Introducing type theory

Simon Thompson
School of Computing, Kent

June 2016

Foundations of maths … back a century

Foundational system with objects in "types": Russell and Whitehead

Set theory: Zermelo and Fraenkel

Foundations of logic: Hilbert's axioms and Prawitz natural deduction

Church's foundations: λ-calculus, a theory of functions

Constructive mathematics … propositions as types: Martin-Löf.

Foundations of maths … to now

Foundational system with objects in "types": Russell and Whitehead

Set theory: Zermelo and Fraenkel

Foundations of logic: Hilbert's axioms and Prawitz natural deduction

Church's foundations: λ-calculus, a theory of functions

Constructive mathematics … propositions as types: Martin-Löf.

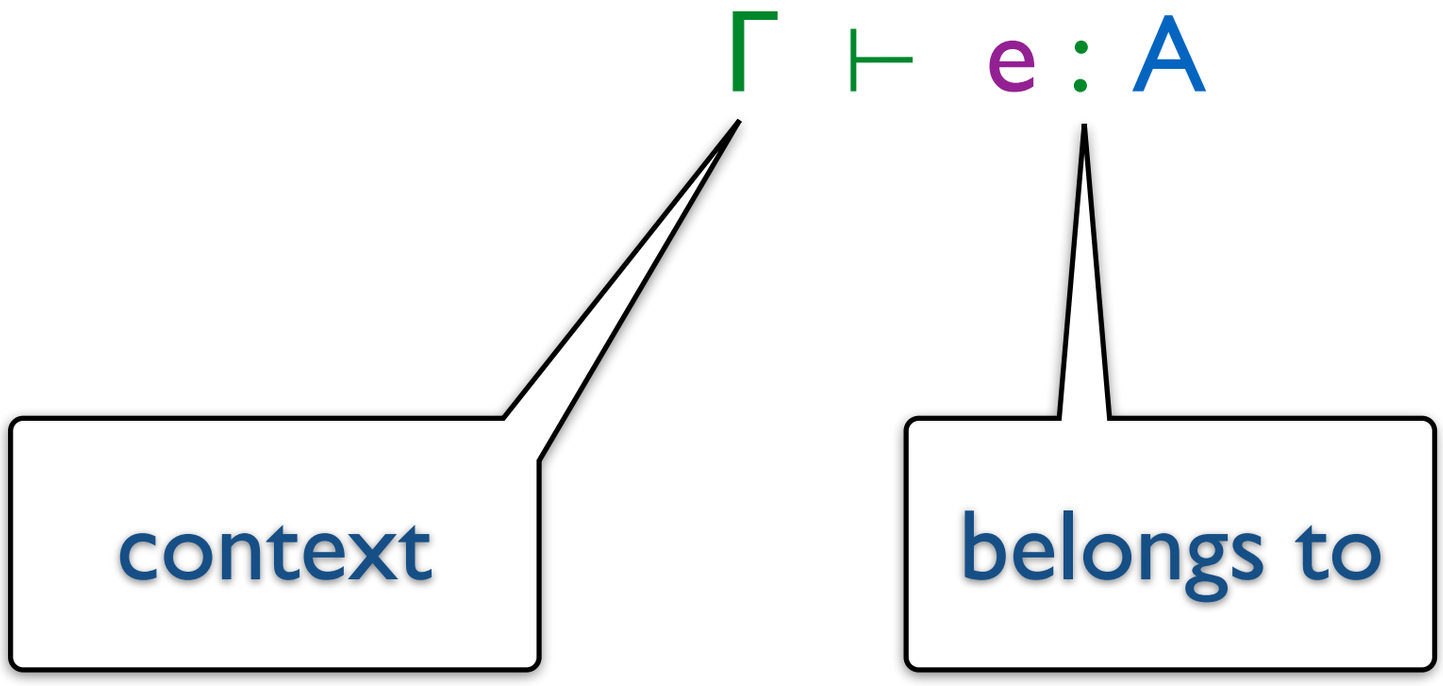
Category theory: a language for type theoretic foundations.

Homotopy theory: topological mechanism for characterising shape.

Judgements

$$\Gamma \vdash e : A$$

Judgements



Rules

$$\frac{\Gamma \vdash x : A \qquad \Gamma \vdash y : B}{\Gamma \vdash (x,y) : A \wedge B}$$

Rules

$$\frac{\Gamma \vdash x : A \wedge B}{\Gamma \vdash \mathrm{fst}(x) : A}$$

Derivations

$$\Gamma \vdash y : B \wedge A \qquad \Gamma \vdash y : B \wedge A$$

where $\Gamma \equiv y : B \wedge A$

Derivations

$$\frac{\Gamma \vdash y : B \wedge A}{\Gamma \vdash \mathsf{snd}(y) : A} \qquad \frac{\Gamma \vdash y : B \wedge A}{\Gamma \vdash \mathsf{fst}(y) : B}$$

where $\Gamma \equiv y : B \wedge A$

Derivations

$$\frac{\Gamma \vdash y : B \wedge A}{\Gamma \vdash snd(y) : A} \qquad \frac{\Gamma \vdash y : B \wedge A}{\Gamma \vdash fst(y) : B}$$

$$\Gamma \vdash (snd(y),fst(y)) : A \wedge B$$

where $\Gamma \equiv y : B \wedge A$

Derivations

$$\Gamma \vdash \quad ??? \quad : A \wedge B$$

where $\Gamma \equiv y : B \wedge A$

Derivations

$$\frac{\Gamma \vdash \; ??? \; : A \qquad \Gamma \vdash \; ??? \; : B}{\Gamma \vdash \; ??? \; : A \wedge B}$$

where $\Gamma \equiv y : B \wedge A$

Derivations

$$\frac{\Gamma \vdash y : B \wedge A}{\Gamma \vdash \; ??? \; : A} \qquad \frac{\Gamma \vdash y : B \wedge A}{\Gamma \vdash \; ??? \; : B}$$

$$\Gamma \vdash \; ??? \; : A \wedge B$$

where $\Gamma \equiv y : B \wedge A$

Derivations

$$\frac{\Gamma \vdash y : B \wedge A}{\Gamma \vdash \mathsf{snd}(y) : A} \qquad \frac{\Gamma \vdash y : B \wedge A}{\Gamma \vdash \mathsf{fst}(y) : B}$$

$$\frac{}{\Gamma \vdash \quad ??? \quad : A \wedge B}$$

where $\Gamma \equiv y : B \wedge A$

Derivations

$$\frac{\Gamma \vdash y : B \wedge A}{\Gamma \vdash snd(y) : A} \qquad \frac{\Gamma \vdash y : B \wedge A}{\Gamma \vdash fst(y) : B}$$

$$\Gamma \vdash (snd(y),fst(y)) : A \wedge B$$

where $\Gamma \equiv y : B \wedge A$

Rules

$$\frac{A \qquad B}{A \wedge B}$$

Rules

$$\frac{A \wedge B}{A}$$

Derivations

$$\frac{B \wedge A}{A} \qquad \frac{B \wedge A}{B}$$

$$\frac{A \quad B}{A \wedge B}$$

Proof in natural deduction

Typically in logic we're interested in whether there is a proof of a particular proposition, e.g. $A \wedge B$ …

… rather than what the proof is.

Introduction rules

$$\frac{\Gamma \vdash x : A \qquad \Gamma \vdash y : B}{\Gamma \vdash (x,y) : A \wedge B}$$

Elimination rules

$$\frac{\Gamma \vdash x : A \wedge B}{\Gamma \vdash \mathrm{fst}(x) : A}$$

Evidence in natural deduction

Suppose we turn to whether these is evidence of whether a particular proposition, e.g. $A \wedge B$ holds …

… then we might be interested in how the evidence simplifies.

Simplification?

$$\frac{\Gamma \vdash x : A \qquad \Gamma \vdash y : B}{\Gamma \vdash (x,y) : A \wedge B}$$

$$\frac{\Gamma \vdash (x,y) : A \wedge B}{\Gamma \vdash \mathrm{fst}((x,y)) : A}$$

Computation rules

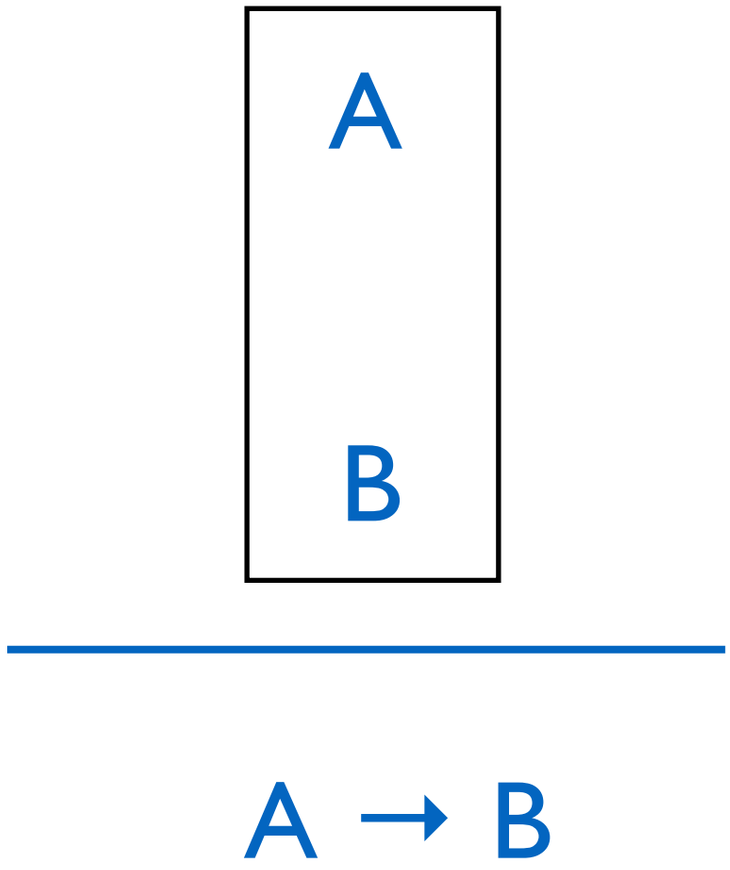
$$\mathrm{fst}((x,y)) \rightsquigarrow x$$

Computation rules

$$\mathrm{fst}((x,y)) \rightsquigarrow x$$

$$\mathrm{snd}((x,y)) \rightsquigarrow y$$

What about contexts?



Transforming evidence

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x.e : A \to B}$$

Modus ponens: apply the transformation

$$\frac{\Gamma \vdash f : A \qquad \Gamma \vdash e : A \to B}{\Gamma \vdash ef : B}$$

Computation rules

$$(\lambda x.e)a \rightsquigarrow e[a/x]$$

Programs in a programming language

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x.e : A \to B}$$

$$\frac{\Gamma \vdash f : A \qquad \Gamma \vdash e : A \to B}{ef : B}$$

$$(\lambda x.e)a \rightsquigarrow e[a/x]$$

Proofs in a logic

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x.e : A \to B}$$

$$\frac{\Gamma \vdash f : A \qquad \Gamma \vdash e : A \to B}{ef : B}$$

$$(\lambda x.e)a \rightsquigarrow e[a/x]$$

The Curry-Howard isomorphism

Conjunction	$\wedge$	Product type
Implication	$\rightarrow$	Function type
Disjunction	$\vee$	Disjoint union type
Trivial	$\top$	One element type
Absurd	$\bot$	Empty type
Universally quantified	$\forall$	Dependent function space
Existentially quantified	$\exists$	Dependent product
Natural numbers / induction	$\mathbb{N}$	Natural numbers / recursion
Inductive predicates	$\mathfrak{I}$	Inductively defined types
Co-inductive predicates	$\mathfrak{C}$	Co-inductively defined types
Identity predicates	$\mathfrak{J}$	Identity types

$\mathbb{N}$ introduction rules

$$\frac{}{\Gamma \vdash 0 : \mathbb{N}}$$

$$\frac{\Gamma \vdash e : \mathbb{N}}{\Gamma \vdash s(e) : \mathbb{N}}$$

$\mathbb{N}$ elimination rule (recursion)

$$\frac{\Gamma \vdash n : \mathbb{N} \qquad \Gamma \vdash c : C \qquad \Gamma \vdash f : N \to C \to C}{\Gamma \vdash \text{prim } n\ c\ f : C}$$

$\mathbb{N}$ elimination rule (induction)

$$\Gamma \vdash n{:}\mathbb{N} \quad \Gamma \vdash c : C(0) \quad \Gamma \vdash f : (\forall n{:}N)(C(n) \to C(s(n)))$$

$$\Gamma \vdash \text{prim } n\ c\ f : C(n)$$

$\mathbb{N}$ elimination rule (induction)

$$\frac{\Gamma \vdash n{:}\mathbb{N} \quad \Gamma \vdash c : C(0) \quad \Gamma \vdash f : (\forall n{:}N)(C(n) \rightarrow C(s(n)))}{\Gamma \vdash \text{prim } n\ c\ f : C(n)}$$

$$\frac{\Gamma \vdash \text{prim } n\ c\ f : C(n)}{\Gamma \vdash \lambda x.\text{prim } x\ c\ f : (\forall x{:}N)(C(x))}$$

$\mathbb{N}$ computation rules

$$\text{prim } 0\ c\ f \rightsquigarrow c$$

$$\text{prim } s(n)\ c\ f \rightsquigarrow f\ n\ (\text{prim } n\ c\ f)$$

What does this have to say for logic?

Initial insight … the logic is constructive

- Choice of which disjunct holds

- Witness for existential

- Axiom of choice

What does this have to say for logic?

Initial insight … the logic is constructive

- Choice of which disjunct holds
- Witness for existential
- Axiom of choice

But that's too simplistic

- sub-structural logics
- computational interpretation of classical logic

What about equality, identity, … ?

We say 2 terms are *convertible* if the (reflexive, symmetric, transitive, congruence) closure of $\rightsquigarrow$ identifies them.

What about equality, identity, … ?

We say 2 terms are *convertible* if the (reflexive, symmetric, transitive, congruence) closure of $\rightsquigarrow$ identifies them.

That's not a *logical* identity: we can't work with it hypothetically. Add identity predicates/types.

What about equality, identity, … ?

We say 2 terms are *convertible* if the (reflexive, symmetric, transitive, congruence) closure of $\rightsquigarrow$ identifies them.

That's not a *logical* identity: we can't work with it hypothetically. Add identity predicates/types.

What do we need to

- be able to program?

- be able to reason?

- be able to do both: reason about programs?

- do mathematics in a natural way?

What does this have to say for programming?

Enriched programming experience: types can embody properties.

mult : $(\forall n,m,p{:}\mathbb{N})(\mathrm{Mat}(n,m) \to \mathrm{Mat}(m,p) \to \mathrm{Mat}(n,p))$

What does this have to say for programming?

Enriched programming experience: types can embody properties.

mult : $(\forall n,m,p{:}\mathbb{N})(\mathrm{Mat}(n,m) \to \mathrm{Mat}(m,p) \to \mathrm{Mat}(n,p))$

sort : $(\forall xs{:}\mathrm{List}(\mathbb{N}))(\exists ys{:}\mathrm{List}(\mathbb{N}))(\mathrm{Ordered}(ys) \wedge \mathrm{Perm}(xs,ys))$

What does this have to say for programming?

Enriched programming experience: types can embody properties.

mult : $(\forall n,m,p{:}\mathbb{N})(\mathrm{Mat}(n,m) \to \mathrm{Mat}(m,p) \to \mathrm{Mat}(n,p))$

sort : $(\forall xs{:}\mathrm{List}(\mathbb{N}))(\exists ys{:}\mathrm{List}(\mathbb{N}))(\mathrm{Ordered}(ys) \wedge \mathrm{Perm}(xs,ys))$

where we can define the property Perm inductively like this:

refl : $(\forall xs{:}\mathrm{List}(\mathbb{N}))\mathrm{Perm}(xs,xs)$

cons : $(\forall xs,ys{:}\mathrm{List}(\mathbb{N}))(\forall z{:}\mathbb{N})(\mathrm{Perm}(xs,ys) \to \mathrm{Perm}(z::xs,z::ys))$

pair : $(\forall xs{:}\mathrm{List}(\mathbb{N}))(\forall y,z{:}\mathbb{N})(\mathrm{Perm}(xs,ys) \to \mathrm{Perm}(y::z::xs,z::y::xs))$

trans : $(\forall xs,ys,zs{:}\mathrm{List}(\mathbb{N}))(\mathrm{Perm}(xs,ys) \to \mathrm{Perm}(ys,zs) \to \mathrm{Perm}(xs,zs))$

Type theories are being used

Proof assistants:

- current: Coq, Agda, Isabelle, …,

- and indeed historical: Lego, Nuprl, Alf, …

Type theories are being used in practice

Proof assistants:

- current: Coq, Agda, Isabelle, …,

- and indeed historical: Lego, Nuprl, Alf, …

Programming languages:

- true type-theories: Idris, Agda, …

- influencing: Haskell, Scala, …

A personal coda

The lambda calculus (λ-calculus)

A theory of functions.

Variables x, y, z.

Application (ef): e.g. ((xy)y).

Abstraction λx.e: e.g. λx.λy.λz.(xz)(yz)

The lambda calculus (λ-calculus)

A theory of functions.

Variables x, y, z.

Application (ef): e.g. ((xy)y).

Abstraction λx.e: e.g. λx.λy.λz.(xz)(yz)

β-reduction: (λx.e)f → e[f/x]

e[f/x] substitute f for x in e; rename variables so no variable capture.

The lambda calculus (λ-calculus)

A theory of functions.

Variables x, y, z.

Application (ef): e.g. ((xy)y).

Abstraction λx.e: e.g. λx.λy.λz.(xz)(yz)

β-reduction: (λx.e)f → e[f/x]

e[f/x] substitute f for x in e; rename variables so no variable capture.

Assume: application left associative and binds more tightly than λ.

Bracket abstraction

All terms of the λ-calculus can be represented using the combinators S, K (and I).

$S \equiv \lambda x.\lambda y.\lambda z.(xz)(yz) \quad K \equiv \lambda x.\lambda y.x \quad I \equiv \lambda x.x \equiv SKK$

Bracket abstraction

All terms of the λ-calculus can be represented using the combinators S, K (and I).

$S \equiv \lambda x.\lambda y.\lambda z.(xz)(yz) \quad K \equiv \lambda x.\lambda y.x \quad I \equiv \lambda x.x \equiv SKK$

Proof is by induction over the formation of λ-terms:

$[x]x \equiv I \qquad [x]y \equiv Ky$

$[x]ef \equiv S([x]e)([x]f)$

$S([x]e)([x]f)z \rightarrow (([x]e)z)(([x]f)z) \equiv (\lambda x.ef)z$ by induction.

Implicational logic

Propositional logic with $\to$ …

Two axiom schemes

$$A \to (B \to A)$$

$$A \to (B \to C) \to ((A \to B) \to (A \to C))$$

and one rule, *modus ponens*:

$$\frac{A \to B \qquad A}{B}\ \mathrm{MP}$$

The deduction theorem

$A \to (B \to A)$

$A \to (B \to C) \to ((A \to B) \to (A \to C))$

If $\Gamma, A \vdash B$ then $\Gamma \vdash A \to B$

Proof of the theorem is by induction on the (size of the) proof of B.

First base case, $A \equiv B$. Use 2 instances of 1st axiom, one of 2nd.

$A \to (A \to A)$

$A \to ((A \to A) \to A)$

$A \to ((A \to A) \to A) \to ((A \to (A \to A)) \to (A \to A))$

$$\dfrac{\dfrac{\text{axiom}\quad (A \to ((A \to A) \to A)) \to ((A \to (A \to A)) \to (A \to A)) \qquad \text{axiom}\quad A \to ((A \to A) \to A)}{(A \to (A \to A)) \to (A \to A)}\ \text{MP} \qquad \text{axiom}\quad (A \to (A \to A))}{A \to A}\ \text{MP}$$

The deduction theorem

$A \to (B \to A)$

$A \to (B \to C) \to ((A \to B) \to (A \to C))$

2nd base case, $B \in \Gamma$. Use one instance of 1st axiom: $B \to (A \to B)$

assum axiom

$$\frac{B \qquad B \to (A \to B)}{(A \to B)}\ \text{MP}$$

The deduction theorem

$A \to (B \to A)$

$A \to (B \to C) \to ((A \to B) \to (A \to C))$

2nd base case, $B \in \Gamma$. Use one instance of 1st axiom: $B \to (A \to B)$

assum axiom

$$\frac{B \qquad B \to (A \to B)}{(A \to B)}\ \text{MP}$$

Induction. Last step in proof is MP: inferring C from $B \to C$ and B

$$\dfrac{\dfrac{\text{axiom}\quad (A \to (B \to C)) \to ((A \to B) \to (A \to C)) \qquad \text{by induction}\quad (A \to (B \to C))}{(A \to B) \to (A \to C)}\ \text{MP} \qquad \text{by induction}\quad (A \to B)}{(A \to C)}\ \text{MP}$$

Putting the two together

Types for the values in the λ-calculus: only form *well-typed* terms

$$\frac{f : A \qquad e : (A \to B)}{(ef) : B}$$

Contexts / assumptions … if $x : A$ then $e : B$

$$\frac{x : A \;\vdash\; e : B}{\lambda x.e : A \to B}$$

Types and terms

$S \equiv \lambda x.\lambda y.\lambda z.(xz)(yz) : A \to (B \to C) \to ((A \to B) \to (A \to C))$

$K \equiv \lambda x.\lambda y.x \quad : A \to (B \to A)$

$I \equiv \lambda x.x \equiv SKK \quad : A \to A$

Deduction theorem shows that all proofs using an assumption (that is all λ-terms) can be built using S, K (and I).

Proofs ≡ Values

Propositions ≡ Types

Looking forward …

Homotopy theory: topological mechanism for characterising shape.

TT+HT: novel characterisation of (formerly problematic) equality.

New foundations for (informal) maths: the univalence axiom.

Category theory: a language for type theoretic foundations.

Induction / co-induction: complex definitional principles.