LambdaConf 2017

Krystal Maughan

May 27th 2017

Day Three

1 Lambda Calculus

The Wizard Towel

Identity Function $\lambda x.x$
x is the function body
λx is the head
Takes an argument, binds the value to x and returns it
In Haskell : \ x $\rightarrow x$

Typed vs untyped lambda calculus
Alpha Equivalence $\lambda x.x \equiv \lambda y.y$
$(\lambda x.x)2$
Simplification *beta* reduction

1. Bind the argument to the name
2. Strip off the function head and iterator
3. Simplify until we can't anymore.

λ 2. 2
Head: λ 2
Body: 2
Identity Simplification: 2
(λ x . x)(λ y . y)

Lambda Calculus: Every function can only take one argument
λ x.(λ y. x y)

Converting a function with multiple args → currying

See John Carmac porting Haskell talk Wolf 3d

$(\lambda y.(\lambda z.z)y)$

Types
const :: a → b → a
take in one value, take in anothe value, and return the first

:k type (kind of type)
Lambda Calculus is a Turing Tarpit

Look up talk on : Lisp Interpreter in AWK
(another turing tarpit → awk)

2 Omega Combinator

$(\lambda x.xx)(\lambda y.yy)$

Beta Reduction
Alpha Conversion
What we need is the Y combinator
$Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$

3 What are Dependent Types

Stephan Boyer

The Lambda Cube

4 Curry-Howard Isomorphism

Coq

System F: Polymorphism
Functions from types to terms
$\forall X.T$

System $\lambda\omega$
Kind syntax(K):
functions from * → *
System LF: dependent types
$\Pi x : T.K$

Curry-Howard Isomorphism
Type Theory vs Logic
eg Sum Type vs Disjunction

5 Coq

GADTs

conj : P → Q we have P ∧ Q

Definition (¬ A → False)
0: nat S: nat → nat. Defined all natural numbers in Coq.

6 Higher Order Abstractions

Stephan Boyer

7 Functor

supports fmap
Takes function a → b fa → fb

8 Bifunctor

Has two type parameters instead of one
bimap :: (a → b) → (c → d)
Either is a bifunctor

9 Contravariant

Looks like fmap.
take fb and returns fa

Predicate is example of Contravariant
contramap :: (a → b) → $fb \rightarrow fa$

$a \rightarrow (a \rightarrow Int) \rightarrow Int)$: Invariant
data Phantom a = Phantom : Bivariant

10 Profunctor

Hungry for bs and contains cs
Dimap
Lens, Prism

11 Haskell Singletons and You

Justin Le (mstksg.github.io)
Phantom Types
closeDoor :: Door 'Opened → Door 'Closed
closeDoor UnsafeMkDoor = UnsafeMkDoor
can only open a close door. Compile type otherwise
type can't depend on your inputs
In Haskell, types only exist at runtime.
They are erased at runtime.

12 Singleton Pattern

Only one value type : we can pattern match
showSingDS :: SingDS s → String
SOpened → "Opened"
SClosed → "Closed"
SLocked → "Locked"

Singletons are Poly-kinded typeclasses.

Further Reading:
https://blog.jle.im/entry/verified-instances-in-haskell.html

13 The Origins of Free

Adam Warski

Free Monads:

Separate Description from Interpretation
Programmes as values

14 Universal Algebra : study of general algebraic structures

Syntax:
Type names $\rightarrow$ set S
Operation Names: family Ω of sets indexed by S* $\times$ S

Interpretation of signature
Function between appropriate sets
Σ algebra A:
$succ_A = \lambda.\lambda x + 1$
F-Algebra

Term Algebra : built of pure syntax

Algebra: defined inductively
base: all constants are terms
Step: any functions we can apply on previous terms
Base case: [0]
$[[0], [0 + succ], etc]$

15 Homomorphism

A Function between algebras
A function between type interpretations such
that operations are preserved

Σ-algebra I is initial when any other
Σ-algebra A there is exactly
one homomorphism from I to A

$f : T_\Sigma \to A$
$f(0_{T_\Sigma}) = 0_A$

16 Initial Algebra

Algebra: Only one way to interpret a term
no junk: term algebra contains only what's necessary

Terms with Variables
For any set $X, T_\Sigma(X)$ is the term algebra with X
added as constants (but called variables)

17 Free Algebra

Σ-algebra I is free over X when for any other
Σ-algebra A, any function
$f : X \to A$
extends uniquely to a homomorphism

Free:
free to interpret in any way
no constraints
free of additional structure
only contains what is necessary

Term Algebras with Equations
some terms need to be combined
Equivalence relation generated by ϕ: $\equiv_\phi$
Homomorphism to other algebra
free object is unique up to an isomorphism

18 Monads

Monads return a pure value
Compute what to do next based on previous result (flatMap)

19 Free Monads

Signature: pure and Flatmap
Variables: operations (our Domain Specific Language)

20 Free in Haskell

data Free f r = Free(f (Free f r)) | Pure r

21 Reference

Foundations of Algebraic Specification and
Formal Software Development
Donald Sannella and Andrzej Tarlecki

22 Refactoring Recursion

Harold Carr

Factor recursion out of data with
Foldable, Traversable, Fix
Recursion over data
cata, para(cata++), histo, zygo/mutu(para++)
both: hylo
corecursion: hylo

Fold left 0 is top element
Fold right 0 is bottom of tree

23 Recursion Schemes

Sub-Categories

24 Catamorphism

catamorphism : means downwards : aka fold

25 Anamorphism

Corecursion, meaning upwards : aka unfold
:: b, → (a, b)→ which it operates on b → [a]

Anamorphism : building something up from a seed analogy

26 Hylomorphism

Anamorphism, then catamorphism
Composition of catamorphism and anamorphism
hyloL :: (a $\rightarrow c \rightarrow c$)
corecursive codata production
followed by recursive data consumption
run co-algebra first build up list from n to 0
go across list and use catamorphism
eg factorial goes from n to zero, then applies (*)

27 Fusion/ Deforestation

Talk on that..later

28 Paramorphism

Beside or parallel with
Extension of Catamorphism
paraL :: (a $\rightarrow [a] \rightarrow b \rightarrow b$)
eg Tails library
Sliding windows
$(slide2[1..5])$
$[[1, 2, 3], [2, 3, 4], [3, 4, 5]]$

29 Apomorphism

Apo meaning apart
dual of parammorphism
extension of anamorphism
enables short-circuiting traversal

apoL :: ([b] $\rightarrow$ Maybe(a, Either [b] [a]))

30 Zygomorphism

Generalization of paramorphism
zygoL :: $(a \rightarrow b \rightarrow b) \rightarrow (a \rightarrow b \rightarrow c \rightarrow c)$
$b \rightarrow c$
$[a]$
c
eg. Every third element I need to do...

L means List in slides

pmL :: $[Int] \rightarrow [Int]$
pmL = zygoL ($\backslash_b \rightarrow$ not b)

31 Histomorphism

Gives access to previously computed values
(see photo data History a b)
histoL :: $(a \rightarrow History$ a b $\rightarrow b)$
$\rightarrow b \rightarrow [a] \rightarrow b$

works bottom up on structure
Histomorphism useful for dynamic programming

32 Futurmorphism

Corecursive dual of histomorphism
(see photo)

33 Refactoring Recursion out of data

deriving Foldable

34 References

Tim Williams' Recursion Schemes presentation
http://www.timphilipwilliams.com/slides.html
Functional Programming with Banana, Lenses, Envelopes and Barbed Wire

35 Intuitionistic Logic III: Subschemas and Topologies

Vlad Patryshev
N is a category of natural numbers
A^B category of functors from B to A aka diagrams
Set^2 category of 2 diagrams in sets - two sets, one function
$S_0 \to S_1 \to S_2$
Temporal Logic for Applications - Leslie Lamport

Ternary Logic: Two-State Sets
"before" and "after"

Subobject Classifier :
(see Slides)

36 Grothendieck Topology

J is the closure of truth in Ω

At some point will be true : diamond classification

Groundhog Week : is a groupoud : Boolean Topology

37 References

P.T.Johnstone "Topos Theory"
Ceciia Fiori "A First Course in Topos Quantum Theory"
Introduction to Intuitionistic Type Theory
Intuitionistic Logic of Database Schema

38 I Command you to be Free

Matt Parsons