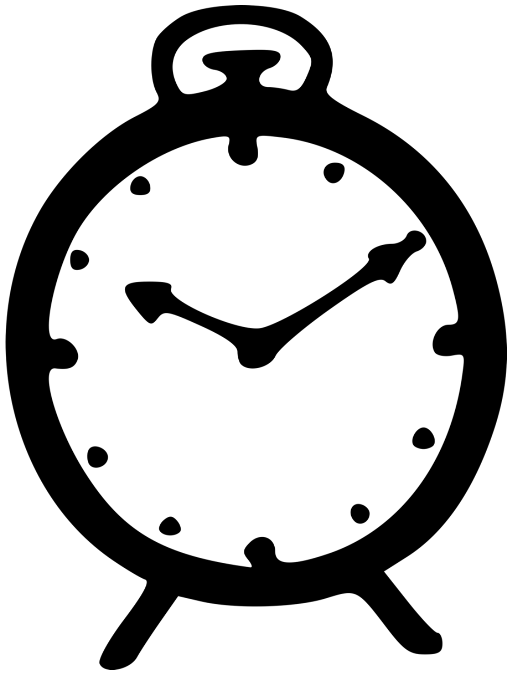
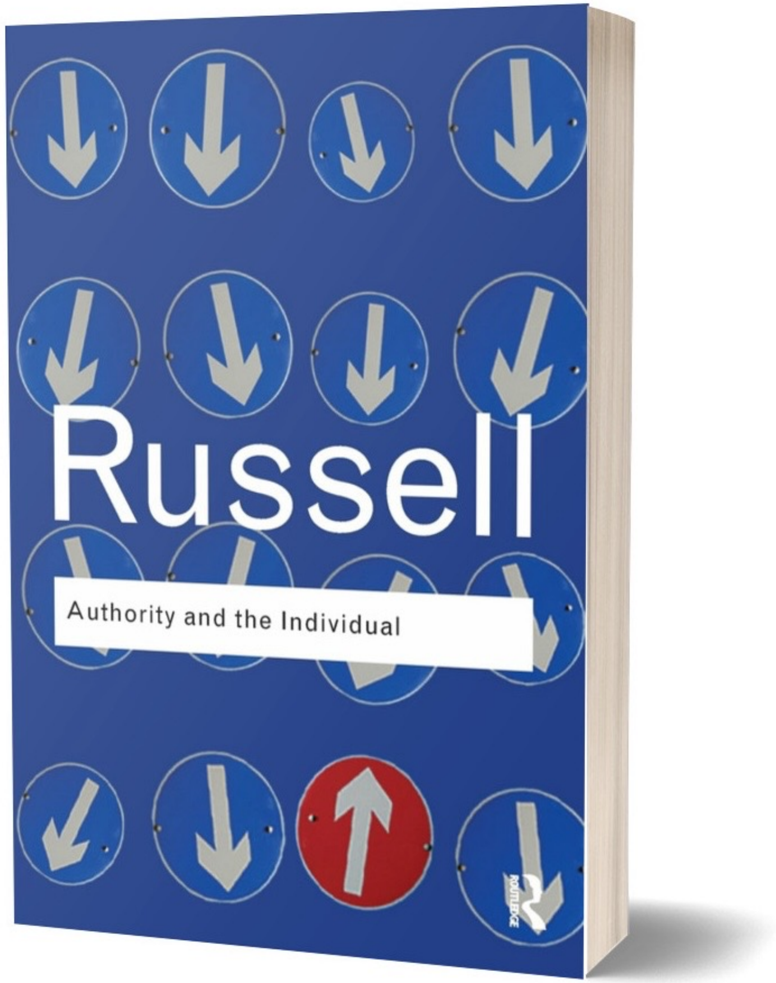
modal (propositions as types)



i. propositions as types

à la Philip Wadler, 2015. *Propositions as Types.*

Bob is an authority


Russell
Authority and the Individual

I hate authority

Do I hate Bob?

Bob is an authority

I hate authority

I hate Bob?!

need a language with unambiguous symbols

propositional logic

$$A, B := \; p, q, r, \ldots \mid A \wedge B \mid A \Rightarrow B \mid \ldots$$

$$\dfrac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \Rightarrow B} \Rightarrow\text{-I} \qquad\qquad \dfrac{\begin{array}{c}\vdots\\ A \Rightarrow B\end{array} \qquad \begin{array}{c}\vdots \\ A\end{array}}{B} \Rightarrow\text{-E}$$

$$\dfrac{\begin{matrix}\vdots & \quad & \vdots \\ A & & B\end{matrix}}{A \wedge B}\ \wedge\text{-I} \qquad \dfrac{\begin{matrix}\vdots \\ A \wedge B\end{matrix}}{A}\ \wedge\text{-E}_1 \qquad \dfrac{\begin{matrix}\vdots \\ A \wedge B\end{matrix}}{B}\ \wedge\text{-E}_2$$

$$
\dfrac{\dfrac{\dfrac{[B \wedge A]^z}{A}\ \wedge\text{-E}_2 \qquad \dfrac{[B \wedge A]^z}{B}\ \wedge\text{-E}_1}{A \wedge B}\ \wedge\text{-I}}{B \wedge A \Rightarrow A \wedge B}\ \Rightarrow\text{-I}^z
$$

$$\dfrac{\dfrac{\dfrac{[B \wedge A]^z}{A}\ \wedge\text{-E}_2 \qquad \dfrac{[B \wedge A]^z}{B}\ \wedge\text{-E}_1}{A \wedge B}\ \wedge\text{-I}}{B}\ \wedge\text{-E}_2$$

$$\frac{[B \wedge A]^z}{B} \wedge\text{-E}_1$$

$$\dfrac{\dfrac{\dfrac{[B \wedge A]^z}{A}\ \wedge\text{-E}_2 \qquad \dfrac{[B \wedge A]^z}{B}\ \wedge\text{-E}_1}{A \wedge B}\ \wedge\text{-I}}{B}\ \wedge\text{-E}_2$$

$$\Big\downarrow\!\!\!\!\sim \ \text{proof}$$

$$\dfrac{[B \wedge A]^z}{B}\ \wedge\text{-E}_1$$

typed-lambda calculus

$$A, B := \ \tau \mid A \times B \mid A \to B \mid \ \ldots$$

$$\frac{\Gamma, z : A \vdash t : B}{\Gamma \vdash \lambda z.t : A \to B}$$

$$\frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}$$

$$\frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \mathrm{fst}\ t : A}$$

$$\frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \mathrm{snd}\ t : B}$$

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash u : B}{\Gamma \vdash (t, u) : A \times B}$$

$$\lambda z.\, (\mathrm{snd}\, z, \mathrm{fst}\, z)$$

$$\dfrac{\dfrac{\dfrac{z : B \times A \vdash z : B \times A}{z : B \times A \vdash \mathrm{snd}\, z : A} \qquad \dfrac{z : B \times A \vdash z : B \times A}{z : B \times A \vdash \mathrm{fst}\, z : B}}{z : B \times A \vdash (\mathrm{snd}\, z, \mathrm{fst}\, z) : A \times B}}{\vdash \lambda z.\, (\mathrm{snd}\, z, \mathrm{fst}\, z) : B \times A \to A \times B}$$

$$\lambda z.\, \mathrm{snd}\, (\mathrm{snd}\, z, \mathrm{fst}\, z)$$

$$\rightsquigarrow \text{prog.}$$

$$\lambda z.\, \mathrm{fst}\, z$$

$$\cfrac{\cfrac{\cfrac{[B \wedge A]^z}{A}\ \wedge\text{-E}_2 \qquad \cfrac{[B \wedge A]^z}{B}\ \wedge\text{-E}_1}{A \wedge B}\ \wedge\text{-I}}{B \wedge A \Rightarrow A \wedge B}\ \Rightarrow\text{-I}^z$$

$$\cfrac{\cfrac{\cfrac{z : B \times A \vdash z : B \times A}{z : B \times A \vdash \mathrm{snd}\, z : A} \qquad \cfrac{z : B \times A \vdash z : B \times A}{z : B \times A \vdash \mathrm{fst}\, z : B}}{z : B \times A \vdash (\mathrm{snd}\, z, \mathrm{fst}\, z) : A \times B}}{\vdash \lambda(z : B \times A).\, (\mathrm{snd}\, z, \mathrm{fst}\, z) : B \times A \to A \times B}$$

$$p, q, r, \ldots \mid A \wedge B \mid A \Rightarrow B \mid \ldots$$

$$\tau \mid A \times B \mid A \to B \mid \ldots$$

propositions *as* types

$$\dfrac{\begin{matrix}[A]^z \\ \vdots \\ B\end{matrix}}{A \Rightarrow B}\ \Rightarrow\text{-I}^z$$

$$\dfrac{\Gamma, z : A \vdash t : B}{\Gamma \vdash \lambda z.t : A \to B}$$

propositions	*as*	types
proofs	*as*	programs

propositions *as* types

proofs *as* programs

$\rightsquigarrow$ proof *as* $\rightsquigarrow$ prog.

is propositions as types useful?

theorem proving for hackers

is propositions as types a coincidence?

ii. modal operators

authority *necessarily* threatens individuality authority *possibly* threatens individuality

$\Box A$: necessarily A

$\Diamond A$: possibly A

$$\Box A \Rightarrow A$$

$$\Box A \Rightarrow \Box\Box A$$

$$A \Rightarrow \Diamond A$$

$$\Diamond\Diamond A \Rightarrow \Diamond A$$

$$\vdots$$

```haskell
class Functor f where
  fmap :: (a -> b) -> f a -> f b
```

```
class Functor m => Monad m where
  return :: a -> m a
  join   :: m (m a) -> m a
```

```haskell
class Functor w => Comonad w where
  extract   :: w a -> a
  duplicate :: w a -> w (w a)
```

$\mathcal{W}A$: comonadic A

$\mathcal{M}A$: monadic A

$$\Box A \Rightarrow A \qquad\qquad \mathcal{W}A \to A$$

$$\Box A \Rightarrow \Box\Box A \qquad\qquad \mathcal{W}A \to \mathcal{W}(\mathcal{W}A)$$

$$A \Rightarrow \Diamond A \qquad\qquad A \to \mathcal{M}A$$

$$\Diamond\Diamond A \Rightarrow \Diamond A \qquad\qquad \mathcal{M}(\mathcal{M}A) \to \mathcal{M}A$$

$$\vdots \qquad\qquad \vdots$$

$\Box A$ as? $\mathcal{W} A$

$\Diamond A$ as? $\mathcal{M} A$

ELSEVIER

Information and Computation

Volume 137, Issue 1, 25 August 1997, Pages 1-33

Regular Article

Propositional Lax Logic ☆

Matt Fairtlough [a], Michael Mendler [b]

Show more

Computational types from a logical perspective

Published online by Cambridge University Press: **01 March 1998**

P. N. BENTON, G. M. BIERMAN and V. C. V. DE PAIVA

Rights & Permissions

A judgmental reconstruction of modal logic

Published online by Cambridge University Press: **25 July 2001**

FRANK PFENNING and ROWAN DAVIES

Article Metrics

Save PDF

A modal analysis of staged computation

Authors: Rowan Davies, Frank Pfenning | Authors Info & Claims

Journal of the ACM (JACM), Volume 48, Issue 3 • Pages 555 - 604 • https://doi.org/10.1145/382780.382785

Published: 01 May 2001 Publication History

Check for updates

"Some claim that each of these variants has an interpretation as a form of computation via Propositions as Types, and a down payment on this claim is given by an interpretation of S4 as staged computation due to Davies and Pfenning [16]"

– Philip Wadler, 2015. *Propositions as Types*

“Benton, Bierman, and de Paiva [4] observed that monads correspond to yet another modal logic, differing from all of S1–S5.”

- Philip Wadler, 2015. *Propositions as Types.*

conundrums

$$(A \to B) \to \mathcal{M}A \to \mathcal{M}B$$

$$\Diamond(A \vee B) \Leftrightarrow \Diamond A \vee \Diamond B$$

$$\vdots$$

iii. modal (propositions as types)

propositions *as* types

constructive proofs *as* programs

proof *as* prog.

proof system for modal logic?

which modal logic?!

more modal logics than proof systems

propositions	*as*	types
??	*as*	programs
??	*as*	↓ prog.

is propositions as types a coincidence?

propositions and types are both meaningless intrinsically

propositions and types are both ^ meaningless

⟦ propositions ⟧	*as*	⟦ types ⟧
??	*as*	⟦ programs ⟧
??	*as*	≈ ⟦ prog.⟧

Bob is an authority

I hate authority

I hate Bob?!

Possible-world semantics

⟦ Bob is an authority ⟧@w1

⟦ I hate authority ⟧@w2

⟦ I hate Bob ⟧@?!

$$(W, \sqsubseteq, V)$$

$$
\begin{aligned}
[\![p]\!]_w &= V(p, w) \\
[\![A \wedge B]\!]_w &= [\![A]\!]_w \times [\![B]\!]_w \\
[\![A \Rightarrow B]\!]_w &= \forall w'.\, w \sqsubseteq w' \to [\![A]\!]_{w'} \to [\![B]\!]_{w'}
\end{aligned}
$$

$$\lambda l.\, \lambda z.\, (\mathrm{snd}\, z, \mathrm{fst}\, z) : [\![B \wedge A \Rightarrow A \wedge B]\!]_w$$

$$(W, \sqsubseteq, R, V)$$

$$\llbracket \Box A \rrbracket_w = \overbrace{\forall v.\, wRv \to}^{\mathcal{W}} \llbracket A \rrbracket_v$$

$$\llbracket \Diamond A \rrbracket_w = \underbrace{\Sigma v.\, wRv \times}_{\mathcal{M}} \llbracket A \rrbracket_v$$

$\Box$ *is* a comonad when R is reflexive and transitive

$\Diamond$ *is* a monad when R is reflexive and transitive

$\Diamond$ *is* strong when R is included in $\sqsubseteq$

Maybe is not a $\Diamond$

$\vdots$

$\Box A$ as? $\mathcal{W}A$

$\Diamond A$ as? $\mathcal{M}A$

$$\Box A \quad \textit{as} \quad \mathcal{W}_{\Box} A$$

$$\Diamond A \quad \textit{as} \quad \mathcal{M}_{\Diamond} A$$

new plan: study semantics of modal logic by embedding modalities in type theory my takeaway: study semantics

⟦ propositions ⟧ *as* ⟦ types ⟧

⟦ propositions ⟧	*as*	⟦ types ⟧
constructive entailment	*as*	⟦ programs ⟧
$\approx$ ent.	*as*	$\approx$ ⟦ prog.⟧

can modal (propositions as types) be useful?

modal logicians study classes of modal logics

PL research can benefit from studying classes of calculi

Compositional Normalisation with Modal Types

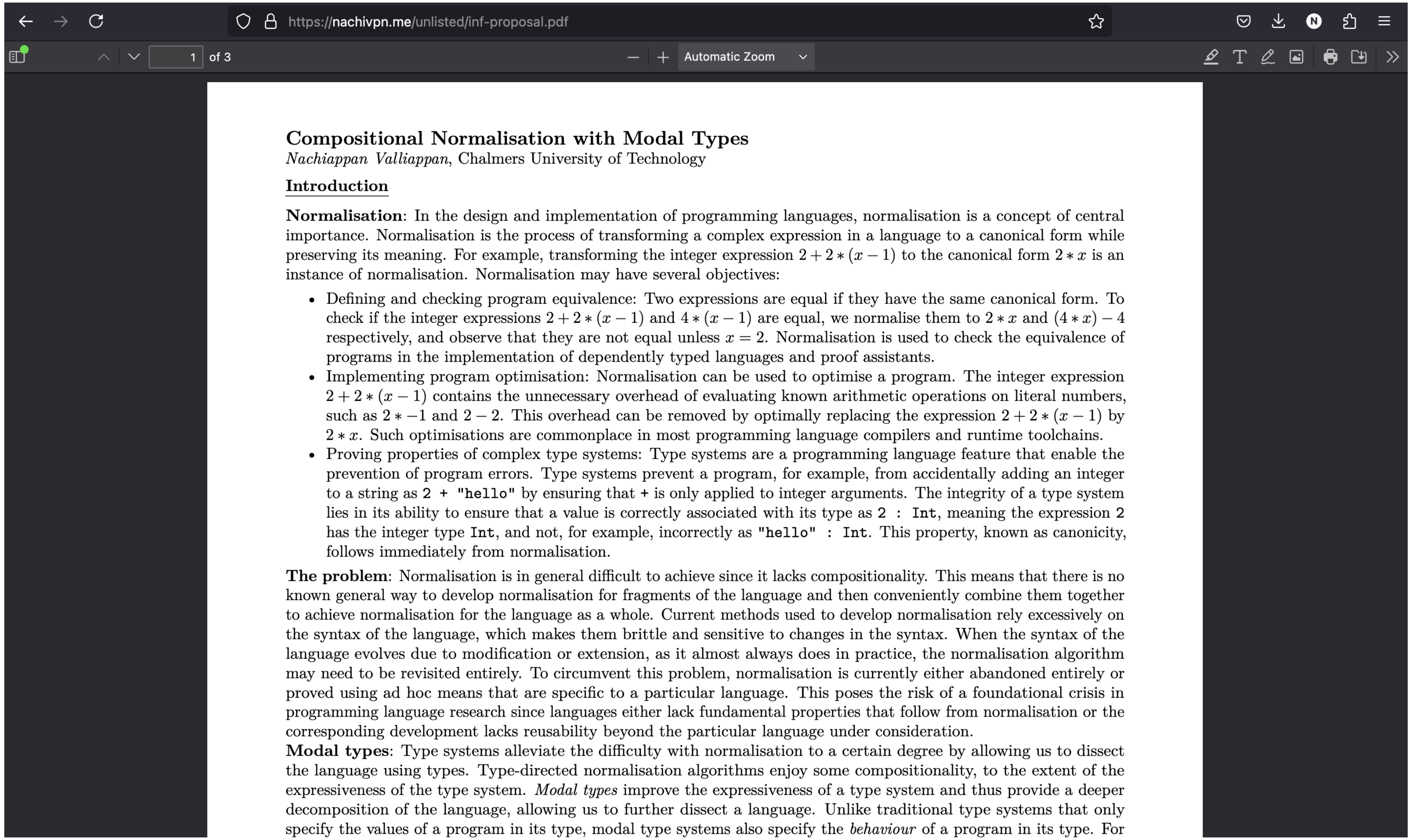
Valliappan, Nachi (Principal Investigator), Lindley, Sam (Sponsor)

School of Informatics, Laboratory for Foundations of Computer Science

Overview

Project Details

Status	Active
Effective start/end date	1/03/24 → 28/02/27



Compositional Normalisation with Modal Types

Nachiappan Valliappan, Chalmers University of Technology

Introduction

Normalisation: In the design and implementation of programming languages, normalisation is a concept of central importance. Normalisation is the process of transforming a complex expression in a language to a canonical form while preserving its meaning. For example, transforming the integer expression $2 + 2 * (x - 1)$ to the canonical form $2 * x$ is an instance of normalisation. Normalisation may have several objectives:

- Defining and checking program equivalence: Two expressions are equal if they have the same canonical form. To check if the integer expressions $2 + 2 * (x - 1)$ and $4 * (x - 1)$ are equal, we normalise them to $2 * x$ and $(4 * x) - 4$ respectively, and observe that they are not equal unless $x = 2$. Normalisation is used to check the equivalence of programs in the implementation of dependently typed languages and proof assistants.
- Implementing program optimisation: Normalisation can be used to optimise a program. The integer expression $2 + 2 * (x - 1)$ contains the unnecessary overhead of evaluating known arithmetic operations on literal numbers, such as $2 * -1$ and $2 - 2$. This overhead can be removed by optimally replacing the expression $2 + 2 * (x - 1)$ by $2 * x$. Such optimisations are commonplace in most programming language compilers and runtime toolchains.
- Proving properties of complex type systems: Type systems are a programming language feature that enable the prevention of program errors. Type systems prevent a program, for example, from accidentally adding an integer to a string as `2 + "hello"` by ensuring that `+` is only applied to integer arguments. The integrity of a type system lies in its ability to ensure that a value is correctly associated with its type as `2 : Int`, meaning the expression `2` has the integer type `Int`, and not, for example, incorrectly as `"hello" : Int`. This property, known as canonicity, follows immediately from normalisation.

The problem: Normalisation is in general difficult to achieve since it lacks compositionality. This means that there is no known general way to develop normalisation for fragments of the language and then conveniently combine them together to achieve normalisation for the language as a whole. Current methods used to develop normalisation rely excessively on the syntax of the language, which makes them brittle and sensitive to changes in the syntax. When the syntax of the language evolves due to modification or extension, as it almost always does in practice, the normalisation algorithm may need to be revisited entirely. To circumvent this problem, normalisation is currently either abandoned entirely or proved using ad hoc means that are specific to a particular language. This poses the risk of a foundational crisis in programming language research since languages either lack fundamental properties that follow from normalisation or the corresponding development lacks reusability beyond the particular language under consideration.

Modal types: Type systems alleviate the difficulty with normalisation to a certain degree by allowing us to dissect the language using types. Type-directed normalisation algorithms enjoy some compositionality, to the extent of the expressiveness of the type system. *Modal types* improve the expressiveness of a type system and thus provide a deeper decomposition of the language, allowing us to further dissect a language. Unlike traditional type systems that only specify the values of a program in its type, modal type systems also specify the *behaviour* of a program in its type. For

nachivpn.me

github.com/nachivpn

iv. shoutout

TypeSig ❤️ you!


ADVENT
of
PROOF
2024
An Advent of Code style
proof competition
running from Dec 12st
to Dec 25th.

organic collective embodiment of propositions as types!

a proof theorist’s dream

not as an obligation
but for the pleasure of
constructing one

I ❤️ TypeSig!

Prop

I ❤️ ~~Type~~Sig!

/nachi