The Axes of Abstraction

"Give me six hours to develop an abstraction and I will spend the first four sharpening the axis."

Paul Phillips Lambdaconf 2017 Boulder, CO

Situation

— Trend: axioms, logic, inference, laws — Math abstractions (e.g. monoids) succeeding — Domain abstractions remain mostly ad hoc — "More art than science" — That's a euphemism for "not science"

Thesis

There is science we can apply.

We can constrain more. We can infer more. We can derive more. We can resist adhocicity.

What is with "axes"

— In the Cartesian sense: x-axis, y-axis, z-axis — Complex numbers literally add an axis — Loosely: a property which varies between extrema — Orthogonality preferred but not required

That didn't sound right...

— ...because abstraction is something else — That was an axis of generalization — Abstraction vs. generalization, coming up — First, the cutting room floor

The axis of axis exoticity

I initially thought I'd talk about axes like:

- Parametricity # giver of free theorems - Arity # size polymorphism - Orderity # kind polymorphism - Fixity # operator associativity - Opacity # "boxity" says @xeno-by - Inductivity # data vs. codata

Algebraic axes

Or algebraic properties like:

- Associativity # category theory - Commutativity # CRDTs - Distributivity # belief propagation - Idempotency # lattices - Invertibility # isomorphisms - Selectivity # idempotent-er

Yet more exoticity

Other candidates:

- Longevity # Lifetimes - Evitability # evaluation strategy - Classity # "first class" is like "power" - Quotidity # macro depth polymorphism

Forget it!

Instead, I learned about something new.

I think it deserves a wider audience.

Coming up.

But first...

Generalization by Example

Generalization

def f(x: 5): 10 = x + x // it's 10!

Generalization

def f(x: 5): 10 = x + x // it's 10! def f(x: Int): Int = x + x // Don't assume 5

Generalization

def f(x: 5): 10 = x + x // it's 10! def f(x: Int): Int = x + x // Don't assume 5 def f[A <: Number](x: A): A = x + x // Don't assume integer

Generalization

def f(x: 5): 10 = x + x // it's 10! def f(x: Int): Int = x + x // Don't assume 5 def f[A <: Number](x: A): A = x + x // Don't assume integer def f[A: CommutativeRing](x: A): A = x + x // Don't assume numbers

Generalization

def f(x: 5): 10 = x + x // it's 10! def f(x: Int): Int = x + x // Don't assume 5 def f[A <: Number](x: A): A = x + x // Don't assume integer def f[A: CommutativeRing](x: A): A = x + x // Don't assume numbers def f[A: Ring](x: A): A = x + x // Don't assume \* is commutative

Generalization

def f(x: 5): 10 = x + x // it's 10! def f(x: Int): Int = x + x // Don't assume 5 def f[A <: Number](x: A): A = x + x // Don't assume integer def f[A: CommutativeRing](x: A): A = x + x // Don't assume numbers def f[A: Ring](x: A): A = x + x // Don't assume \* is commutative def f[A: Rng](x: A): A = x + x // Don't assume \* has identity

Generalization

def f(x: 5): 10 = x + x // it's 10! def f(x: Int): Int = x + x // Don't assume 5 def f[A <: Number](x: A): A = x + x // Don't assume integer def f[A: CommutativeRing](x: A): A = x + x // Don't assume numbers def f[A: Ring](x: A): A = x + x // Don't assume \* is commutative def f[A: Rng](x: A): A = x + x // Don't assume \* has identity def f[A: CommutativeGroup](x: A): A = x + x // Don't assume \* exists

Generalization

def f(x: 5): 10 = x + x // it's 10! def f(x: Int): Int = x + x // Don't assume 5 def f[A <: Number](x: A): A = x + x // Don't assume integer def f[A: CommutativeRing](x: A): A = x + x // Don't assume numbers def f[A: Ring](x: A): A = x + x // Don't assume \* is commutative def f[A: Rng](x: A): A = x + x // Don't assume \* has identity def f[A: CommutativeGroup](x: A): A = x + x // Don't assume \* exists def f[A: Group](x: A): A = x + x // Don't assume + is commutative

Generalization

def f(x: 5): 10 = x + x // it's 10! def f(x: Int): Int = x + x // Don't assume 5 def f[A <: Number](x: A): A = x + x // Don't assume integer def f[A: CommutativeRing](x: A): A = x + x // Don't assume numbers def f[A: Ring](x: A): A = x + x // Don't assume \* is commutative def f[A: Rng](x: A): A = x + x // Don't assume \* has identity def f[A: CommutativeGroup](x: A): A = x + x // Don't assume \* exists def f[A: Group](x: A): A = x + x // Don't assume + is commutative def f[A: Monoid](x: A): A = x + x // Don't assume + has an inverse

Generalization

def f(x: 5): 10 = x + x // it's 10! def f(x: Int): Int = x + x // Don't assume 5 def f[A <: Number](x: A): A = x + x // Don't assume integer def f[A: CommutativeRing](x: A): A = x + x // Don't assume numbers def f[A: Ring](x: A): A = x + x // Don't assume \* is commutative def f[A: Rng](x: A): A = x + x // Don't assume \* has identity def f[A: CommutativeGroup](x: A): A = x + x // Don't assume \* exists def f[A: Group](x: A): A = x + x // Don't assume + is commutative def f[A: Monoid](x: A): A = x + x // Don't assume + has an inverse def f[A: Semigroup](x: A): A = x + x // Don't assume + has identity

Generalization

def f(x: 5): 10 = x + x // it's 10! def f(x: Int): Int = x + x // Don't assume 5 def f[A <: Number](x: A): A = x + x // Don't assume integer def f[A: CommutativeRing](x: A): A = x + x // Don't assume numbers def f[A: Ring](x: A): A = x + x // Don't assume \* is commutative def f[A: Rng](x: A): A = x + x // Don't assume \* has identity def f[A: CommutativeGroup](x: A): A = x + x // Don't assume \* exists def f[A: Group](x: A): A = x + x // Don't assume + is commutative def f[A: Monoid](x: A): A = x + x // Don't assume + has an inverse def f[A: Semigroup](x: A): A = x + x // Don't assume + has identity def f[A: Magma](x: A): A = x + x // Don't assume + is associative

Generalization

def f(x: 5): 10 = x + x // it's 10! def f(x: Int): Int = x + x // Don't assume 5 def f[A <: Number](x: A): A = x + x // Don't assume integer def f[A: CommutativeRing](x: A): A = x + x // Don't assume numbers def f[A: Ring](x: A): A = x + x // Don't assume \* is commutative def f[A: Rng](x: A): A = x + x // Don't assume \* has identity def f[A: CommutativeGroup](x: A): A = x + x // Don't assume \* exists def f[A: Group](x: A): A = x + x // Don't assume + is commutative def f[A: Monoid](x: A): A = x + x // Don't assume + has an inverse def f[A: Semigroup](x: A): A = x + x // Don't assume + has identity def f[A: Magma](x: A): A = x + x // Don't assume + is associative def f[A](x: A): A = x // Assume nothing

Generalization

def f(x: 5): 10 = x + x // it's 10! def f(x: Int): Int = x + x // Don't assume 5 def f[A <: Number](x: A): A = x + x // Don't assume integer def f[A: CommutativeRing](x: A): A = x + x // Don't assume numbers def f[A: Ring](x: A): A = x + x // Don't assume \* is commutative def f[A: Rng](x: A): A = x + x // Don't assume \* has identity def f[A: CommutativeGroup](x: A): A = x + x // Don't assume \* exists def f[A: Group](x: A): A = x + x // Don't assume + is commutative def f[A: Monoid](x: A): A = x + x // Don't assume + has an inverse def f[A: Semigroup](x: A): A = x + x // Don't assume + has identity def f[A: Magma](x: A): A = x + x // Don't assume + is associative def f[A](x: A): A = x // Assume nothing

Generalization

def f(x: 5): 10 = x + x // it's 10! def f(x: Int): Int = x + x // Don't assume 5 def f[A <: Number](x: A): A = x + x // Don't assume integer def f[A: CommutativeRing](x: A): A = x + x // Don't assume numbers def f[A: Ring](x: A): A = x + x // Don't assume \* is commutative def f[A: Rng](x: A): A = x + x // Don't assume \* has identity def f[A: CommutativeGroup](x: A): A = x + x // Don't assume \* exists def f[A: Group](x: A): A = x + x // Don't assume + is commutative def f[A: Monoid](x: A): A = x + x // Don't assume + has an inverse def f[A: Semigroup](x: A): A = x + x // Don't assume + has identity def f[A: Magma](x: A): A = x + x // Don't assume + is associative def f[A](x: A): A = x // Assume nothing

Abstraction v. Generalization

Abstraction

SAGREDO: This is a marvelous idea! It suggests that when we try to understand nature, we should look at the phenomena as if they were messages to be understood. Except that each message appears to be random until we establish a code to read it. This code takes the form of an abstraction, that is, we choose to ignore certain things as irrelevant and we thus partially select the content of the message by a free choice.

-- Gödel, Escher, Bach: an Eternal Golden Braid

Someone else's definitions, my emphasis:

Abstraction is an emphasis on the idea, qualities and properties rather than the particulars (a suppression of detail).

Generalization is a broadening of application to encompass a larger domain of objects of the same or different type.

Definition golf

Abstraction is ignoring differences

Generalization is acknowledging differences

Abstraction: Bats, whales, mice, humans... mammals!

Generalization: It lays EGGS?

Abstraction, abstractly

— The suppression of irrelevant detail — A concept with a "platonic" definition — A precise "semantic level" — Neither overspecified nor underspecified

Abstraction

— Suppressing detail sounds like a quotient set — Partitioning elements into equivalence classes — all members in a class are "the same" — we position and dim the lights as we choose

Abstraction, concretely

— The set of natural numbers N — An equivalence relation P = "congruent modulo 1" — Equivalence classes: { 1, 3, ... }, { 2, 4, ... } — These classes even have names

Abstraction

— I have objects B, C, and D — Notice: B <: A, C <: A, D <: A — A abstracts B, C, and D

Generalization

— I have property R which classifies B — Notice: property S <: R classifies B, C, and D — S generalizes R — Note contravariance

Do these actually differ?

— The difference (in this conception) is the arrow — This duality pervades not only math but life itself — It is extension vs. intension

Extension <-> Intention

abstract generalize

unification anti-unification

data codata

being doing

Set[A] Indicator[A]

objects attributes

**Paul Phillips**

OContrarivariant

Stalactite, from Greek stalaktos 'dripping". Stalagmite, from Greek Stalagma "a drop'.

COClata VS. Data.

The Marriage of Extent and Intent

What is this?

What am I describing?

- It has wings. "A dragonfly?" - It isn't an insect. "A bat?" - It lays eggs. "A pterosaur?" - It isn't a dinosaur. "A platypus in a bird costume?" - It flies. "...in a hot air balloon?" - It has feathers. "A realistic avian drone?"

Imperfection

— We usually work with "objects" already in mind — Naturally we will select properties they share — So the flaws tend to be in the "negative space" — How much "slack" is there in our abstraction?

Bagel abstraction

Mop abstraction

Muffin abstraction

Towel abstraction

Abstraction abstractly

— Ideally, extent and intent would converge — An object set induces a common attribute set — An attribute set induces a common object set — When they correspond, we have something interesting — This is the idea behind formal concept analysis

Formal Concept Analysis (FCA)

FCA finds practical application in fields including...

— data and text mining — machine learning — knowledge management — the semantic web — software development — chemistry & biology

FCA (cont)

— Imagine facts as a matrix of booleans — Each row is an object — Each column is an a!ribute — Each cell is true if object has attribute — We can pre-filter as far as we like

This is an incidence matrix

This is its concept la!ice

tarn trickle ri beck riVulet runnel brook burn Streann

**ΟΓΙΘηί river channel Canal lagoon lake**

ΠΠΘΙΘ. plash pond pool puddle reservoir

**Sea**

plash, puddle

mere,

trickle, rill, river. d

pOIC1,

rivulet, runnel,

eSeVOll beck, brook, burn,

stream, torrent tarn, lake, pool

Formal Concepts

— A formal context is K = (G, M, I) where — G, a set of objects (Gegenstand = object) — M, a set of attributes (Merkmal = characteristic) — I, a binary relation GxM (Inzidenzrelation = incidence) — From the formal context we derive the concept la!ice

Formal Concepts (cont)

— For any set A ⊆ G and any set B ⊆ M — The attribute set common to objects in A is A' — The objects which have all attributes in B is B' — If A=B' and A'=B, then (A,B) is a formal concept.

Formal Concepts (cont)

— The concept lattice orders these formal concepts — The ordering is set inclusion — (A, B) is a subconcept of (A', B') if A ⊆ A' (and B' ⊆ B) — Extent is covariant and intent is contravariant — A subconcept has fewer objects and more attributes

Formal Concepts (cont)

— I⇡ : P(G) => P(M) (intersection of input attribute sets) — I⇣ : P(M) => P(G) (set of objects with all attributes) — Compose both ways to create closure operators: — I⇡⇣: P(G) => P(G) and I⇣⇡: P(M) => P(M) — Now for any set of objects A... — (I⇡I⇣(A), I⇡(A)) is a formal concept

Formal Concepts (cont)

— Given formal concept (A,B) — The objects in A are the concept's extent. — The attributes in B are the concept's intent. — There is a Galois correspondence between objects

and attributes.

Darwin's 'survival of the fittest' is really a special case of a more general law of survival of the stable.

It may be a class of entities that come into existence at a sufficiently high rate to deserve a collective name.

An EXTENDED ladder of functional programming

Profunctor intuition

Jean Bénabou, who invented the term and originally used “profunctor,” now prefers “distributor,” which is supposed to carry the intuition that a distributor generalizes a functor in a similar way to how a distribution generalizes a function. — n-lab

LL LLLL LLLLLL GLLLLLLL LLLL LLLLLL LLLL LLLLLLL LLLLLLL

SS S ZSSSuSuSSLL LLS S S SLSuSSuSuSS S SLLLS SLLL SLLLL SSLLS LLLLLLLLS L S SL T LLLL TM D TMS SDD TS SS SSLSSS S LLLL TT LL L S L S SSMSMMSMM S TTTLLLS T L S L D DD SDD LS S S TT S TM MT SMS S L TT S SSTS TMS SSST S TMM MT T LS SS LL S SSSS SSSSS S S L S S S L S L S SSS S SSSZ SSSSS S uu M STSTSS T S STS SS S SS SZYLSSSSL S SL TTLSS S S S S STT S SLLS S S S S S S S S S S S S S SSSTSTSS STT SS T TSS S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S SSSSSL

ா

title

until in

| | | | P |

Albert PETP

att fall PT PE \* Tip II, II, LO SLLLLL LL LLLLL LL LLL T L L L LS

LL L T S

**COEPITE**

era e alla

\*T冒匡\*

edited

SS TMLL S LLL LL

TEEL UIT DIE

FIRE ERUs

Rur she

un Abfalli (Cinnand han TTTTTTTTT

t

■■晶『\*』『疊

LL L SSSSS S S S S S S S S S S

wana wana antinuation, , , 『冒■鬥l■,■■■

CGCEPTE

PICTIC PE ப்ோப் ரிப்பா TIL FICTITring LEETE

The put Hist " " On Tull || || || || || ||

POLICIO EPIC

LIDRE ECTRE LLLLLL S LLLL LS SL L L L L L L L L L L S L L L L L S L L LSLS SLLLSL L S LSLSLSL LL LSLLLLLLDS

S L L L L L L L

SSSSSSSSSSSS S S S LS SLL LSS SLSS

SiSSSSSSL

S S S S S S S S S S S S SS SLS L S SZS S S S S S S S L S S S S S S S LLL S L L L L L L C S L L L

■■■■■\*\*直『■ L LLS L S L S S S T T S TS LLL LLLLLS S S LLLLL auS TT S L L LSLTTS S S S L L L L L S S S L LLLL SZS S L S L L L

**EILLE**

S דמיוחדיםםחםםחם פחםחםחשיחים שחששוחחיםם חשוחח ח

S S S S S L S S S S S S C S SS SSSS SSS S S S S S S S S S S S S SLLSS S

SS S S S S S S S S SS S S S S S S S S S S

S S S S LSLS SS SS

S SSSSSSSSSSSSSSS LSSSSSSLSTSS S SSSSSSS SS SSLS SSSSS TTSS S SSS

**ls**

SS SS S S S S S S S S S LL S S L L S L L S S L SLL L S LSLS SLSSLSSSSSSSSSLLSSS SS SSLLL L S S S L SLLS SLLLSS S SSS S SSSLLLLSLLSS SSS SSSuuSSSSSS SSSuuuuSSSSS SSSLSSSSS SSSSS LLS L S S S S S LSL L LSS S SY L L S S

SS SSSSSSSSSSSSSSSSS L L L L S L S S CCCC C LS

L L L L L L

**s**

S S S S S S S S S S S S LLLSS

SS SSSS SS SS SSLLLLSLSSLLSLSSLS S SS

LLSL S SZS S u S S S S S u u u SSS S LLS

LSSLSS LSSSLSS SSSL S SSSS S LYS SSSL S S S SS C SSL L L LLLLLS L LLLL S SS LL S S L S S L S SSS uuS LLL uCCSS S LCCSLSS SLSS S uuS S LL LSLLLLL S L L L L L L L L LLLL SSLSS S LSSSSLSSSSZSSSZSSSSSS S S SSS S SSS S SS SS SS SSSSSSSSSS Su SuSHHSSSSS SSSLLLLLSSS LLSSSSSLSSSSSLSSSSSSLSSS LSS SLS SS SSSSSLSS S SuSuuSSSSSS LL S S L L S TT S L S T

**FILM**

- | - - 0 S-1-

**CONCEPTS**

COOOO

co) Recursion Schemes Advonced Optics Duo Abstractions (Comonod) Monolo Trons for ners

Free Moncos S. Extensible Effects

Function o ATCite CiU e A CVCCGC FUNCIOS Exponential ProfUnctors. COntrOVOriant) Embedded DSLS USing GADTS. Finoly Togess Advanced Monads (continuation, Logic) Type Fonies, Functiono Dependencies

O CO

**CONCEPTS**

High-Performance Kind Polynorphism Generic Programming Type - Le Vel Programming Dependent-Types, Singleton Types COtegory. Theory Graph Reduction Higher-Order Abstract Syntox Compiler Design for Functional Longudges Profunctor Optics

Groph Reduction Higher-Order AbStroC

Compiler Design for F. Profunctor Optics

Several of you will have realised that the relation I can be viewed as a profunctor enriched in truth values, that the Galois connection is an adjunction between presheaf-type enriched categories and the concept lattice is the centre of

the adjunction, i.e. the nucleus of the profunctor.

Welcome to the SECRET RUNG

"AXE-WIELDING FIRE MONSTER" Profunctor Nuclei NOTHING MORE TO LEARN