Logic As A Tool For Building Theories

Samson Abramsky
Oxford University Computing Laboratory

Introduction

Practice-based Philosophy of Logic?

Practice-based Philosophy of Logic?

What is the practice?

Practice-based Philosophy of Logic?

What is the practice?

Huge effect of Computer Science on Logic over the past 5 decades:

- new ways of **using** logic
- new attitudes to logic
- new questions
- new methods

Practice-based Philosophy of Logic?

What is the practice?

Huge effect of Computer Science on Logic over the past 5 decades:

- new ways of **using** logic
- new attitudes to logic
- new questions
- new methods

Hence new perspective on the question:

$\boxed{\text{What logic is — and should be!}}$

The Plan

Before (and while) trying to extract general points, some case studies:

Before (and while) trying to extract general points, some case studies:

- modal and temporal logic: verification and model-checking

The Plan

Before (and while) trying to extract general points, some case studies:

- modal and temporal logic: verification and model-checking
- λ-calculus

The Plan

Before (and while) trying to extract general points, some case studies:

- modal and temporal logic: verification and model-checking
- λ-calculus
- coalgebra

Before (and while) trying to extract general points, some case studies:

- modal and temporal logic: verification and model-checking
- λ-calculus
- coalgebra

Aims: not to put forward any philosophical theses, but to provide some materials and raise some questions.

Case Study I: Modal and Temporal Logic

Changing Perspectives

Changing Perspectives

- From ‘philosophical logic’ to computer-assisted verification. From metaphysics to (not just potentially but **actually**) applied mathematics.

Changing Perspectives

- From ‘philosophical logic’ to computer-assisted verification. From metaphysics to (not just potentially but **actually**) applied mathematics.
- From the ‘sacred’ to the ‘profane’. From logic as Guardian of The Truth to logic out in the world, to be used as a tool for understanding many aspects of our world.

Changing Perspectives

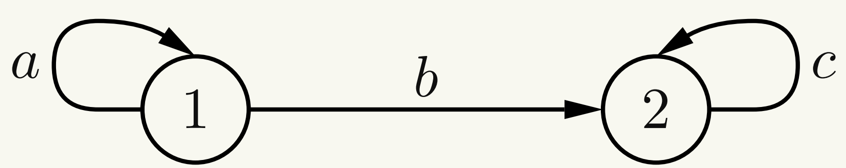
- From ‘philosophical logic’ to computer-assisted verification. From metaphysics to (not just potentially but **actually**) applied mathematics.
- From the ‘sacred’ to the ‘profane’. From logic as Guardian of The Truth to logic out in the world, to be used as a tool for understanding many aspects of our world.
- More concretely: from ‘possible worlds’ and ‘accessibility’ to **states** and **transitions**. Systems of states evolving under discrete transitions turn up in a huge variety of situations. (Communications protocols, hardware circuits, software, nowadays biological and physical systems . . .). Modal and temporal logics are canonical formalisms for expressing and reasoning about properties of such systems.

- From ‘philosophical logic’ to computer-assisted verification. From metaphysics to (not just potentially but **actually**) applied mathematics.

- From the ‘sacred’ to the ‘profane’. From logic as Guardian of The Truth to logic out in the world, to be used as a tool for understanding many aspects of our world.

- More concretely: from ‘possible worlds’ and ‘accessibility’ to **states** and **transitions**. Systems of states evolving under discrete transitions turn up in a huge variety of situations. (Communications protocols, hardware circuits, software, nowadays biological and physical systems . . .). Modal and temporal logics are canonical formalisms for expressing and reasoning about properties of such systems.

A transition system:



Some Issues

Some Issues

- Note the reverse engineering here. Historically, formal systems of modal logic were developed to study notions of necessity. Then Kripke semantics was developed to shed light on these formal systems. Now we think of the structures as the naturally occurring objects of study, the logics as tools for reasoning about them.

Some Issues

- Note the reverse engineering here. Historically, formal systems of modal logic were developed to study notions of necessity. Then Kripke semantics was developed to shed light on these formal systems. Now we think of the structures as the naturally occurring objects of study, the logics as tools for reasoning about them.

- New questions: algorithmic feasibility. New methods:

Some Issues

- Note the reverse engineering here. Historically, formal systems of modal logic were developed to study notions of necessity. Then Kripke semantics was developed to shed light on these formal systems. Now we think of the structures as the naturally occurring objects of study, the logics as tools for reasoning about them.

- New questions: algorithmic feasibility. New methods:

  - **model-checking**. Given a system description (a transition system) S, and a property ϕ we wish the system to satisfy, check if $S \models \phi$. This has become an enormously influential paradigm over the past 25 years. Much of the real value lies in cases where the property is **not** satisfied, and we get a **trace** which can lead us to the bug.

Some Issues

- Note the reverse engineering here. Historically, formal systems of modal logic were developed to study notions of necessity. Then Kripke semantics was developed to shed light on these formal systems. Now we think of the structures as the naturally occurring objects of study, the logics as tools for reasoning about them.

- New questions: algorithmic feasibility. New methods:

  - **model-checking**. Given a system description (a transition system) S, and a property ϕ we wish the system to satisfy, check if $S \models \phi$. This has become an enormously influential paradigm over the past 25 years. Much of the real value lies in cases where the property is **not** satisfied, and we get a **trace** which can lead us to the bug.
  - The automata-theoretical paradigm. Encode formulas as automata, reduce satisfaction to language inclusion, ultimately to graph reachability.

Some Issues

- Note the reverse engineering here. Historically, formal systems of modal logic were developed to study notions of necessity. Then Kripke semantics was developed to shed light on these formal systems. Now we think of the structures as the naturally occurring objects of study, the logics as tools for reasoning about them.

- New questions: algorithmic feasibility. New methods:

  - **model-checking**. Given a system description (a transition system) S, and a property ϕ we wish the system to satisfy, check if $S \models \phi$. This has become an enormously influential paradigm over the past 25 years. Much of the real value lies in cases where the property is **not** satisfied, and we get a **trace** which can lead us to the bug.
  - The automata-theoretical paradigm. Encode formulas as automata, reduce satisfaction to language inclusion, ultimately to graph reachability.

- Huge expansion to cover real-time, probabilistic and hybrid systems, and of applications to include biological systems, security, networks, agent-based modelling, control systems etc.

Some Remarks and Questions for P-B PoL

Some Remarks and Questions for P-B PoL

- The attitude that is is just 'grubby engineering' — engineers in overalls infringing on the sacred groves — just won't wash.

- The attitude that is is just 'grubby engineering' — engineers in overalls infringing on the sacred groves — just won't wash. (Cf. 17th century vis-à-vis the Greeks. McCarthy!).

Some Remarks and Questions for P-B PoL

- The attitude that is is just 'grubby engineering' — engineers in overalls infringing on the sacred groves — just won't wash. (Cf. 17th century vis-à-vis the Greeks. McCarthy!).
- Concrete interpretations grounded in tangible applications which have an independent existence for their own reasons transform the possibilities and give the subject new depth and new energies.

Some Remarks and Questions for P-B PoL

- The attitude that is is just 'grubby engineering' — engineers in overalls infringing on the sacred groves — just won't wash. (Cf. 17th century vis-à-vis the Greeks. McCarthy!).

- Concrete interpretations grounded in tangible applications which have an independent existence for their own reasons transform the possibilities and give the subject new depth and new energies.

- There are passionate methodological debates within this applied field, e.g. 'linear time vs. branching time', which are fertile ground for conceptual and philosophical analysis. Feasibility becomes a major new criterion, and approximate answers must be considered. Such issues are already deeply embedded in physics, but rarely studied philosophically — they should be!

- The attitude that is is just 'grubby engineering' — engineers in overalls infringing on the sacred groves — just won't wash. (Cf. 17th century vis-à-vis the Greeks. McCarthy!).

- Concrete interpretations grounded in tangible applications which have an independent existence for their own reasons transform the possibilities and give the subject new depth and new energies.

- There are passionate methodological debates within this applied field, e.g. 'linear time vs. branching time', which are fertile ground for conceptual and philosophical analysis. Feasibility becomes a major new criterion, and approximate answers must be considered. Such issues are already deeply embedded in physics, but rarely studied philosophically — they should be!

- A deep conceptual issue of logic in CS: the 'next 700 . . . problem'.

The ‘Next 700’ Problem

The ‘Next 700’ Problem

After Peter Landin. ‘The next 700 Programming Languages’ (in 1966!)

The ‘Next 700’ Problem

After Peter Landin. ‘The next 700 Programming Languages’ (in 1966!)

A profusion of possibilities, in e.g.

- programming languages
- type systems
- process calculi
- behavioural equivalences
- logics

The ‘Next 700’ Problem

After Peter Landin. ‘The next 700 Programming Languages’ (in 1966!)

A profusion of possibilities, in e.g.

- programming languages
- type systems
- process calculi
- behavioural equivalences
- logics

Is this profusion a ‘scandal’ of our subject?

Or are the underlying paradigms and templates, the methodological toolkits, sufficient providers of unity?

The ‘Next 700’ Problem

After Peter Landin. ‘The next 700 Programming Languages’ (in 1966!)

A profusion of possibilities, in e.g.

- programming languages
- type systems
- process calculi
- behavioural equivalences
- logics

Is this profusion a ‘scandal’ of our subject?

Or are the underlying paradigms and templates, the methodological toolkits, sufficient providers of unity?

The jury is still out . . .

The ‘Next 700’ Problem

After Peter Landin. ‘The next 700 Programming Languages’ (in 1966!)
A profusion of possibilities, in e.g.

- programming languages
- type systems
- process calculi
- behavioural equivalences
- logics

Is this profusion a ‘scandal’ of our subject?
Or are the underlying paradigms and templates, the methodological toolkits, sufficient providers of unity?
The jury is still out . . .

Cf. André Weil: he compared finding the right definitions in algebraic number theory — which was like carving adamantine rock — to making definitions in the theory of uniform spaces (which he founded), which was like sculpting with snow.

Case Study II: λ-calculus

λ-calculus: a pure calculus of functions.

λ-calculus: a pure calculus of functions.

Variables x, y, z, $\ldots$

Terms

$$t \quad ::= \quad x \mid \underbrace{tu}_{\text{application}} \mid \underbrace{\lambda x.\, t}_{\text{abstraction}}$$

The λ-calculus

λ-calculus: a pure calculus of functions.

Variables x, y, z, ...
Terms

$$t \quad ::= \quad x \mid \underbrace{tu}_{\text{application}} \mid \underbrace{\lambda x.\, t}_{\text{abstraction}}$$

The basic equation governing this calculus is β**-conversion**:

$$(\lambda x.\, t)u = t[u/x]$$

E.g.

$$(\lambda f.\, \lambda x.\, f(fx))(\lambda x.\, x + 1)0 = \cdots 2.$$

λ-calculus: a pure calculus of functions.

Variables x, y, z, ...

Terms

$$t \quad ::= \quad x \mid \underbrace{tu}_{\text{application}} \mid \underbrace{\lambda x.\, t}_{\text{abstraction}}$$

The basic equation governing this calculus is β**-conversion**:

$$(\lambda x.\, t)u = t[u/x]$$

E.g.

$$(\lambda f.\, \lambda x.\, f(fx))(\lambda x.\, x+1)0 = \cdots 2.$$

By orienting this equation, we get a 'dynamics' — β**-reduction**

$$(\lambda x.\, t)u \to t[u/x]$$

Remarks

Remarks

This calculus, encapsulated in one slide, is **incredibly rich**.

Remarks

This calculus, encapsulated in one slide, is **incredibly rich**.

- A universal model of computation — incomparably more wieldy than Turing machines.

Remarks

This calculus, encapsulated in one slide, is **incredibly rich**.

- A universal model of computation — incomparably more wieldy than Turing machines. (Caveats: Church's thesis, resources).

Remarks

This calculus, encapsulated in one slide, is **incredibly rich**.

- A universal model of computation — incomparably more wieldy than Turing machines. (Caveats: Church's thesis, resources).
- Indeed, in sugared form the basis of all contemporary functional programming languages (e.g. ML, Haskell).

Remarks

This calculus, encapsulated in one slide, is **incredibly rich**.

- A universal model of computation — incomparably more wieldy than Turing machines. (Caveats: Church's thesis, resources).
- Indeed, in sugared form the basis of all contemporary functional programming languages (e.g. ML, Haskell).
- Kleene translated the basic results of recursion theory out of lambda calculus into the familiar ϕ_n form.

Remarks

This calculus, encapsulated in one slide, is **incredibly rich**.

- A universal model of computation — incomparably more wieldy than Turing machines. (Caveats: Church's thesis, resources).
- Indeed, in sugared form the basis of all contemporary functional programming languages (e.g. ML, Haskell).
- Kleene translated the basic results of recursion theory out of lambda calculus into the familiar ϕ_n form.
- The untyped calculus allows e.g. terms like $\omega \equiv \lambda x.\, xx$ — self-application.

Remarks

This calculus, encapsulated in one slide, is **incredibly rich**.

- A universal model of computation — incomparably more wieldy than Turing machines. (Caveats: Church's thesis, resources).
- Indeed, in sugared form the basis of all contemporary functional programming languages (e.g. ML, Haskell).
- Kleene translated the basic results of recursion theory out of lambda calculus into the familiar ϕ_n form.
- The untyped calculus allows e.g. terms like $\omega \equiv \lambda x.\, xx$ — self-application.

  Hence also $\Omega \equiv \omega\omega$, which **diverges**:

$$\Omega \rightarrow \Omega \rightarrow \cdots$$

Remarks

This calculus, encapsulated in one slide, is **incredibly rich**.

- A universal model of computation — incomparably more wieldy than Turing machines. (Caveats: Church's thesis, resources).
- Indeed, in sugared form the basis of all contemporary functional programming languages (e.g. ML, Haskell).
- Kleene translated the basic results of recursion theory out of lambda calculus into the familiar ϕ_n form.
- The untyped calculus allows e.g. terms like $\omega \equiv \lambda x.\, xx$ — self-application.

  Hence also $\Omega \equiv \omega\omega$, which **diverges**:

$$\Omega \rightarrow \Omega \rightarrow \cdots$$

  Also, $\mathbf{Y} \equiv \lambda f.\, (\lambda x.\, f(xx))(\lambda x.\, f(xx))$ — recursion.

$$\mathbf{Y}t \rightarrow (\lambda x.\, t(xx))(\lambda x.\, t(xx)) \rightarrow t((\lambda x.\, t(xx))(\lambda x.\, t(xx))) = t(\mathbf{Y}t).$$

Remarks

This calculus, encapsulated in one slide, is **incredibly rich**.

- A universal model of computation — incomparably more wieldy than Turing machines. (Caveats: Church's thesis, resources).
- Indeed, in sugared form the basis of all contemporary functional programming languages (e.g. ML, Haskell).
- Kleene translated the basic results of recursion theory out of lambda calculus into the familiar ϕ_n form.
- The untyped calculus allows e.g. terms like $\omega \equiv \lambda x.\, xx$ — self-application.

  Hence also $\Omega \equiv \omega\omega$, which **diverges**:

  $$\Omega \rightarrow \Omega \rightarrow \cdots$$

  Also, $\mathbf{Y} \equiv \lambda f.\, (\lambda x.\, f(xx))(\lambda x.\, f(xx))$ — recursion.

  $$\mathbf{Y}t \rightarrow (\lambda x.\, t(xx))(\lambda x.\, t(xx)) \rightarrow t((\lambda x.\, t(xx))(\lambda x.\, t(xx))) = t(\mathbf{Y}t).$$

  Historically, Curry extracted $\mathbf{Y}$ from an analysis of **Russell's Paradox**.

Simple Type System for $\times$, $\rightarrow$.

Variable

$$\frac{}{\Gamma, x : t \vdash x : T}$$

Product

$$\frac{\Gamma \vdash t : T \qquad \Gamma \vdash u : U}{\Gamma \vdash \langle t, u \rangle : T \times U} \qquad \frac{\Gamma \vdash v : T \times U}{\Gamma \vdash \pi_1 v : T} \qquad \frac{\Gamma \vdash v : T \times U}{\Gamma \vdash \pi_2 v : U}$$

Function

$$\frac{\Gamma, x : U \vdash t : T}{\Gamma \vdash \lambda x.\, t : U \rightarrow T} \qquad \frac{\Gamma \vdash t : U \rightarrow T \qquad \Gamma \vdash u : U}{\Gamma \vdash tu : T}$$

Natural Deduction System for $\wedge$, $\supset$

Identity

$$\frac{}{\Gamma, A \vdash A}\ \text{Id}$$

Conjunction

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge\text{-intro} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge\text{-elim-1} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge\text{-elim-2}$$

Implication

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\ \supset\text{-intro} \qquad \frac{\Gamma \vdash A \supset B \qquad \Gamma \vdash A}{\Gamma \vdash B}\ \supset\text{-elim}$$

Types and Curry-Howard Correspondence

If we equate

$$\begin{array}{ccc} \wedge & \equiv & \times \\ \supset & \equiv & \longrightarrow \end{array}$$

they are the same!

Developments

- ‘The stone the builders rejected . . . ’

- ‘The stone the builders rejected . . . ’
  The λ-calculus and combinatory logic were a neglected corner of logic studied by a handful of people until Computer Science — initially Strachey and Landin, with a part played by Roger Penrose — put it centre stage in logical methods in CS.

Developments

- 'The stone the builders rejected . . . '
  The λ-calculus and combinatory logic were a neglected corner of logic studied by a handful of people until Computer Science — initially Strachey and Landin, with a part played by Roger Penrose — put it centre stage in logical methods in CS.

- These calculi in turn put the study of **substitution** centre-stage — not such a humble topic! Russell's paradox, cut-elimination, linearity and resources, decidability, complexity, . . .

- 'The stone the builders rejected . . . '
  The λ-calculus and combinatory logic were a neglected corner of logic studied by a handful of people until Computer Science — initially Strachey and Landin, with a part played by Roger Penrose — put it centre stage in logical methods in CS.

- These calculi in turn put the study of **substitution** centre-stage — not such a humble topic! Russell's paradox, cut-elimination, linearity and resources, decidability, complexity, . . .

- Paradoxes: not just biting the bullet — not bugs but features! Recursion, fixpoints, the creative uses of computationally specified infinite objects.

Domain Theory

Domain Theory

Providing extensional models for λ-calculus — spaces satisfying

$$D \cong [D \longrightarrow D]$$

led Dana Scott to **Domain Theory**.

Domain Theory

Providing extensional models for λ-calculus — spaces satisfying

$$D \cong [D \longrightarrow D]$$

led Dana Scott to **Domain Theory**.

Many interesting conceptual aspects of Domain theory:

Domain Theory

Providing extensional models for λ-calculus — spaces satisfying

$$D \cong [D \longrightarrow D]$$

led Dana Scott to **Domain Theory**.

Many interesting conceptual aspects of Domain theory:

- Reconciling paradoxes with fixpoints by introducing additional **partially defined** elements.

Domain Theory

Providing extensional models for λ-calculus — spaces satisfying

$$D \cong [D \longrightarrow D]$$

led Dana Scott to **Domain Theory**.

Many interesting conceptual aspects of Domain theory:

- Reconciling paradoxes with fixpoints by introducing additional **partially defined** elements.
- A general theory of partial information, dynamics of information increase.

Domain Theory

Providing extensional models for λ-calculus — spaces satisfying

$$D \cong [D \longrightarrow D]$$

led Dana Scott to **Domain Theory**.

Many interesting conceptual aspects of Domain theory:

- Reconciling paradoxes with fixpoints by introducing additional **partially defined** elements.
- A general theory of partial information, dynamics of information increase.
- Opens up the (analytical) topology of computation.

Domain Theory

Providing extensional models for λ-calculus — spaces satisfying

$$D \cong [D \longrightarrow D]$$

led Dana Scott to **Domain Theory**.

Many interesting conceptual aspects of Domain theory:

- Reconciling paradoxes with fixpoints by introducing additional **partially defined** elements.
- A general theory of partial information, dynamics of information increase.
- Opens up the (analytical) topology of computation.
- Conceptual ambiguity between ontic and epistemic interpretations.

Domain Theory

Providing extensional models for λ-calculus — spaces satisfying

$$D \cong [D \longrightarrow D]$$

led Dana Scott to **Domain Theory**.

Many interesting conceptual aspects of Domain theory:

- Reconciling paradoxes with fixpoints by introducing additional **partially defined** elements.
- A general theory of partial information, dynamics of information increase.
- Opens up the (analytical) topology of computation.
- Conceptual ambiguity between ontic and epistemic interpretations.

A discussion of domain theory emphasizing conceptual aspects in my article in the Handbook of Philosophy of Information (ed. van Benthem and Adriaans, Elsevier 2008).

Some Remarks and Questions for P-B PoL

- The λ-calculus is essentially canonical for functional computation — no ‘700 problem’ there.

- The λ-calculus is essentially canonical for functional computation — no '700 problem' there.

- What should Church's thesis for concurrency be?

- The λ-calculus is essentially canonical for functional computation — no ‘700 problem’ there.
- What should Church’s thesis for concurrency be?
- The gap between intension and extension: λ-calculus and its models vs. recursion theory. Applications of the recursion theory framework to partial evaluation and mixed computation, program specialization, computational learning theory, computer viruses!

- The λ-calculus is essentially canonical for functional computation — no '700 problem' there.
- What should Church's thesis for concurrency be?
- The gap between intension and extension: λ-calculus and its models vs. recursion theory. Applications of the recursion theory framework to partial evaluation and mixed computation, program specialization, computational learning theory, computer viruses!

  All based on mining the computation content of the $S{-}m{-}n$ theorem and Kleene's Second Recursion Theorem. λ-calculus and its models are too extensional to allow access to this content. Can we find a unified theory?

- The λ-calculus is essentially canonical for functional computation — no '700 problem' there.

- What should Church's thesis for concurrency be?

- The gap between intension and extension: λ-calculus and its models vs. recursion theory. Applications of the recursion theory framework to partial evaluation and mixed computation, program specialization, computational learning theory, computer viruses!

  All based on mining the computation content of the $S{-}m{-}n$ theorem and Kleene's Second Recursion Theorem. λ-calculus and its models are too extensional to allow access to this content. Can we find a unified theory?

- Game Semantics, full abstraction and full completeness. Again, see my article in the Handbook of Philosophy of Information (and, hopefully, forthcoming article in SEP).

Case Study III: Category Theory and Coalgebra

Some Theses About Category Theory

Some Theses About Category Theory

- Category theory (and not just categorical logic) should be seen as part of logic — or vice versa!

Some Theses About Category Theory

- Category theory (and not just categorical logic) should be seen as part of logic — or vice versa!
- Logicians should learn category theory!!

Some Theses About Category Theory

- Category theory (and not just categorical logic) should be seen as part of logic — or vice versa!
- Logicians should learn category theory!!
- Philosophers should learn category theory!!!

Some Theses About Category Theory

- Category theory (and not just categorical logic) should be seen as part of logic — or vice versa!
- Logicians should learn category theory!!
- Philosophers should learn category theory!!!
- Category theory is the language of structure. It enables us to see common patterns far beyond what is otherwise possible.

Some Theses About Category Theory

- Category theory (and not just categorical logic) should be seen as part of logic — or vice versa!
- Logicians should learn category theory!!
- Philosophers should learn category theory!!!
- Category theory is the language of structure. It enables us to see common patterns far beyond what is otherwise possible.

  'Trivial' example: isomorphism.

Some Theses About Category Theory

- Category theory (and not just categorical logic) should be seen as part of logic — or vice versa!
- Logicians should learn category theory!!
- Philosophers should learn category theory!!!
- Category theory is the language of structure. It enables us to see common patterns far beyond what is otherwise possible.

  ‘Trivial’ example: isomorphism.
- Category theory has a strong **normative force**: methodologically, it compels us to ask certain questions — is it **functorial**, **natural**, **universal**? — which point to the key notions in developing a theory.

Some Theses About Category Theory

- Category theory (and not just categorical logic) should be seen as part of logic — or vice versa!

- Logicians should learn category theory!!

- Philosophers should learn category theory!!!

- Category theory is the language of structure. It enables us to see common patterns far beyond what is otherwise possible.

  ‘Trivial’ example: isomorphism.

- Category theory has a strong **normative force**: methodologically, it compels us to ask certain questions — is it **functorial**, **natural**, **universal**? — which point to the key notions in developing a theory.

- Category theory enables us to think bigger thoughts. Many of the most interesting conceptual developments in modern mathematics cannot even be articulated without category theory.

Some Theses About Category Theory

- Category theory (and not just categorical logic) should be seen as part of logic — or vice versa!
- Logicians should learn category theory!!
- Philosophers should learn category theory!!!
- Category theory is the language of structure. It enables us to see common patterns far beyond what is otherwise possible.

  ‘Trivial’ example: isomorphism.
- Category theory has a strong **normative force**: methodologically, it compels us to ask certain questions — is it **functorial**, **natural**, **universal**? — which point to the key notions in developing a theory.
- Category theory enables us to think bigger thoughts. Many of the most interesting conceptual developments in modern mathematics cannot even be articulated without category theory.

  Examples: Cohomology, categorification, the microcosm principle.

Coalgebras

Category theory allows us to **dualize** our entire discussion of algebras to obtain a notion of **coalgebras of an endofunctor**. However, while algebras abstract a familiar set of notions (inductive data types, structural recursion), colagebras open up a new and rather unexpected territory, and provides an effective abstraction and mathematical theory for a central class of computational phenomena:

- Programming over **infinite data structures**: streams, lazy lists, infinite trees ...
- A novel notion of **coinduction**
- Modelling **state-based computations** of all kinds
- The key notion of **bisimulation equivalence** between processes.

Basic Concepts

Let $F : \mathcal{C} \longrightarrow \mathcal{C}$ be a functor.

An F**-coalgebra** is an arrow $\gamma : A \longrightarrow FA$ for some object A of $\mathcal{C}$. We say that A is the **carrier** of the coalgebra, while γ is the **behaviour map**.

An F**-coalgebra homomorphism** from $\gamma : A \longrightarrow FA$ to $\delta : B \longrightarrow FB$ is an arrow $h : A \longrightarrow B$ such that

$$\begin{tikzcd} A \arrow[r, "\gamma"] \arrow[d, "h"'] & FA \arrow[d, "Fh"] \\ B \arrow[r, "\delta"'] & FB \end{tikzcd}$$

F-coalgebras and their homomorphisms form a category $F\!-\!\mathbf{Coalg}$.

An F-coalgebra γ is **final** if for every F-coalgebra δ there is a unique homomorphism from δ to γ.

Proposition 1 *If a final F-coalgebra exists, it is unique up to isomorphism.*

Proposition 2 (Lambek Lemma) *If $\gamma : A \longrightarrow FA$ is final, it is an isomorphism*

Let A be a set of **actions**. A **labelled transition system over** A is a coalgebra for the functor

$$\mathsf{LT}_A : \mathbf{Set} \longrightarrow \mathbf{Set} :: X \mapsto \mathcal{P}_\mathsf{f}(A \times X).$$

Such a coalgebra

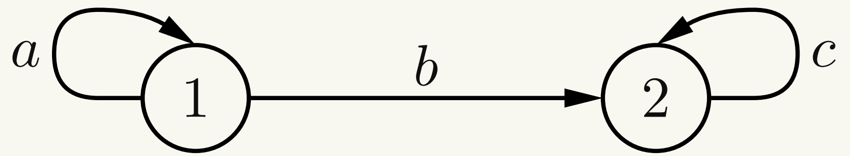
$$\gamma : S \longrightarrow \mathcal{P}_\mathsf{f}(A \times S)$$

can be understood operationally as follows:

- S is a set of **states**
- For each state $s \in S$, $\gamma(s)$ specifies the possible **transitions** from that state. We write $s \xrightarrow{a} s'$ if $(a, s') \in \gamma(s)$. We think of such a transition as consisting of the system performing the action a, and changing state from s to s'. Note that we regard actions as directly **observable**, while states are not.

Note that any labelled transition graph (or "state machine") with labels in A is a coalgebra for LT_A.

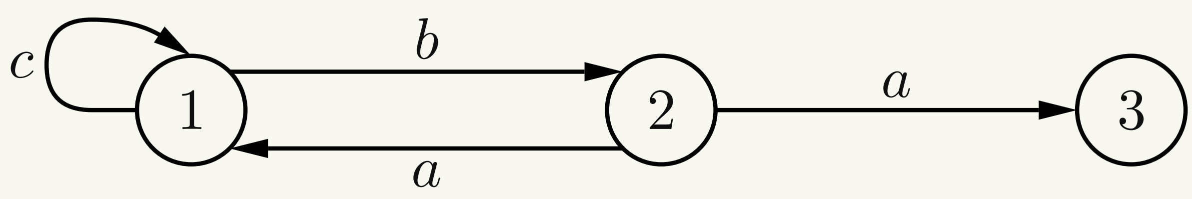
Examples 1.



This corresponds to the coalgebra $(\{1, 2\}, \gamma)$

$$\gamma : 1 \mapsto \{(a, 1), (b, 2)\}, \qquad 2 \mapsto \{(c, 2)\}$$

2.



$$1 \mapsto \{(b, 2), (c, 1)\}, \qquad 2 \mapsto \{(a, 1), (a, 3)\}, \qquad 3 \mapsto \varnothing$$

The key point is this.

Proposition 3 *For any set A of actions, there is a final LT_A-coalgebra $(\mathsf{Proc}_A, \mathsf{out})$.*

The Final Coalgebra

The key point is this.

Proposition 3 *For any set A of actions, there is a final LT_A-coalgebra $(\mathsf{Proc}_A, \mathsf{out})$.*

We think of elements of the final coalgebra as **processes**. The final coalgebra provides a “universal semantics” for transition systems, which is “fully abstract” with respect to observable behaviour.

The key point is this.

Proposition 3 *For any set A of actions, there is a final LT_A-coalgebra $(\mathsf{Proc}_A, \mathsf{out})$.*

We think of elements of the final coalgebra as **processes**. The final coalgebra provides a “universal semantics” for transition systems, which is “fully abstract” with respect to observable behaviour.

All of this generalizes to a large class of endofunctors.

Some Remarks and Questions for P-B PoL

- Coalgebras naturally model state-based systems. They provide a promising basis for reconciling **ontic** and **epistemic** views of states. The final coalgebra is a universal solution — hence unique up to isomorphism — to the problem of constructing states as determined purely by their observational behaviour.

- Coalgebras naturally model state-based systems. They provide a promising basis for reconciling **ontic** and **epistemic** views of states. The final coalgebra is a universal solution — hence unique up to isomorphism — to the problem of constructing states as determined purely by their observational behaviour.

- Coalgebraic logic. A generalized modal logic which can be **read off systematically** from the type functor T. Generalized duality theory.

Some Remarks and Questions for P-B PoL

- Coalgebras naturally model state-based systems. They provide a promising basis for reconciling **ontic** and **epistemic** views of states. The final coalgebra is a universal solution — hence unique up to isomorphism — to the problem of constructing states as determined purely by their observational behaviour.
- Coalgebraic logic. A generalized modal logic which can be **read off systematically** from the type functor T. Generalized duality theory.
- Corecursion, coinduction: mathematically well-founded treatment of non-well-founded objects.

- Coalgebras naturally model state-based systems. They provide a promising basis for reconciling **ontic** and **epistemic** views of states. The final coalgebra is a universal solution — hence unique up to isomorphism — to the problem of constructing states as determined purely by their observational behaviour.
- Coalgebraic logic. A generalized modal logic which can be **read off systematically** from the type functor T. Generalized duality theory.
- Corecursion, coinduction: mathematically well-founded treatment of non-well-founded objects.
  Examples: non-well-founded sets, even non-well-found proofs!

Final Remarks

Logic As A Tool For Building Theories

Computer Science theories of:

- Processes of various kinds, how to mathematically describe and reason about them.
- Information: statics of information representation, dynamics of information flow.

Computer Science theories of:

- Processes of various kinds, how to mathematically describe and reason about them.
- Information: statics of information representation, dynamics of information flow.

The formulation and development of these theories uses a lot of logic — essentially **is** logic, broadly (and properly) construed.

Logic As A Tool For Building Theories

Computer Science theories of:

- Processes of various kinds, how to mathematically describe and reason about them.
- Information: statics of information representation, dynamics of information flow.

The formulation and development of these theories uses a lot of logic — essentially **is** logic, broadly (and properly) construed.

Logic in the mode of open-ended, outward-reaching modelling, rather than conservative codification.

Logic As A Tool For Building Theories

Computer Science theories of:

- Processes of various kinds, how to mathematically describe and reason about them.
- Information: statics of information representation, dynamics of information flow.

The formulation and development of these theories uses a lot of logic — essentially **is** logic, broadly (and properly) construed.

Logic in the mode of open-ended, outward-reaching modelling, rather than conservative codification.

Considerable potential beyond Computer Science: in physics, biology, cognitive and social sciences etc.

Some Challenges for Practice-Based Philosophy of Logic

Some Challenges for Practice-Based Philosophy of Logic

Analyze a **real**, **contemporary** research programme in mathematics, logic or theoretical computer science.

Some Challenges for Practice-Based Philosophy of Logic

Analyze a **real**, **contemporary** research programme in mathematics, logic or theoretical computer science.

Study the choices made, the reasons given, the methodological disagreements, what these were really about, why certain contributions were decisive, why conceptual arguments about approaches were decided in a certain way.

Some Challenges for Practice-Based Philosophy of Logic

Analyze a **real**, **contemporary** research programme in mathematics, logic or theoretical computer science.

Study the choices made, the reasons given, the methodological disagreements, what these were really about, why certain contributions were decisive, why conceptual arguments about approaches were decided in a certain way.

This will engage the interest, enthusiasm, and eventually the active participation of the practitioner community.

Some Challenges for Practice-Based Philosophy of Logic

Analyze a **real**, **contemporary** research programme in mathematics, logic or theoretical computer science.

Study the choices made, the reasons given, the methodological disagreements, what these were really about, why certain contributions were decisive, why conceptual arguments about approaches were decided in a certain way.

This will engage the interest, enthusiasm, and eventually the active participation of the practitioner community.

Contrast: Philosophy of Physics vs. Philosophy of Logic and Mathematics.