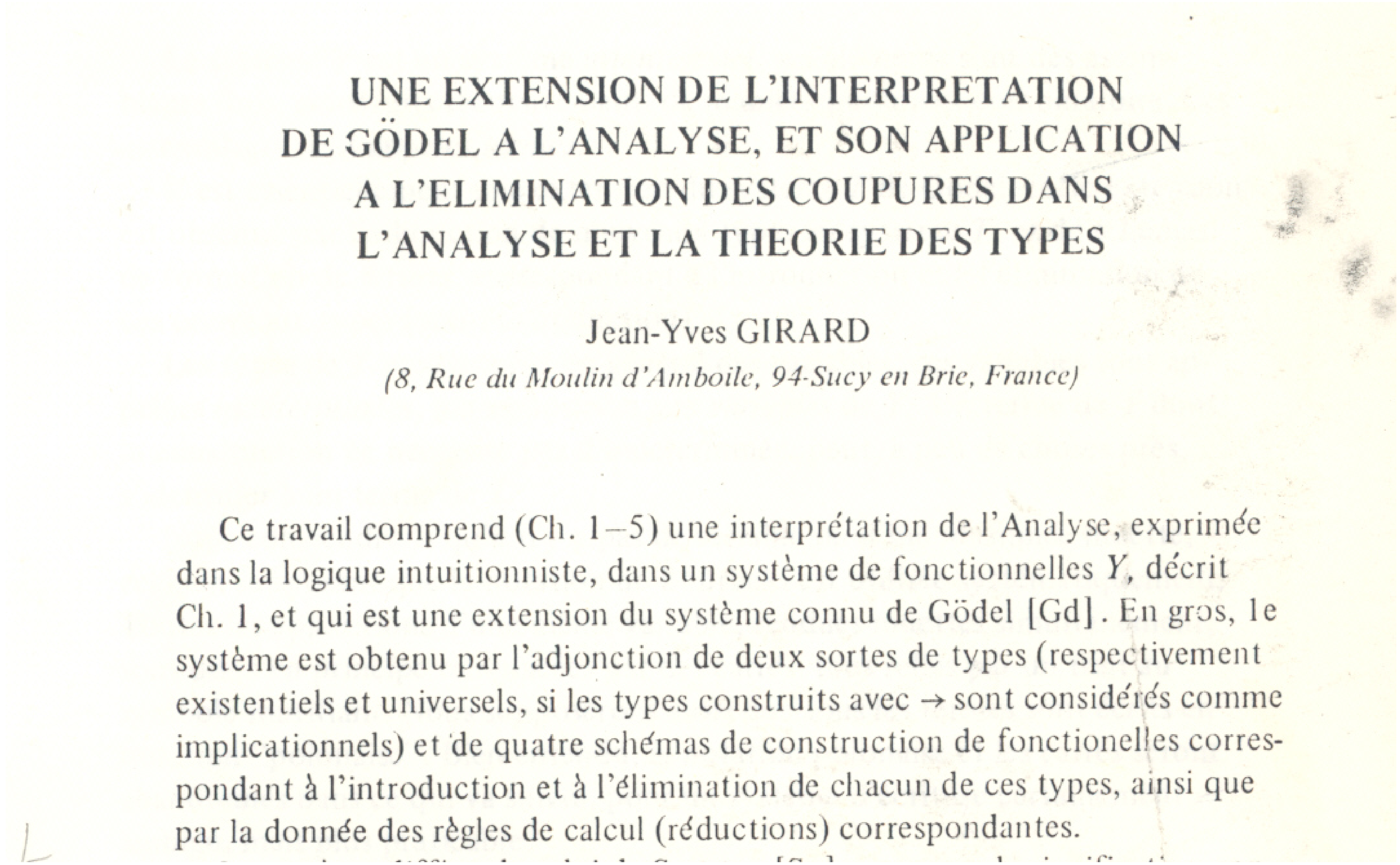
(1b) If $w_1, w_2 \in W$ then:

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

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

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

Jean-Yves Girard (1972) — Polymorphism



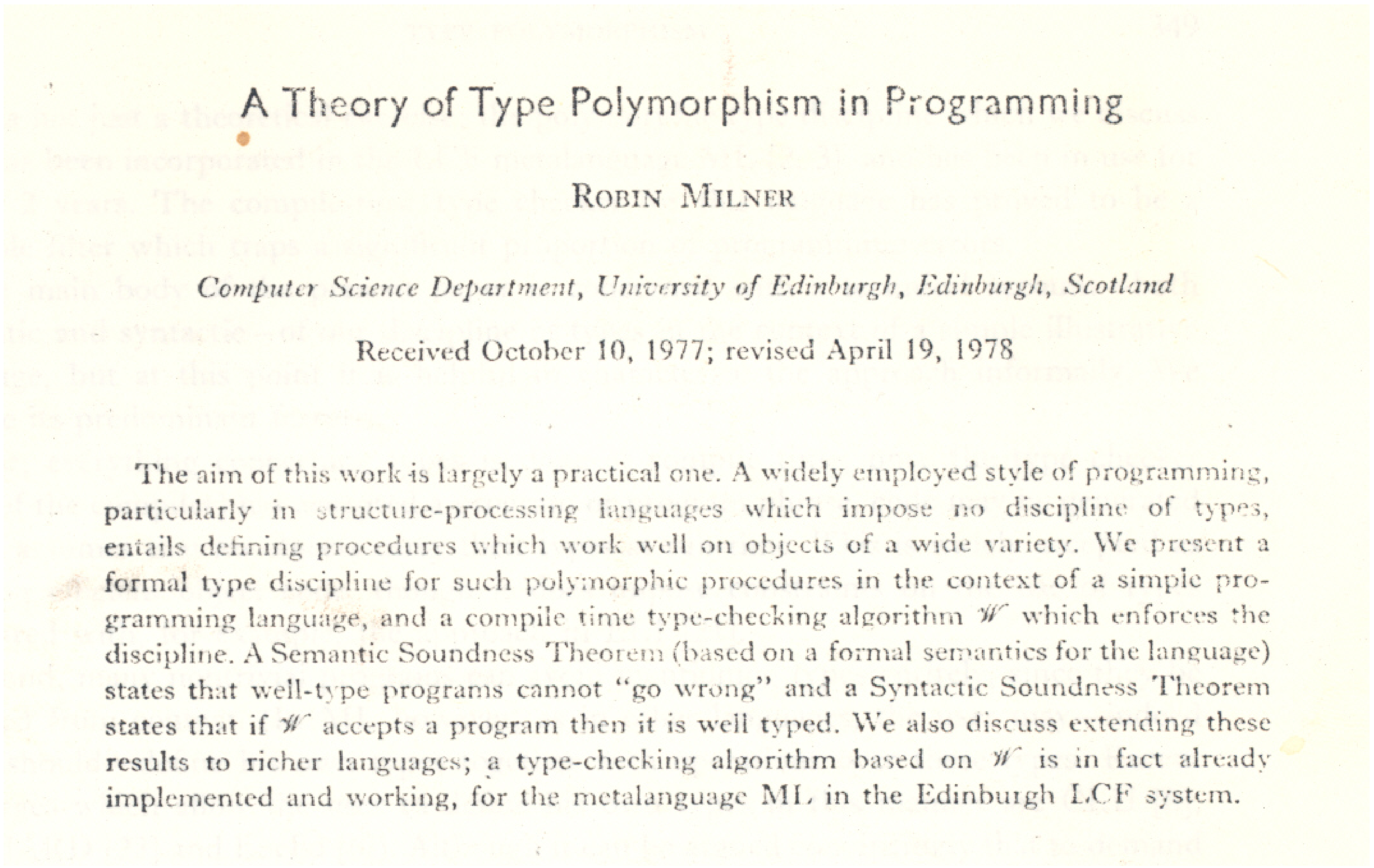
UNE EXTENSION DE L'INTERPRETATION DE GÖDEL A L'ANALYSE, ET SON APPLICATION A L'ELIMINATION DES COUPURES DANS L'ANALYSE ET LA THEORIE DES TYPES

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 $\rightarrow$ 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

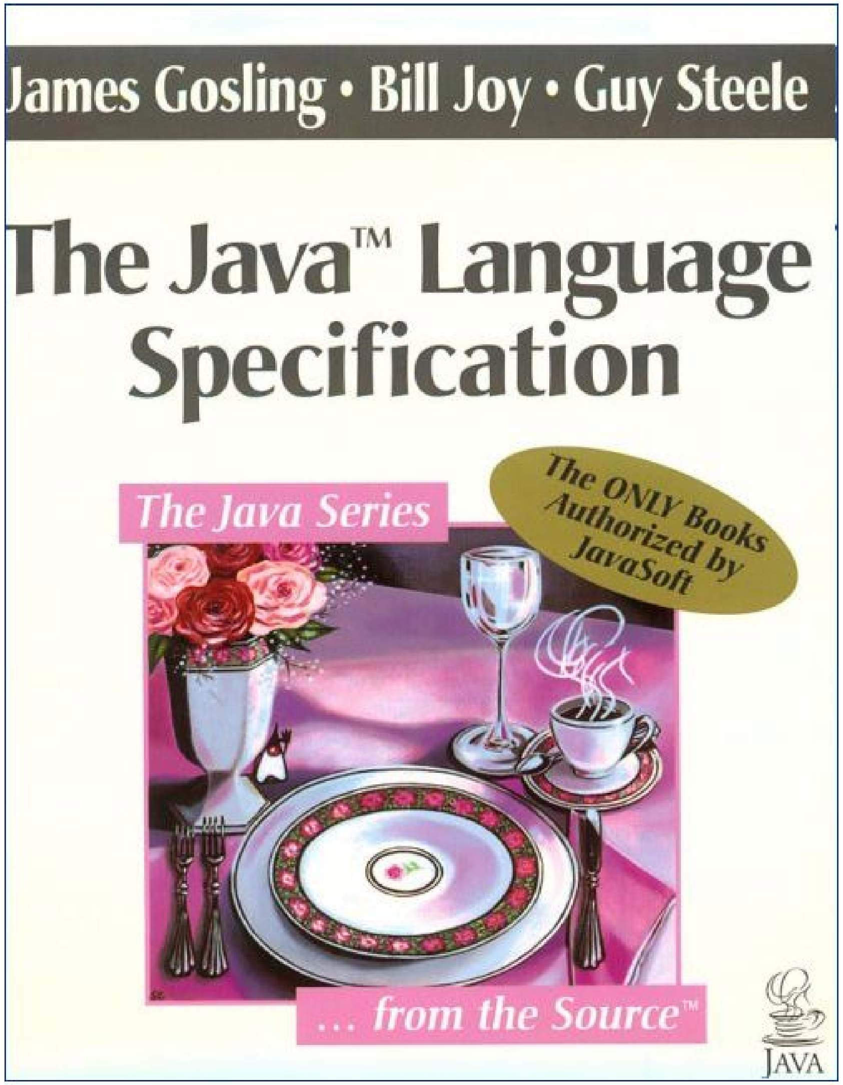
ROBIN MILNER

Computer Science Department, University of Edinburgh, Edinburgh, Scotland

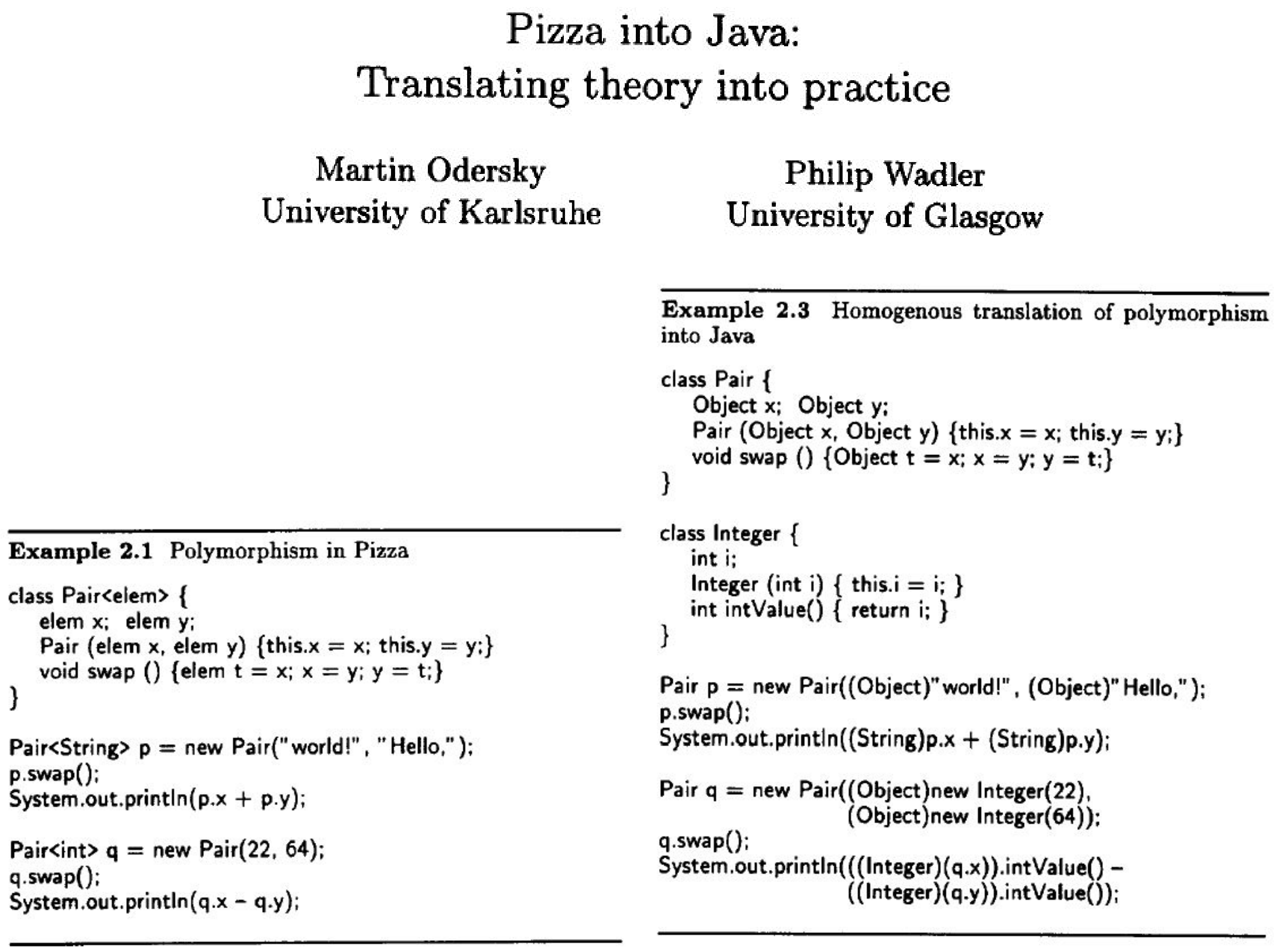
Received October 10, 1977; revised April 19, 1978

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


James Gosling • Bill Joy • Guy Steele
The Java™ Language Specification
The Java Series
The ONLY Books Authorized by JavaSoft
... from the Source™
JAVA

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

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

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

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

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

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

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

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

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

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

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

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

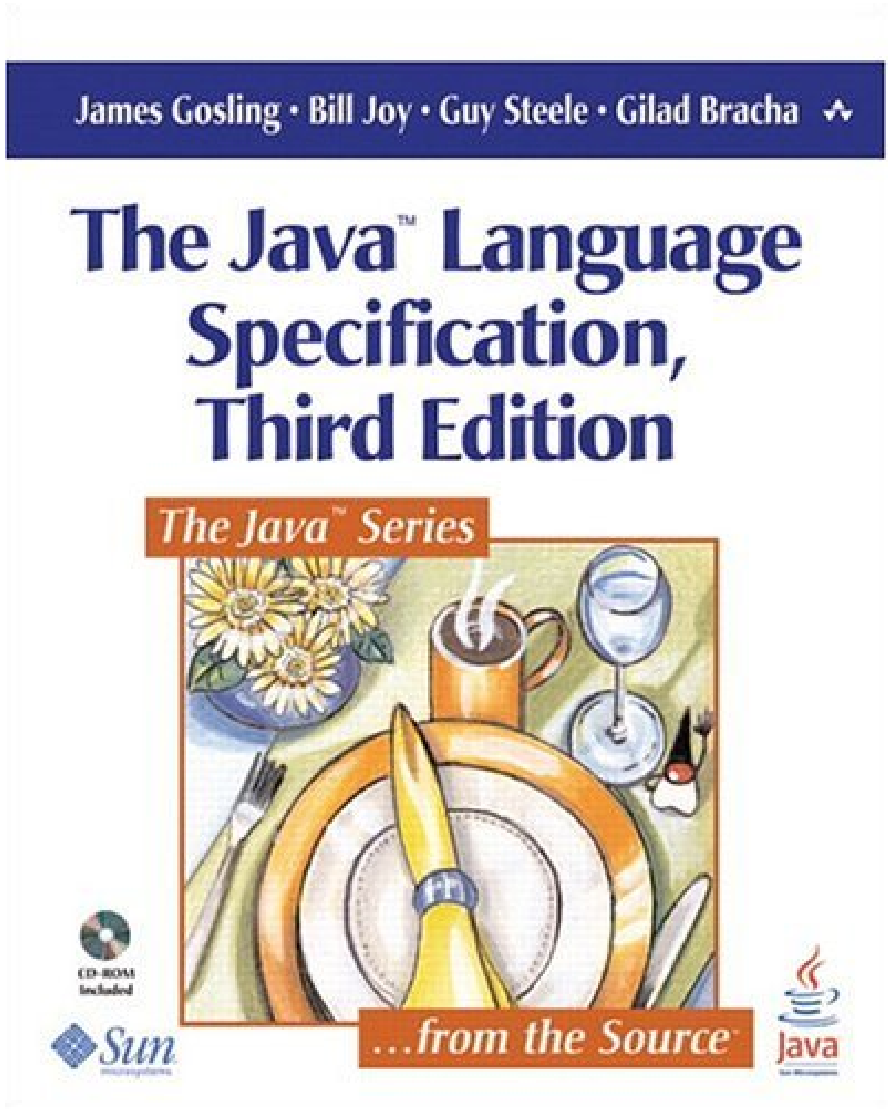
$$\frac{\Delta \vdash \mathtt{N}\ \mathrm{ok} \qquad \mathit{fields}(\mathtt{N}) = \bar{\mathtt{T}}\ \bar{\mathtt{f}} \qquad \Delta;\Gamma \vdash \bar{\mathtt{e}} : \bar{\mathtt{S}} \qquad \Delta \vdash \bar{\mathtt{S}} <: \bar{\mathtt{T}}}{\Delta;\Gamma \vdash \mathtt{new}\ \mathtt{N}(\bar{\mathtt{e}})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

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

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

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

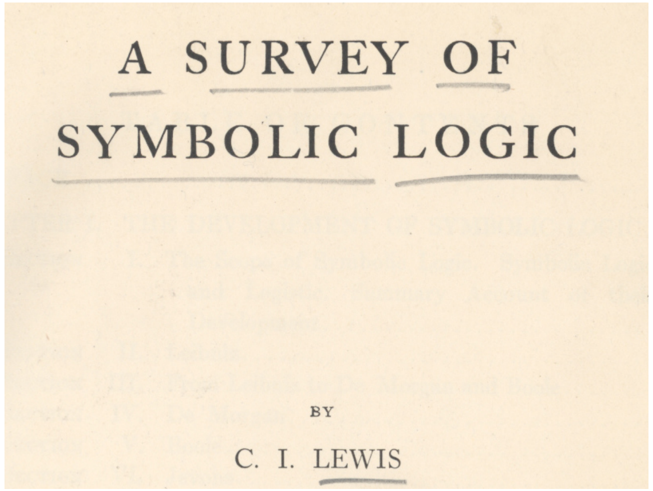
Gosling, Joy, Steele, Bracha (2004) — Java 5


James Gosling · Bill Joy · Guy Steele · Gilad Bracha
The Java™ Language Specification, Third Edition
The Java™ Series
CD-ROM Included
Sun
...from the Source
Java

Part III

Modality, monads, and XML

Clarence Lewis (1918) — Modal Logic

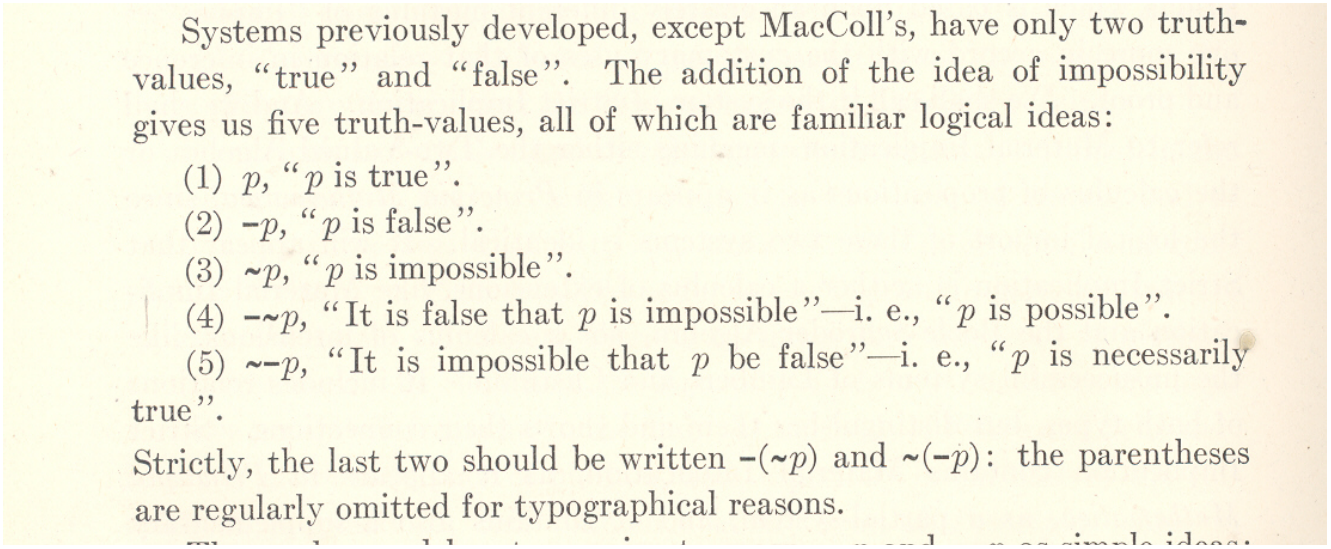


A SURVEY OF

SYMBOLIC LOGIC

BY

C. I. LEWIS

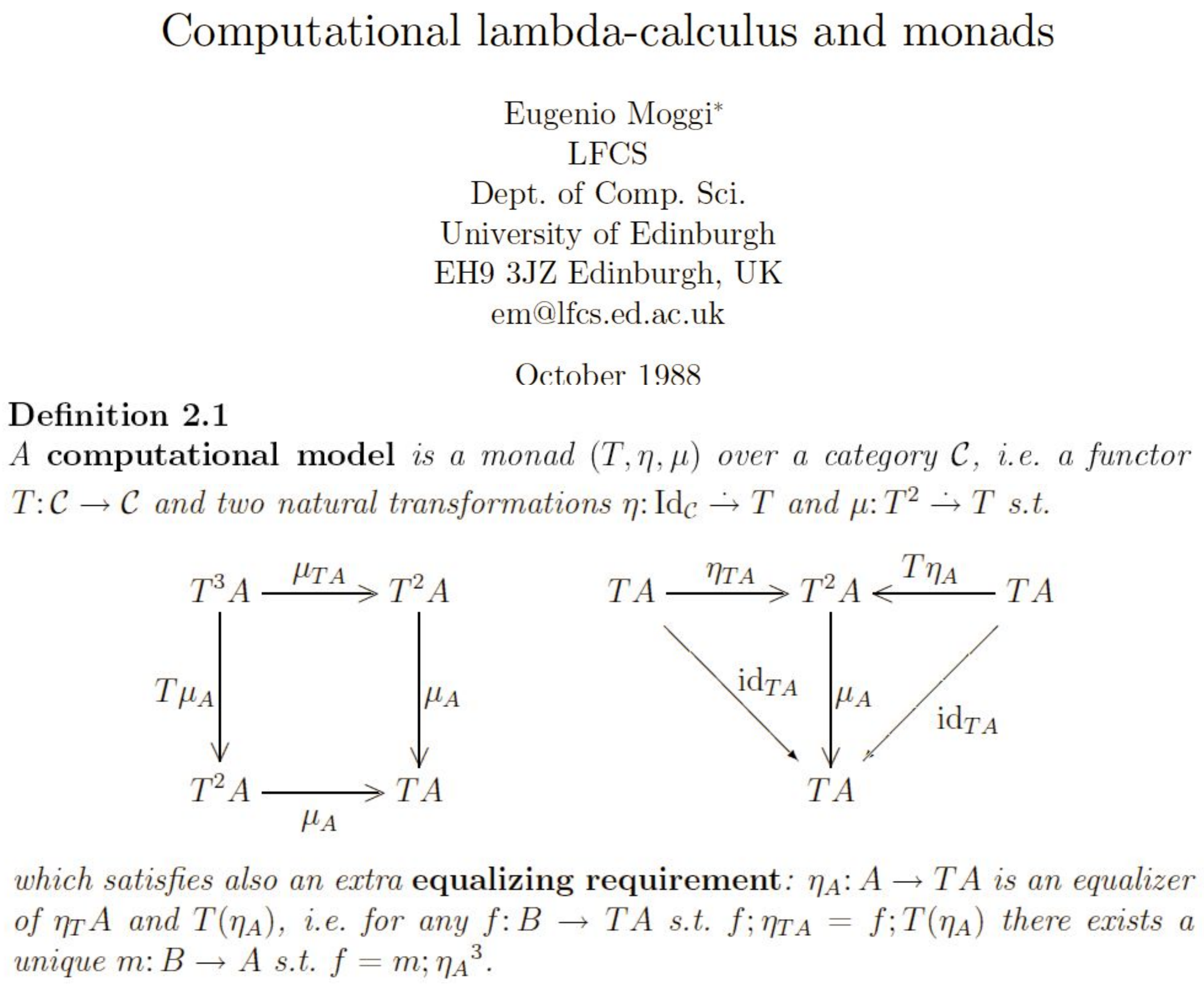


Systems previously developed, except MacColl's, have only two truth-values, "true" and "false". The addition of the idea of impossibility gives us five truth-values, all of which are familiar logical ideas:

(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



Computational lambda-calculus and monads

Eugenio Moggi[*]
LFCS
Dept. of Comp. Sci.
University of Edinburgh
EH9 3JZ Edinburgh, UK
em@lfcs.ed.ac.uk

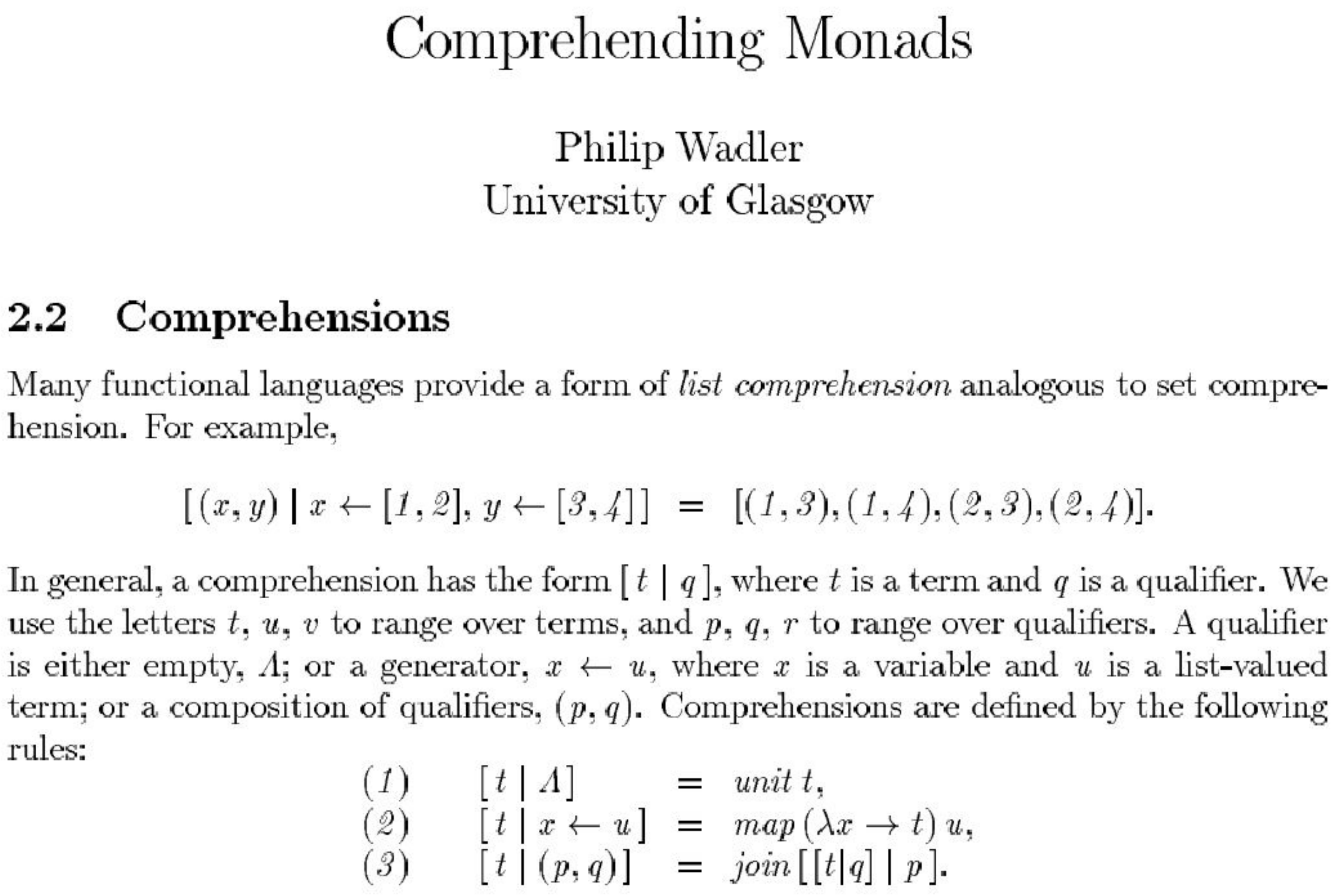
October 1988

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* $\eta: \mathrm{Id}_{\mathcal{C}} \dot{\to} T$ *and* $\mu: T^2 \dot{\to} T$ *s.t.*

$$\begin{tikzcd}
T^3A \arrow[r, "\mu_{TA}"] \arrow[d, "T\mu_A"'] & T^2A \arrow[d, "\mu_A"] \\
T^2A \arrow[r, "\mu_A"'] & TA
\end{tikzcd}
\qquad
\begin{tikzcd}
TA \arrow[r, "\eta_{TA}"] \arrow[dr, "\mathrm{id}_{TA}"'] & T^2A \arrow[d, "\mu_A"] & TA \arrow[l, "T\eta_A"'] \arrow[dl, "\mathrm{id}_{TA}"] \\
& TA &
\end{tikzcd}$$

which satisfies also an extra **equalizing requirement***:* $\eta_A: A \to TA$ *is an equalizer of* η_{TA} *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



Comprehending Monads

Philip Wadler
University of Glasgow

2.2 Comprehensions

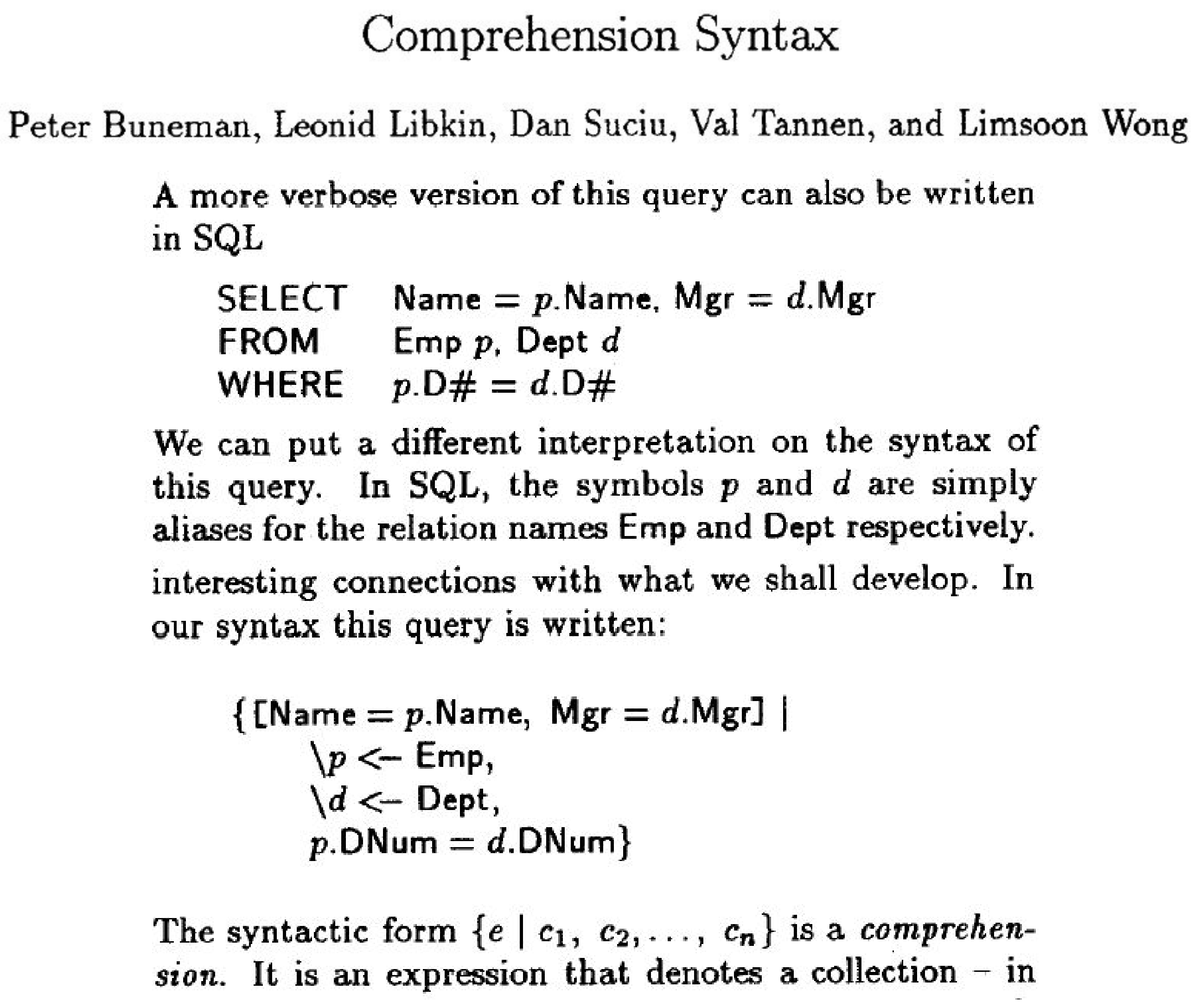
Many functional languages provide a form of *list comprehension* analogous to set comprehension. For example,

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

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:

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

Peter Buneman *et al* (1991) — Comprehensions



Comprehension Syntax

Peter Buneman, Leonid Libkin, Dan Suciu, Val Tannen, and Limsoon Wong

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#
```

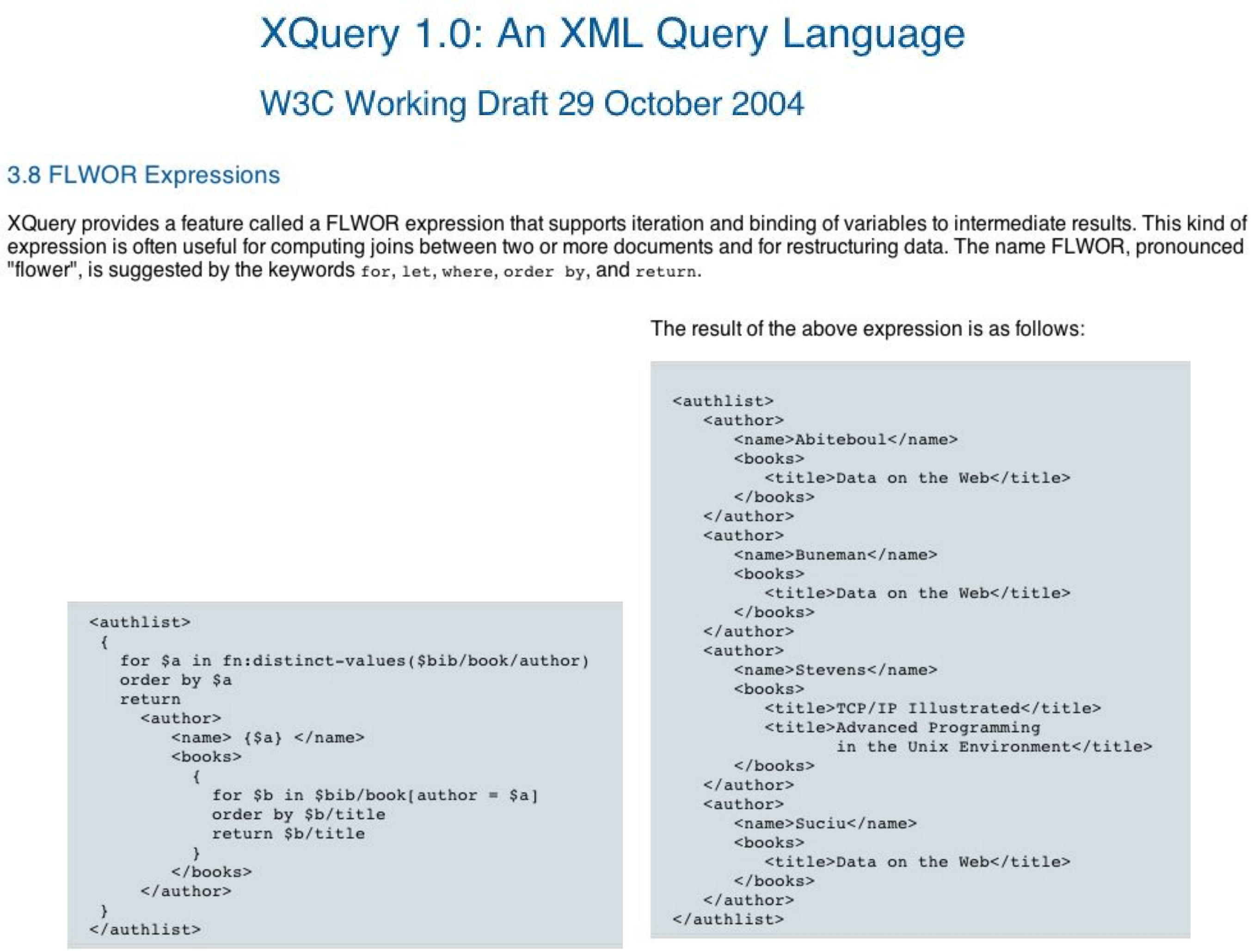
We can put a different interpretation on the syntax of this query. In SQL, the symbols p and d are simply aliases for the relation names Emp and Dept respectively.

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



XQuery 1.0: An XML Query Language

W3C Working Draft 29 October 2004

3.8 FLWOR Expressions

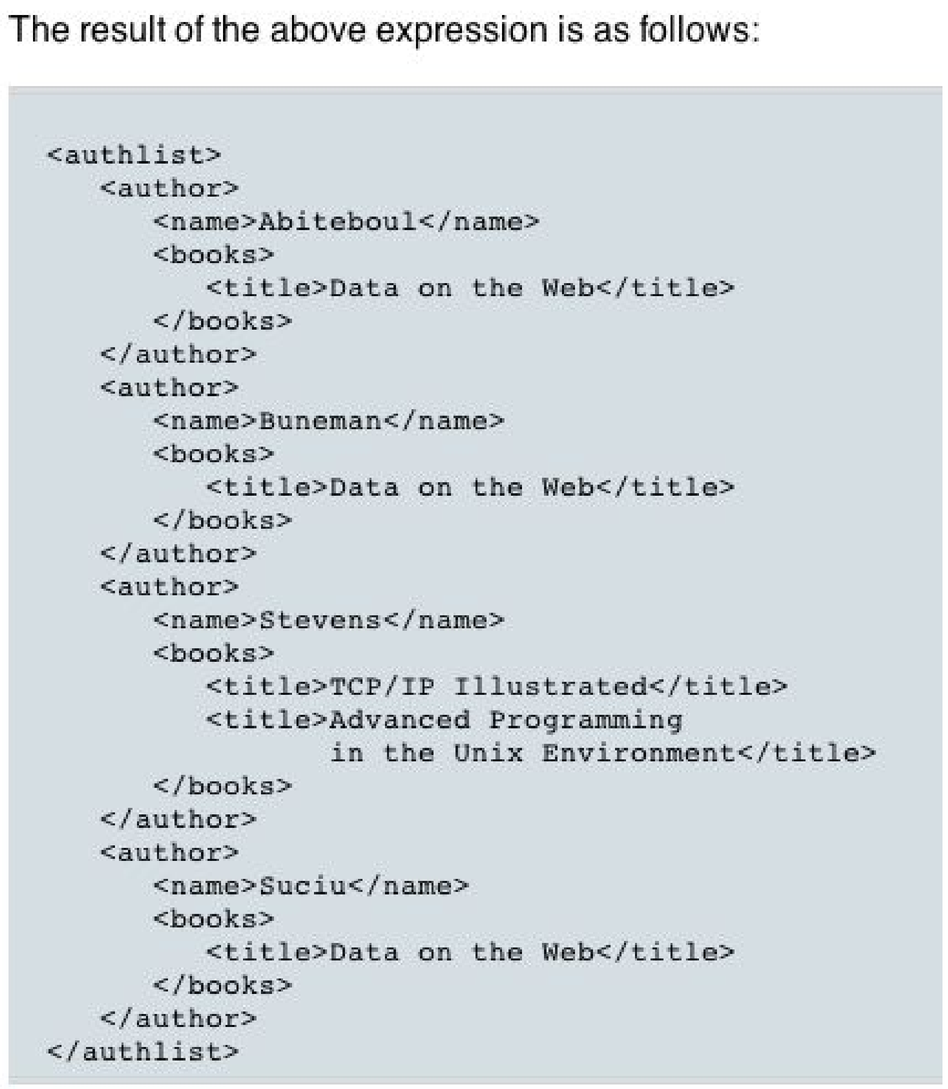
XQuery provides a feature called a FLWOR expression that supports iteration and binding of variables to intermediate results. This kind of expression is often useful for computing joins between two or more documents and for restructuring data. The name FLWOR, pronounced "flower", is suggested by the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

The result of the above expression is as follows:

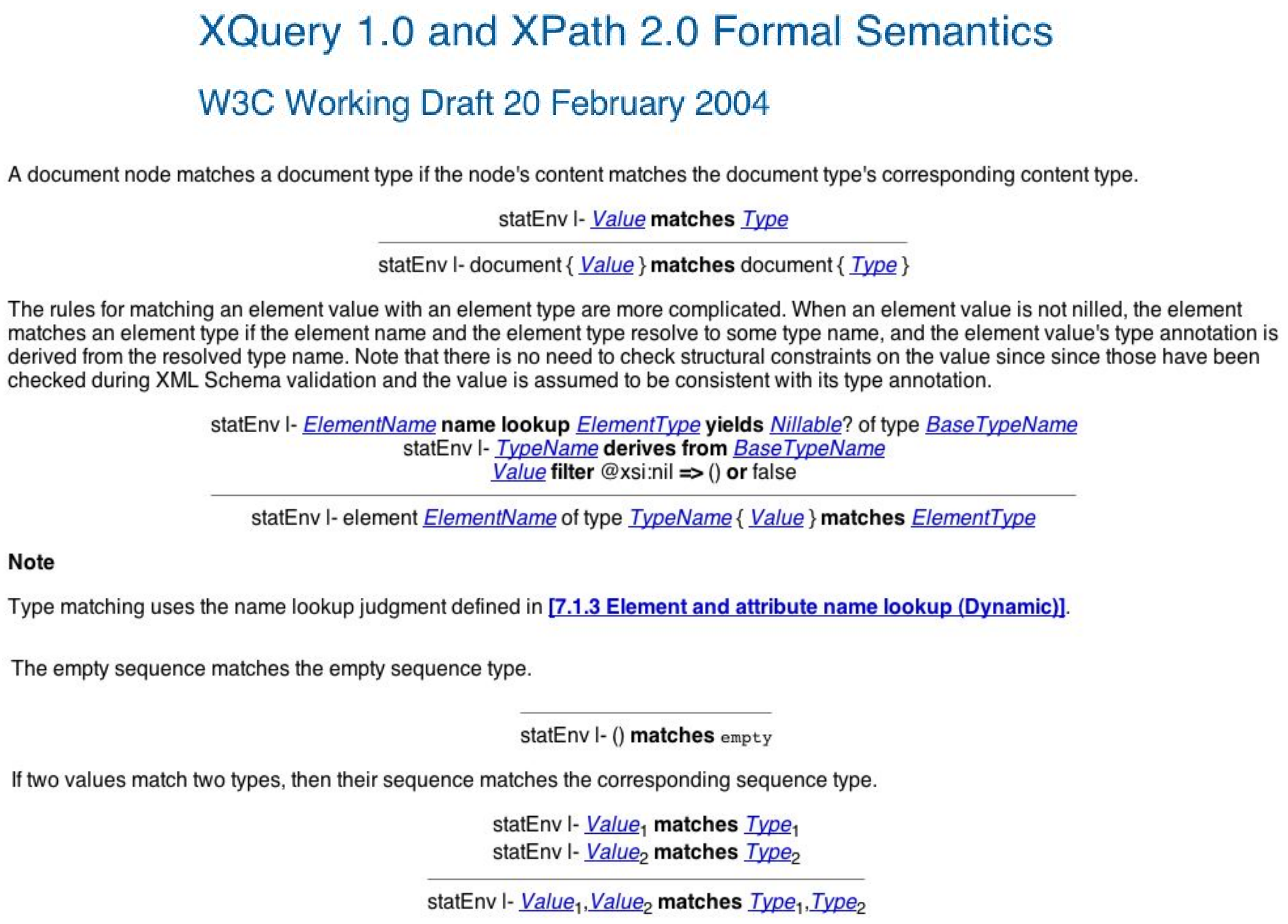
```
<authlist>
   <author>
      <name>Abiteboul</name>
      <books>
         <title>Data on the Web</title>
      </books>
   </author>
   <author>
      <name>Buneman</name>
      <books>
         <title>Data on the Web</title>
      </books>
   </author>
   <author>
      <name>Stevens</name>
      <books>
         <title>TCP/IP Illustrated</title>
         <title>Advanced Programming
                in the Unix Environment</title>
      </books>
   </author>
   <author>
      <name>Suciu</name>
      <books>
         <title>Data on the Web</title>
      </books>
   </author>
</authlist>
```

XQuery (2004) — FLWOR


The result of the above expression is as follows:

```
<authlist>
   <author>
      <name>Abiteboul</name>
      <books>
         <title>Data on the Web</title>
      </books>
   </author>
   <author>
      <name>Buneman</name>
      <books>
         <title>Data on the Web</title>
      </books>
   </author>
   <author>
      <name>Stevens</name>
      <books>
         <title>TCP/IP Illustrated</title>
         <title>Advanced Programming
                in the Unix Environment</title>
      </books>
   </author>
   <author>
      <name>Suciu</name>
      <books>
         <title>Data on the Web</title>
      </books>
   </author>
</authlist>
```

XQuery (2004) — Formal Semantics



XQuery 1.0 and XPath 2.0 Formal Semantics

W3C Working Draft 20 February 2004

A document node matches a document type if the node's content matches the document type's corresponding content type.

$$\frac{\text{statEnv} \mid\!- \ \textit{Value}\ \textbf{matches}\ \textit{Type}}{\text{statEnv} \mid\!- \ \text{document}\{\ \textit{Value}\ \}\ \textbf{matches}\ \text{document}\{\ \textit{Type}\ \}}$$

The rules for matching an element value with an element type are more complicated. When an element value is not nilled, the element matches an element type if the element name and the element type resolve to some type name, and the element value's type annotation is derived from the resolved type name. Note that there is no need to check structural constraints on the value since since those have been checked during XML Schema validation and the value is assumed to be consistent with its type annotation.

$$\frac{\begin{array}{c}\text{statEnv} \mid\!- \ \textit{ElementName}\ \textbf{name lookup}\ \textit{ElementType}\ \textbf{yields}\ \textit{Nillable}?\ \text{of type}\ \textit{BaseTypeName} \\ \text{statEnv} \mid\!- \ \textit{TypeName}\ \textbf{derives from}\ \textit{BaseTypeName} \\ \textit{Value}\ \textbf{filter}\ \text{@xsi:nil} \Rightarrow ()\ \textbf{or}\ \text{false}\end{array}}{\text{statEnv} \mid\!- \ \text{element}\ \textit{ElementName}\ \text{of type}\ \textit{TypeName}\ \{\ \textit{Value}\ \}\ \textbf{matches}\ \textit{ElementType}}$$

Note

Type matching uses the name lookup judgment defined in **[7.1.3 Element and attribute name lookup (Dynamic)]**.

The empty sequence matches the empty sequence type.

$$\frac{}{\text{statEnv} \mid\!- \ ()\ \textbf{matches}\ \texttt{empty}}$$

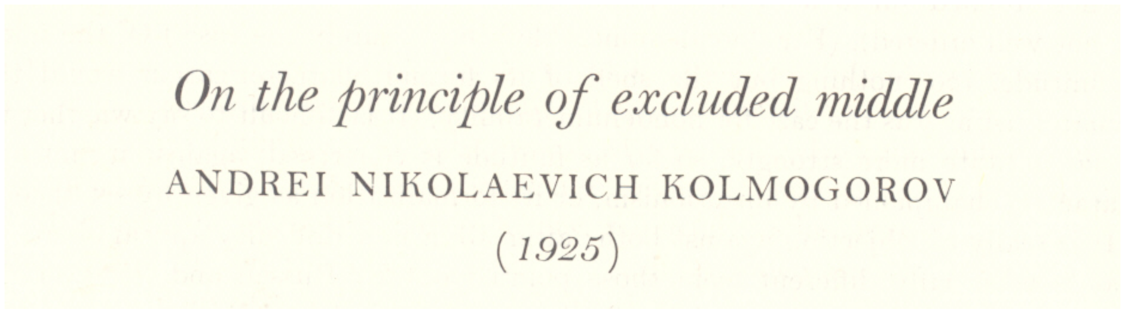
If two values match two types, then their sequence matches the corresponding sequence type.

$$\frac{\begin{array}{c}\text{statEnv} \mid\!- \ \textit{Value}_1\ \textbf{matches}\ \textit{Type}_1 \\ \text{statEnv} \mid\!- \ \textit{Value}_2\ \textbf{matches}\ \textit{Type}_2\end{array}}{\text{statEnv} \mid\!- \ \textit{Value}_1,\textit{Value}_2\ \textbf{matches}\ \textit{Type}_1,\textit{Type}_2}$$

Part IV

Classical logic, continuations, and the Web

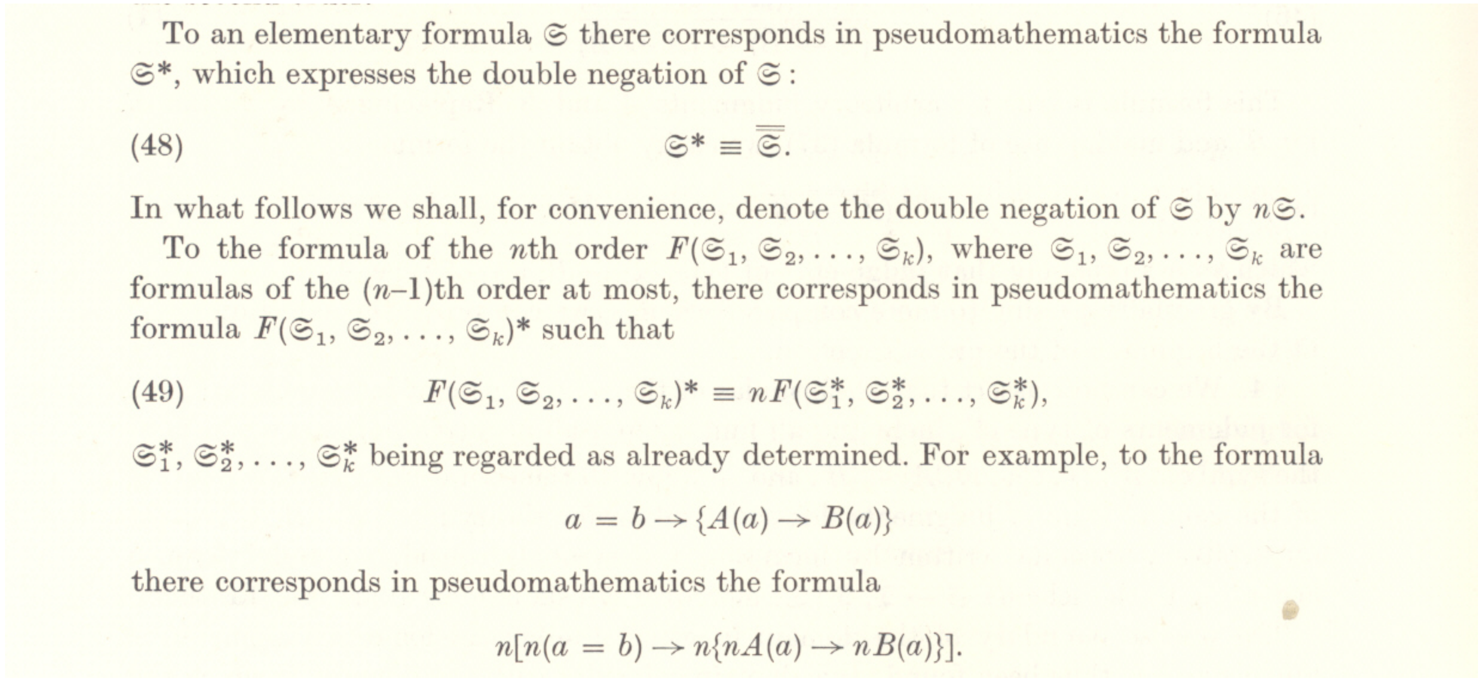
Andrei Kolmogorov (1925)



On the principle of excluded middle

ANDREI NIKOLAEVICH 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

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

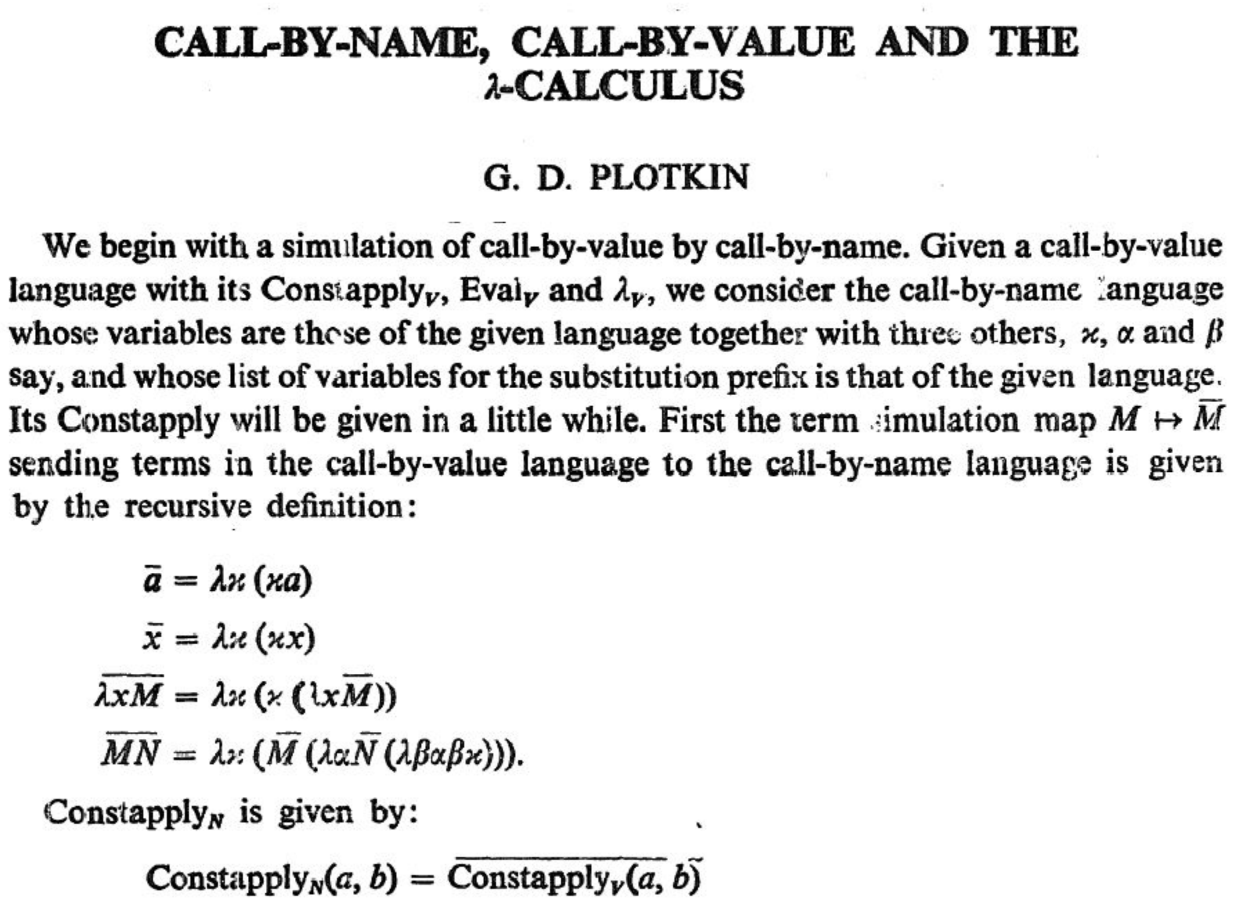
$\mathfrak{S}_1^*, \mathfrak{S}_2^*, \ldots, \mathfrak{S}_k^*$ being regarded as already determined. For example, to the formula

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

there corresponds in pseudomathematics the formula

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

Gordon Plotkin (1975)



CALL-BY-NAME, CALL-BY-VALUE AND THE λ-CALCULUS

G. D. PLOTKIN

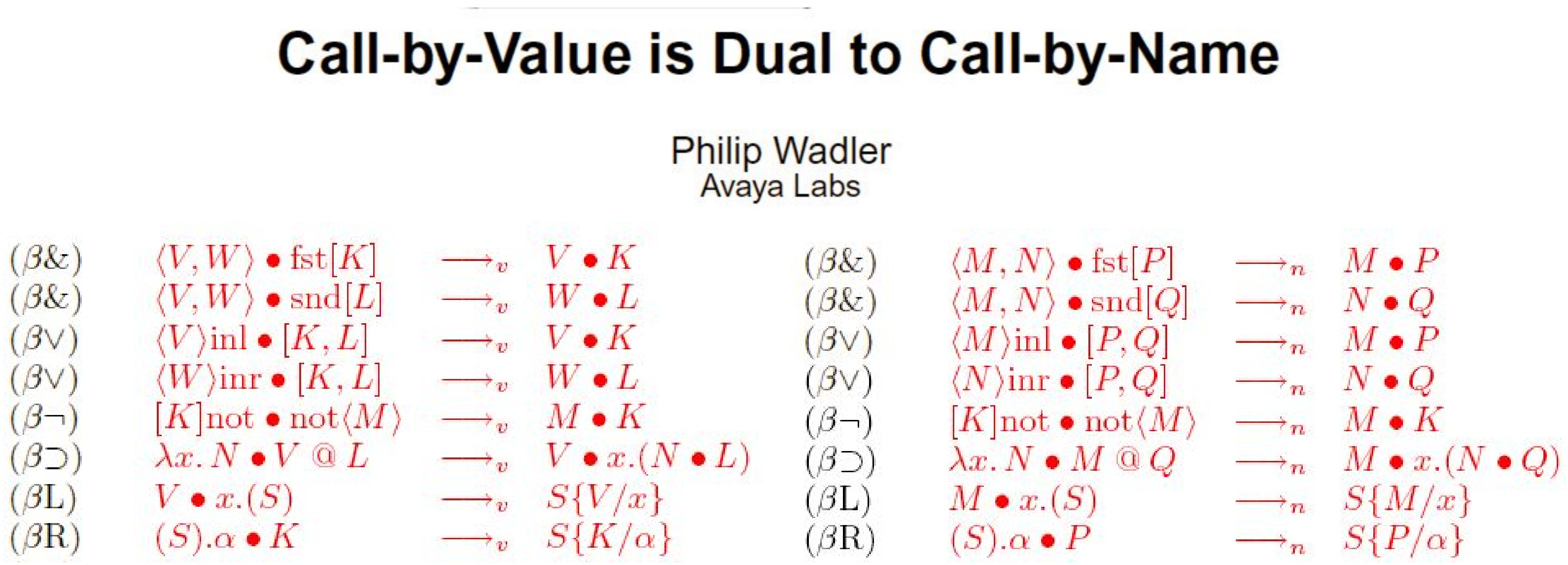
We begin with a simulation of call-by-value by call-by-name. Given a call-by-value language with its Constapply$_V$, Eval$_V$ and λ_V, we consider the call-by-name language whose variables are those of the given language together with three others, $\varkappa$, α and β say, and whose list of variables 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:

$$\begin{aligned}
\overline{a} &= \lambda\varkappa\,(\varkappa a)\\
\overline{x} &= \lambda\varkappa\,(\varkappa x)\\
\overline{\lambda x M} &= \lambda\varkappa\,(\varkappa\,(\lambda x \overline{M}))\\
\overline{MN} &= \lambda\varkappa\,(\overline{M}\,(\lambda\alpha\overline{N}\,(\lambda\beta\alpha\beta\varkappa))).
\end{aligned}$$

Constapply$_N$ is given by:

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

Philip Wadler (2000)



Call-by-Value is Dual to Call-by-Name

Philip Wadler
Avaya Labs

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

Philip Wadler (2000)

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

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

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

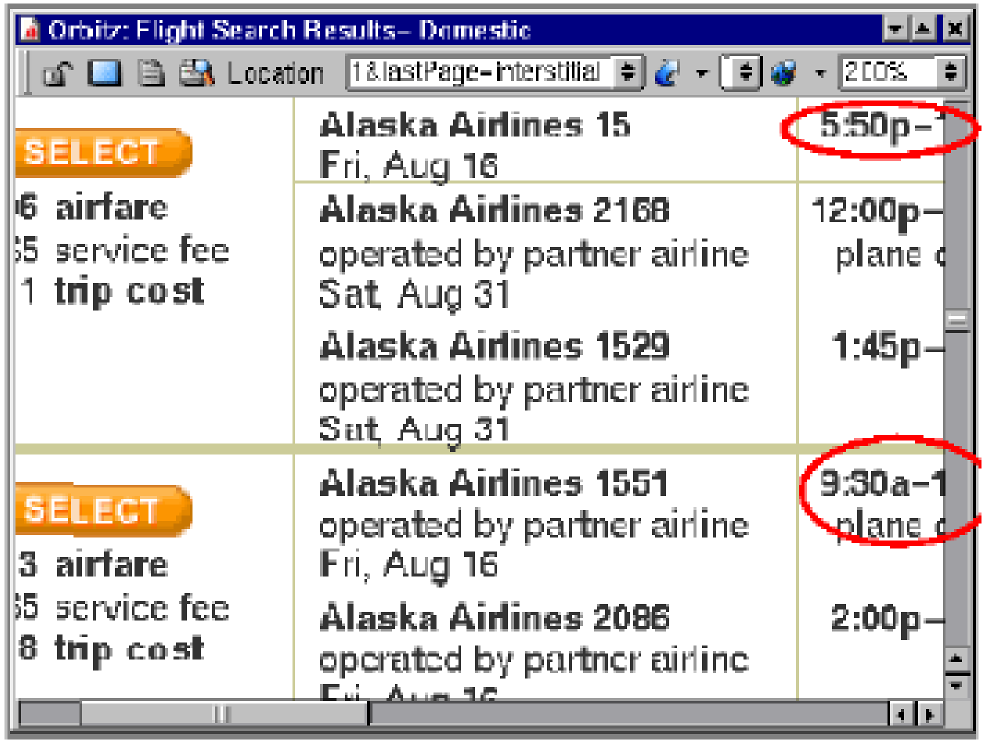
$$\frac{K : A \mid \Gamma \rightarrow \Theta}{\Gamma \rightarrow \Theta \mid [K]\text{not} : \neg A}\,\neg\text{R} \qquad \frac{\Gamma \rightarrow \Theta \mid M : A}{\text{not}\langle M\rangle : \neg A \mid \Gamma \rightarrow \Theta}\,\neg\text{L}$$

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

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

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

Orbitz: Two flights


Orbitz: Flight Search Results– Domestic

Location 1&lastPage=interstitial 200%

SELECT
6 airfare
5 service fee
1 trip cost

Alaska Airlines 15
Fri, Aug 16
5:50p–

Alaska Airlines 2168
operated by partner airline
Sat, Aug 31
12:00p–
plane c

Alaska Airlines 1529
operated by partner airline
Sat, Aug 31
1:45p–

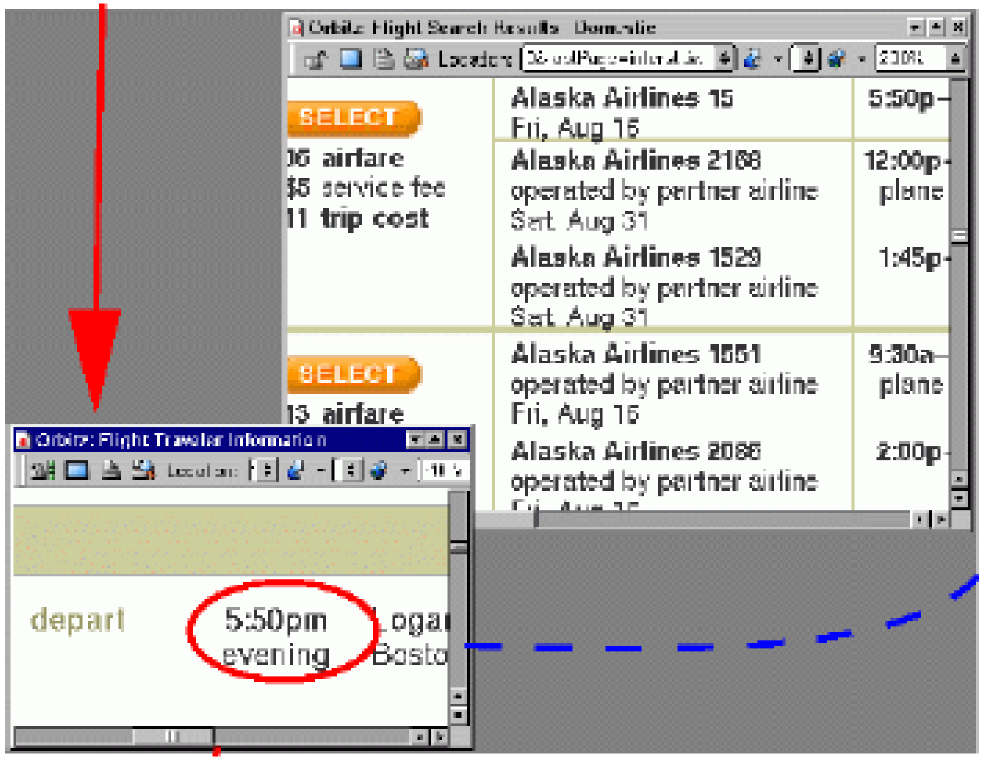
SELECT
3 airfare
5 service fee
8 trip cost

Alaska Airlines 1551
operated by partner airline
Fri, Aug 16
9:30a–1
plane c

Alaska Airlines 2086
operated by partner airline
2:00p–

Graunke, Findler, Krishnamurthi, Felleisen (ESOP 2003)

Orbitz: Clone and submit first



Orbitz: Flight Search Results - Domestic

SELECT

06 airfare
$5 service fee
11 trip cost

Alaska Airlines 15
Fri, Aug 16
5:50p

Alaska Airlines 2168
operated by partner airline
Sat, Aug 31
12:00p
plane

Alaska Airlines 1529
operated by partner airline
Sat, Aug 31
1:45p

SELECT

13 airfare

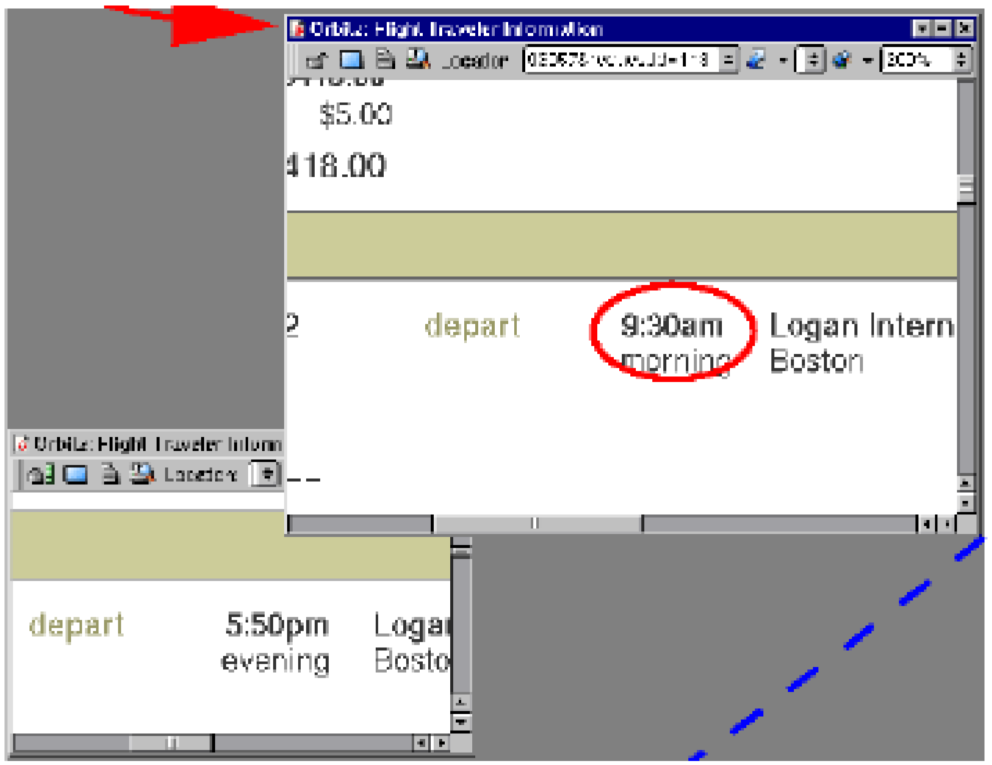
Alaska Airlines 1551
operated by partner airline
Fri, Aug 16
9:30a
plane

Alaska Airlines 2086
operated by partner airline
2:00p

Orbitz: Flight Traveler Information

depart
5:50pm
evening
Logan
Bosto

Orbitz: Submit second



Orbitz: Flight Traveler Information

$5.00

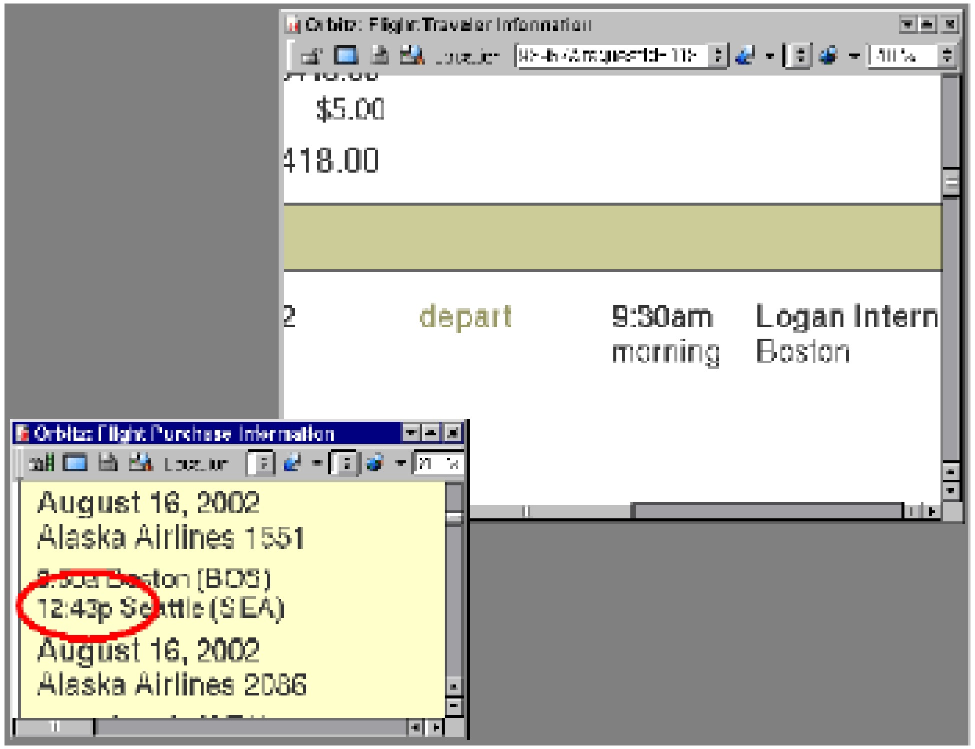
418.00

2 depart 9:30am morning Logan Intern Boston

Orbitz: Flight Traveler Inform

depart 5:50pm evening Logan Boston

Orbitz: Select first – problem!



Orbitz: Flight Traveler Information

$5.00

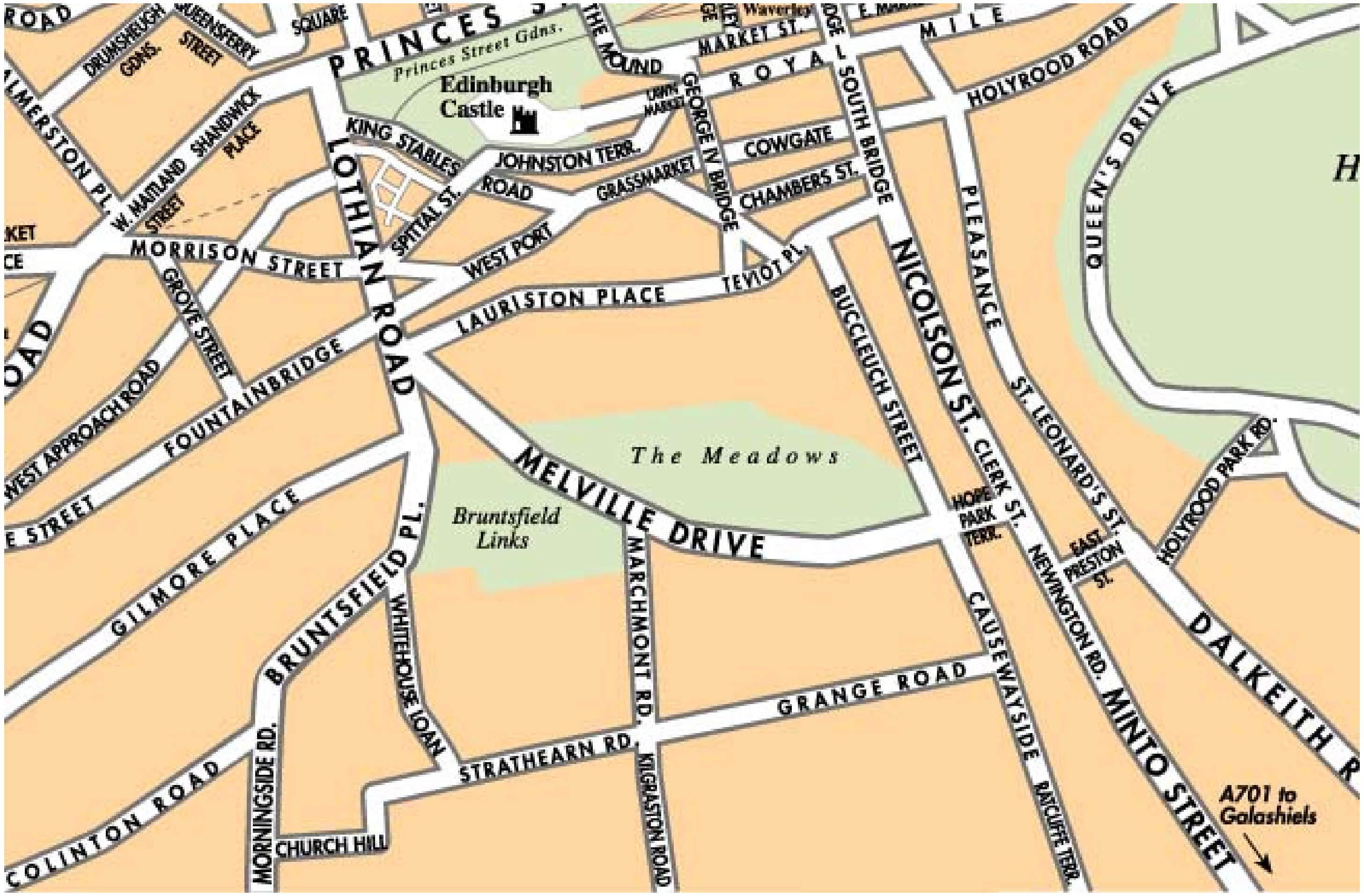
418.00

2 depart 9:30am morning Logan Intern Boston

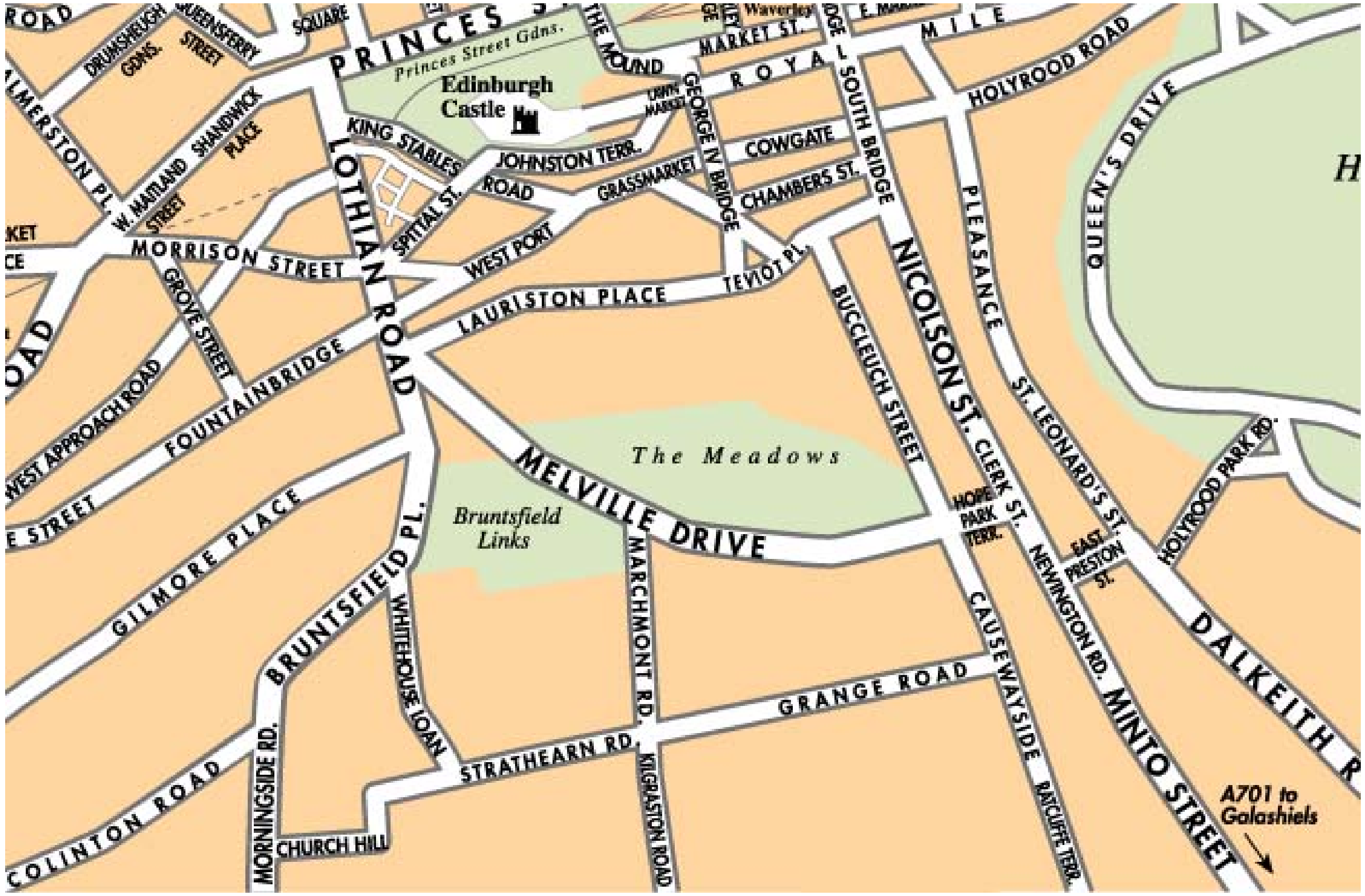
Orbitz: Flight Purchase Information

August 16, 2002
Alaska Airlines 1551
9:30a Boston (BOS)
12:43p Seattle (SEA)
August 16, 2002
Alaska Airlines 2086

Burstall, MacQueen, and Sannella (1980) — Hope

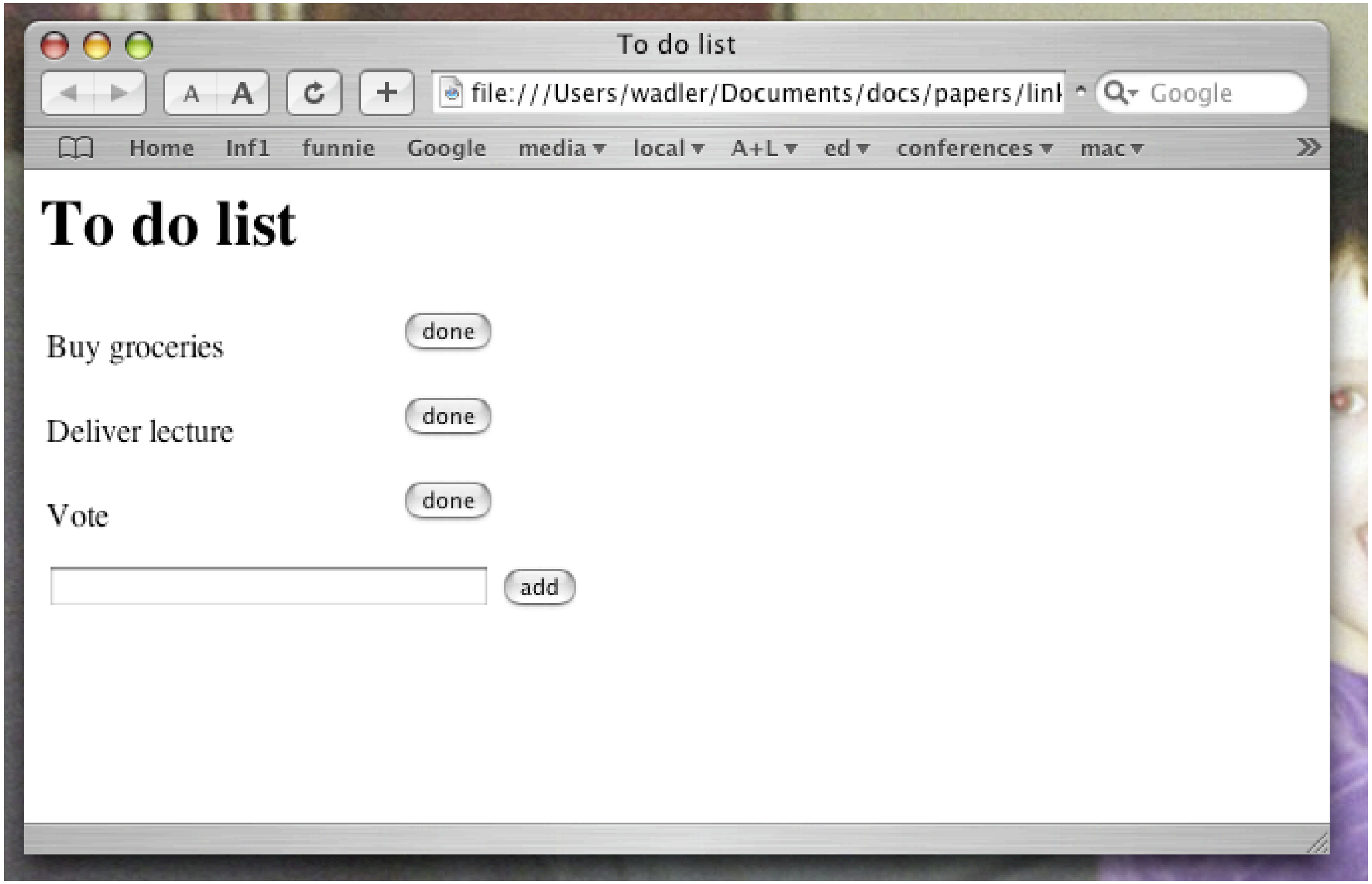


Burstall, MacQueen, and Sannella (1980) — Hope



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 action="{todo(items\\[item])}">
              <input type="submit" value="done"/>
            </form>
          </td>
        </tr>
    }</table>
    <form action="{todo(items++[new])}">
      <input name="{new}" type="text" size="40">
      <input type="submit" value="add"/>
    </form>
  </body></html>.
```



To do list

Buy groceries done

Deliver lecture done

Vote done

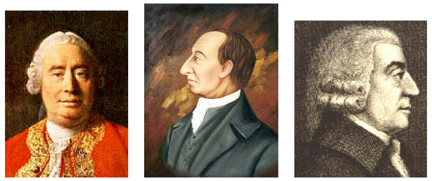
add

Part V

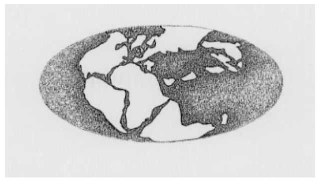
Conclusions

Kinds of coincidence

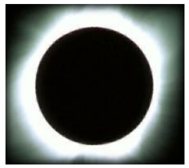
Historical confluence of great minds — Hume, Hutton, Smith



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	

Invitation

Scottish Programming Language Seminar
Invited speakers: John Reynolds, David Watt

10.00–16.00, 7 December 2004, University of Glasgow

Simon Gay, simon@dcs.gla.ac.uk

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