modal (propositions as types)

Nachi Valliappan

University of Edinburgh

Edinburgh Bayes Coffee House Tech Talk, Huawei-Edinburgh Joint Lab, 27 November ‘25

a bird's eye view of a bridge

propositions as types

$$\varphi, \psi := \; p, q, \ldots \; \mid \varphi \wedge \psi \mid \varphi \Rightarrow \psi \mid \; \ldots$$

$$A, B := \; \mathrm{int}, \mathrm{bool}, \ldots \; \mid A \times B \mid A \to B \mid \; \ldots$$

“propositions and types are isomorphic”

type systems for programmers

Agda

proof assistants for CS researchers

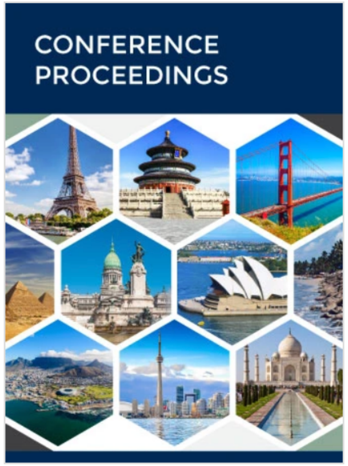
LEAN THEOREM PROVER

ROCQ

proof assistants for logicians

LEAN THEOREM PROVER

ROCQ



Previous | Next

Table of Contents
Similar Articles

2025 40th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)

Semantical Analysis of Intuitionistic Modal Logics between CK and IK

Year: 2025, Pages: 169-182
DOI Bookmark: 10.1109/LICS65433.2025.00020

Authors

Jim De Groot, University of Bern,Mathematical Institute,Bern,Switzerland
Ian Shillito, University of Birimingham,School of Computer Science,Birmingham,UK
Ranald Clouston, Australian National University,School of Computing,Canberra,Australia

DOWNLOAD PDF | SHARE ARTICLE | GENERATE CITATION

We formalise all our results in the Rocq Prover [68], which not only adds confidence to our results (in particular, the doubt raised [50] about the relational semantics for WK may now be considered settled), but is a crucial working tool for managing the profusion of logics which arise as one considers new axioms. As a proof of concept of this methodology of working from a base relational semantics for CK with support from Rocq, we go on to provide relational semantics and conservativity results for Kojima's logic, and for the weakening of $\mathsf{I}_{\Diamond\Box}$ used in FIK. Each mechanised result in the paper is accompanied by a clickable rooster symbol "" leading to its mechanisation. The full mechanisation can be found at https://github.com/ianshil/CK and its documentation at https://ianshil.github.io/CK/toc.html.

We formalise all our results in the Rocq Prover [68], which not only adds confidence to our results (in particular, the doubt raised [50] about the relational semantics for WK may now be considered settled), but is a crucial working tool for managing the profusion of logics which arise as one considers new axioms. As a proof of concept of this methodology of working from a base relational semantics for CK with support from Rocq, we go on to provide relational semantics and conservativity results for Kojima's logic, and for the weakening of $\mathsf{I}_{\Diamond\Box}$ used in FIK. Each mechanised result in the paper is accompanied by a clickable rooster symbol “” leading to its mechanisation. The full mechanisation can be found at https://github.com/ianshil/CK and its documentation at https://ianshil.github.io/CK/toc.html.

Formalization papers using Lean

- Riccardo Brasca, Christopher Birkbeck, Eric Boidi, Alex Best, Ruben De Velde, Andrew Yang, A complete formalization of Fermat's Last Theorem for regular primes in Lean. Annals of Formalized Mathematics, 2025 lean4
- David Loeffler, Michael Stoll, Formalizing zeta and L-functions in Lean. Annals of Formalized Mathematics, 2025 lean4
- Salvatore Mercuri, Formalising the local compactness of the adele ring. Annals of Formalized Mathematics, 2025 lean4
- Joël Riou, Formalization of derived categories in Lean/mathlib. Annals of Formalized Mathematics, 2025 lean4
- Dagur Asgeirsson, Riccardo Brasca, Nikolas Kuhn, Filippo Nuccio Mortarino Majno di Capriglio, Adam Topaz, Categorical foundations of formalized condensed mathematics. The Journal of Symbolic Logic, 2024 lean4
- Dagur Asgeirsson, Towards Solid Abelian Groups: A Formal Proof of Nöbeling's Theorem. 15th International Conference on Interactive Theorem Proving (ITP 2024), 2024 lean4
- Henning Basold, Peter Bruin, Dominique Lawson, The Directed Van Kampen Theorem in Lean. 15th International Conference on Interactive Theorem Proving (ITP 2024), 2024 lean4
- Siddharth Bhat, Alex Keizer, Chris Hughes, Andrés Goens, Tobias Grosser, Verifying Peephole Rewriting in SSA Compiler IRs. 15th International Conference on Interactive Theorem Proving (ITP 2024), 2024 lean4
- Joshua Clune, Yicheng Qian, Alexander Bentkamp, Jeremy Avigad, Duper: A Proof-Producing Superposition Theorem Prover for Dependent Type Theory. 15th International Conference on Interactive Theorem Proving (ITP 2024), 2024 lean4
- María Frutos-Fernández, Filippo Nuccio Mortarino Majno di Capriglio, A Formalization of Complete Discrete Valuation Rings and Local Fields. Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '24), 2024 lean3
- Sam Ezeh, Graphical Rewriting for Diagrammatic Reasoning in Monoidal Categories in Lean4. 15th International Conference on Interactive Theorem Proving (ITP 2024), 2024 lean4
- Patrick Massot, Teaching Mathematics Using Lean and Controlled Natural Language. 15th International Conference on Interactive Theorem Proving (ITP 2024), 2024 lean4
- Kai Obendrauf, Anne Baanen, Patrick Koopmann, Vera Stebletsova, Lean Formalization of Completeness Proof for Coalition Logic with Common Knowledge. 15th International Conference on Interactive Theorem Proving (ITP 2024), 2024 lean4
- Bernardo Subercaseaux, Wojciech Nawrocki, James Gallicchio, Cayden Codel, Mario Carneiro, Marijn Heule, Formal Verification of the Empty Hexagon Number. 15th International Conference on Interactive Theorem Proving (ITP 2024), 2024 lean4
- Floris Doorn, Heather Macbeth, Integrals Within Integrals: A Formalization of the Gagliardo-Nirenberg-Sobolev Inequality. 15th International Conference on Interactive Theorem Proving (ITP 2024), 2024 lean4

“modal” types

Oxidizing OCaml with Modal Memory Management

ANTON LORENZEN, University of Edinburgh, United Kingdom
LEO WHITE, Jane Street, United Kingdom
STEPHEN DOLAN, Jane Street, United Kingdom
RICHARD A. EISENBERG, Jane Street, USA
SAM LINDLEY,

Programmers can
by allocating on th
optimizations can
a design based on

2024

Multi-stage Programming with Splice Variables

TSUNG-JU CHIANG, University of Toronto, Canada
NINGNING XIE, University of Toronto, Canada

ion overhead
as introduced
nd also often
. This paper
binds splice
omputations.
xt to capture
es to features

2025

Structural Information Flow: A Fresh Look at Types for Non-interference

HEMANT GOUNI, Carnegie Mellon University, USA
FRANK PFENNING, Carnegie Mellon University, USA
JONATHAN ALDRICH, Carnegie Mellon University, USA

Information flow
For instance, it can
that the former do
and their maturity
information flow
for simple progra
information leaka
their purpose. Use
non-interference
We introduce *st*

2025

Modal Effect Types

WENHAO TANG, The University of Edinburgh, United Kingdom
LEO WHITE, Jane Street, United Kingdom
STEPHEN DOLAN, Jane Street, United Kingdom
DANIEL HILLERSTRÖM, The University of Edinburgh, United Kingdom
SAM LINDLEY, The University of Edinburgh, United Kingdom
ANTON LORENZEN, The University of Edinburgh, United Kingdom

2025

What are modal types and where does one start studying modal type checking?

Ask Question

Asked 4 months ago Modified 4 months ago Viewed 339 times

6

It looks like "modal" type systems are getting somewhat more popular, with OxCaml, and some of the recent publications like Modal Effect Types.

I want to understand modal type checking, but I have no idea where to start. "Modal type checking" (and "type theory", "type systems") don't even have a Wikipedia page. Search engines either return unrelated "modal" things, or fairly advanced dependent types papers.

I'm wondering where does a practitioner who understand basics of type checking and inference (in OOP and function languages) start studying modal type checking?

Note: I'm aware of the other question on this site about modal types, but it doesn't answer my question.

$$F :: \mathrm{Type} \to \mathrm{Type}$$

+ carefully selected operations

$$\varphi, \psi := \; p, q, \ldots \; | \; \varphi \wedge \psi \; | \; \varphi \Rightarrow \psi \; | \; \ldots$$

$$\ldots | \; \Box\varphi \; | \; \Diamond\varphi \; | \; \nabla\varphi \ldots$$

$$A, B := \; \mathrm{int}, \mathrm{bool}, \ldots \; | \; A \times B \; | \; A \to B \; | \; \ldots$$

$$\ldots | \; \mathrm{W}A \; | \; \mathrm{M}A \; | \; \mathrm{F}A \ldots$$

1. propositions as types

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

1.1. propositional logic

For, while we must begin with what is evident, things are evident in two ways—some to us, some without qualification. Presumably, then, *we* must begin with things evident to *us*.

— Aristotle, *The Nicomachean Ethics (translated)*

Dangling assumptions
and dubious axioms
fly past victims
shackled in awe
by the coherence of
an Aristotelian discourse

$$\varphi, \psi := \; p, q, \ldots \;\mid \varphi \wedge \psi \mid \varphi \Rightarrow \psi \mid \; \ldots$$

$$\dfrac{\begin{matrix}[\varphi]\\ \vdots \\ \psi\end{matrix}}{\varphi \Rightarrow \psi}\ \Rightarrow\text{-I} \qquad\qquad \dfrac{\begin{matrix}\vdots & \qquad & \vdots\\ \varphi \Rightarrow \psi & \qquad & \varphi\end{matrix}}{\psi}\ \Rightarrow\text{-E}$$

$$\dfrac{\begin{matrix}\vdots & & \vdots \\ \varphi & & \psi\end{matrix}}{\varphi \wedge \psi}\ \wedge\text{-I} \qquad \dfrac{\begin{matrix}\vdots \\ \varphi \wedge \psi\end{matrix}}{\varphi}\ \times\text{-E}_1 \qquad \dfrac{\begin{matrix}\vdots \\ \varphi \wedge \psi\end{matrix}}{\psi}\ \times\text{-E}_2$$

$$\dfrac{\dfrac{\dfrac{[\psi \wedge \varphi]^z}{\varphi}\,\wedge\text{-E}_2 \qquad \dfrac{[\psi \wedge \varphi]^z}{\psi}\,\wedge\text{-E}_1}{\varphi \wedge \psi}\,\wedge\text{-I}}{\psi \wedge \varphi \Rightarrow \varphi \wedge \psi}\,\Rightarrow\text{-I}^z$$

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

$$\frac{[\psi \wedge \varphi]^z}{\psi} \ \wedge\text{-E}_1$$

$$\cfrac{\cfrac{\cfrac{[\psi \wedge \varphi]^z}{\varphi}\ \wedge\text{-E}_2 \qquad \cfrac{[\psi \wedge \varphi]^z}{\psi}\ \wedge\text{-E}_1}{\varphi \wedge \psi}\ \wedge\text{-I}}{\psi}\ \wedge\text{-E}_2$$

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

$$\cfrac{[\psi \wedge \varphi]^z}{\psi}\ \wedge\text{-E}_1$$

1.2. typed-lambda calculus

$$A, B := \ \mathrm{int}, \mathrm{bool}, \ldots \ \mid A \times B \mid A \rightarrow B \mid \ \ldots$$

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

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

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

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

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

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

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

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

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

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

$$
\dfrac{\dfrac{\dfrac{[\psi \wedge \varphi]^z}{\varphi}\,\wedge\text{-E}_2 \qquad \dfrac{[\psi \wedge \varphi]^z}{\psi}\,\wedge\text{-E}_1}{\varphi \wedge \psi}\,\wedge\text{-I}}{\psi \wedge \varphi \Rightarrow \varphi \wedge \psi}\,\Rightarrow\text{-I}^z
$$

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

$$\varphi, \psi := \ p, q, \ldots \ \mid \varphi \wedge \psi \mid \varphi \Rightarrow \psi \mid \ \ldots$$

$$A, B := \ \mathrm{int}, \mathrm{bool}, \ldots \ \mid A \times B \mid A \to B \mid \ \ldots$$

propositions *as* types

$$\dfrac{\begin{array}{c}[\varphi]^z \\ \vdots \\ \psi\end{array}}{\varphi \Rightarrow \psi}\ \Rightarrow\text{-I}^z$$

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

propositions	*as*	types
proofs	*as*	programs

propositions	*as*	types
proofs	*as*	programs
$\big\downarrow$ proof	*as*	$\big\downarrow$ prog.

2. modal operators?

"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]"

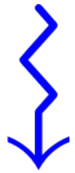
— Philip Wadler, 2015. *Propositions as Types*

“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.*

propositions	*as*	types
intuitionistic proofs	*as*	programs
⇝ proof	*as*	⇝ prog.

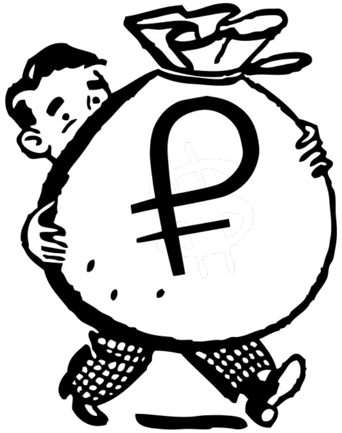
intuitionistic modal logic?

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

is propositions as types a coincidence?

For a moment in despair, it would seem,
“Propositions as Types” is a dead school
cloaking the timidity of a delusional fool

why not give up?



modal propositions

modal types

1912

Valeria de Paiva’s blog

Logic ForAll: Intuitionistic Modal | New Tab

logic-forall.blogspot.com/2025/06/intuitionistic-modal-logic-are-you-sure.html?m=1

Logic ForAll

Forro' Logico: A blog to keep things.

Tuesday, June 17, 2025

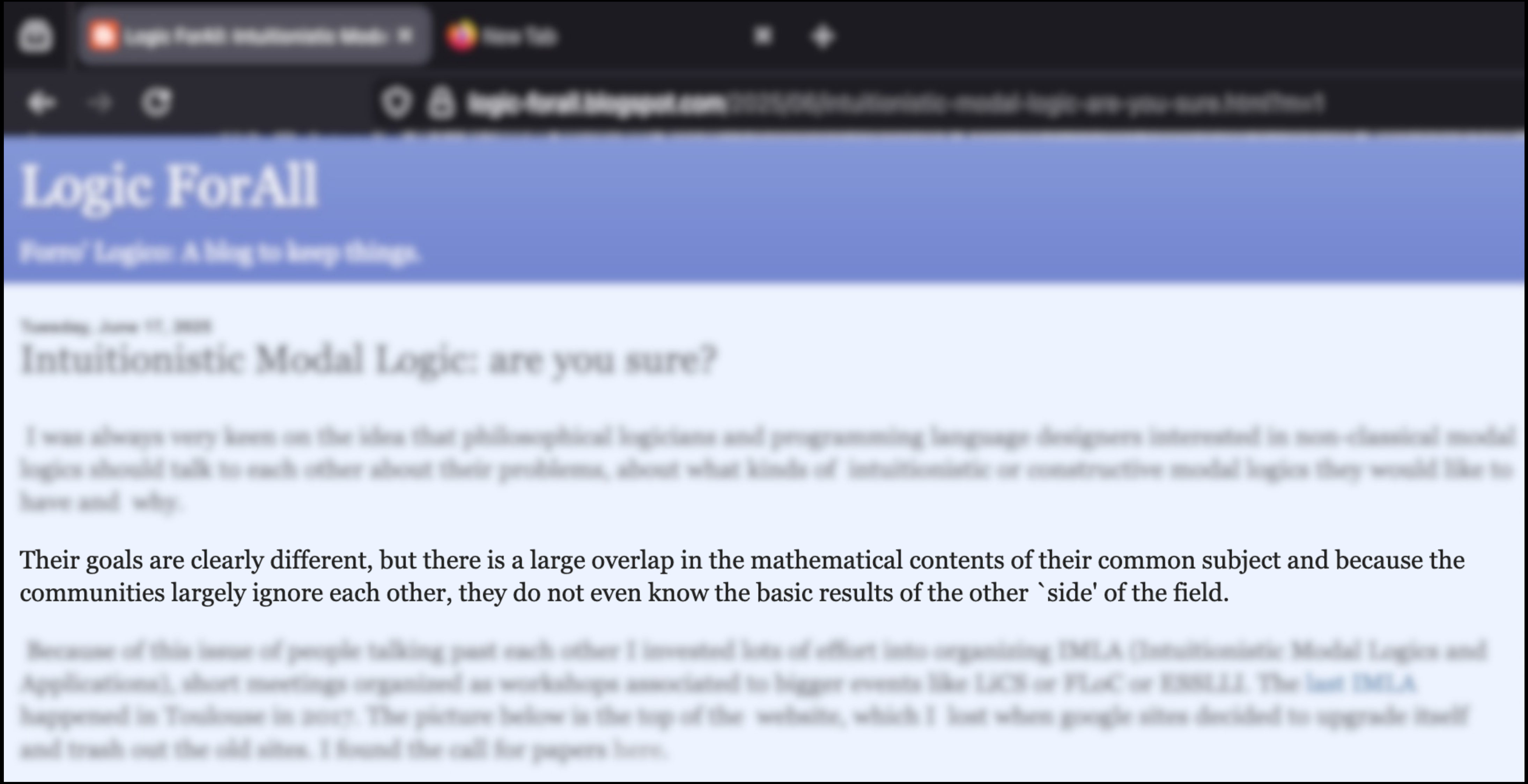
Intuitionistic Modal Logic: are you sure?

I was always very keen on the idea that philosophical logicians and programming language designers interested in non-classical modal logics should talk to each other about their problems, about what kinds of intuitionistic or constructive modal logics they would like to have and why.

Their goals are clearly different, but there is a large overlap in the mathematical contents of their common subject and because the communities largely ignore each other, they do not even know the basic results of the other `side' of the field.

Because of this issue of people talking past each other I invested lots of effort into organizing IMLA (Intuitionistic Modal Logics and Applications), short meetings organized as workshops associated to bigger events like LiCS or FLoC or ESSLLI. The last IMLA happened in Toulouse in 2017. The picture below is the top of the website, which I lost when google sites decided to upgrade itself and trash out the old sites. I found the call for papers here.

Valeria de Paiva’s blog



Logic ForAll

Intuitionistic Modal Logic: are you sure?

Their goals are clearly different, but there is a large overlap in the mathematical contents of their common subject and because the communities largely ignore each other, they do not even know the basic results of the other `side' of the field.

3. intuitionistic modal logic

$\Box\varphi$: "necessarily phi"

$\Diamond\varphi$: "possibly phi"

$\blacklozenge\varphi$: "previously phi"

$$\vdots$$

3.1. intuitionistic boxes

$\Box\varphi$: "necessarily phi"

Fitch-Style Modal Lambda Calculi

Ranald Clouston[✉]

Department of Compute

ra

2017–18

Modal Dependent Type Theory and Dependent Right Adjoints

Lars Birkedal[1], Ranald Clouston[2], Bassel Mannaa[3], Rasmus Ejlers

us University, Aarhus,

2018–19

Simply RaTT: A Fitch-Style Modal Calculus for Reactive Programming without Space Leaks

PATRICK BAHR, IT University
CHRISTIAN ULDAL GRAU
RASMUS EJLERS MØGELB

Functiona
user to d
must ens
leaks. To
Recent
in the sy
RaTT, a
approac
previous
executes

Implementing a Modal Dependent Type Theory

DANIEL GRATZER, Aarhus University, Denmark
JONATHAN STERLING, Carnegie Mellon University, United States

still significant
nt a dependent
as an **S4**-style
s and provides
We design and
ovel extension
of assistant for

Proof theory

107

2019

Multimodal Dependent Type Theory

Daniel Gratzer
Aarhus University
gratzer@cs.au.dk

G. A. Kavvos
Aarhus University
alex.kavvos@cs.au.dk

Andreas Nuyts
imec-DistriNet, KU Leuven

Lars Birkedal
Aarhus University

2020

Fitch-Style Modal Lambda Calculi

Ranald Clouston(✉)

Department of Computer Science, Aarhus University, Aarhus, Denmark
ranald.clouston@cs.au.dk

2017–18

Abstract. Fitch-style modal deduction, in which modalities are eliminated by opening a subordinate proof, and introduced by shutting one, were investigated in the 1990s as a basis for lambda calculi. We show that such calculi have good computational properties for a variety of intuitionistic modal logics. Semantics are given in cartesian closed categories equipped with an adjunction of endofunctors, with the necessity modality interpreted by the right adjoint. Where this functor is an idempotent comonad, a coherence result on the semantics allows us to present a calculus for intuitionistic S4 that is simpler than others in the literature. We show the calculi can be extended à la tense logic with the left

Coming to Terms with Modal Logic:
On the interpretation of modalities in typed λ-calculus

Tijn Borghuis

Coming to Terms with Modal Logic:
On the Interpretation of Modalities in Typed λ-Calculus

Proefschrift

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof.dr. J.H. van Lint, voor een commissie aangewezen door het College van Dekanen in het openbaar te verdedigen op vrijdag 9 december 1994 om 16.00 uur

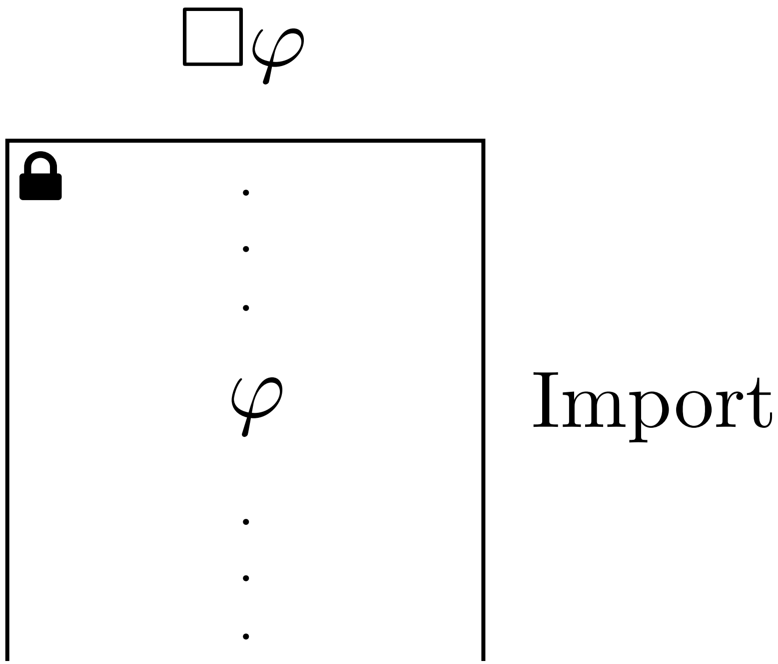
door
Valentijn Anton Johan Borghuis
geboren te Oldenzaal

1994

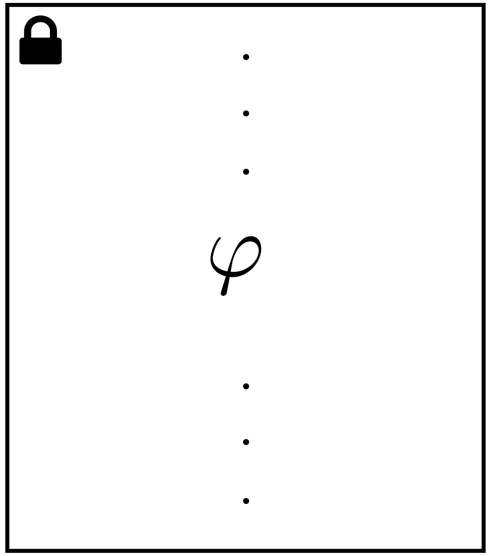
$$\text{If } \vdash \varphi \text{ then } \Gamma \vdash \Box\varphi$$

IK, IK4, IT, IS4

$$\begin{cases} \Box(\varphi \Rightarrow \psi) \Rightarrow \Box\varphi \Rightarrow \Box\psi \\ \Box\varphi \Rightarrow \varphi \\ \Box\varphi \Rightarrow \Box\Box\varphi \end{cases}$$

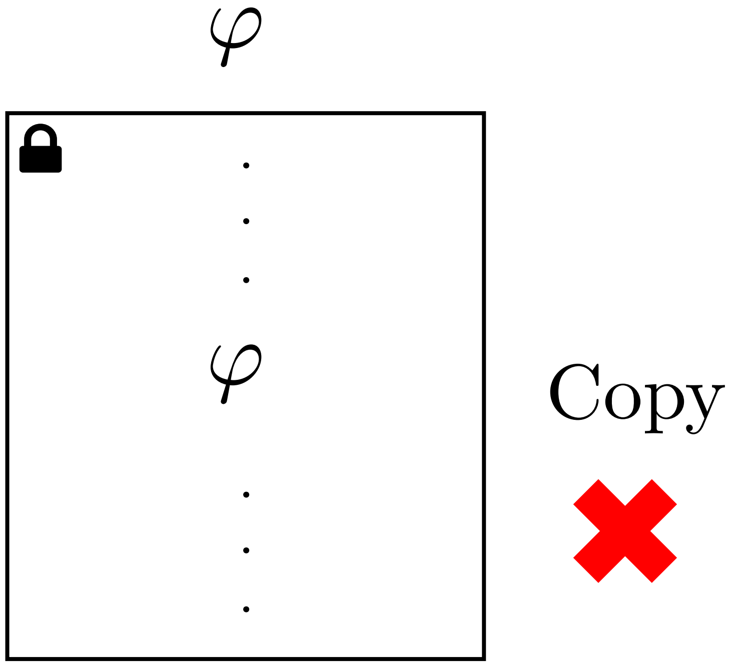


$$\frac{\Gamma \vdash t : \Box A}{\Gamma, \text{🔒} \vdash \textbf{unbox}\ t : A}$$



$\Box\varphi$ Export

$$\frac{\Gamma, \text{🔒} \vdash t : A}{\Gamma \vdash \mathbf{box}\ t : \Box A}$$



$$\dfrac{}{\Gamma, x : A, \Gamma' \vdash x : A}\ \text{🔒} \notin \Gamma'$$

box propositions	*as*	box types
proofs in IK/...	*as*	programs in FS-IK/...
??	*as*	$\rightsquigarrow$ prog.

modal logicians use relational semantics

modal type theorists use categorical semantics easy to construct

aid formulating program reduction

relational models vs categorical models

enjoy a wealth of completeness results

more general

can we steal relational semantics?

Check for updates

Normalization for Fitch-Style Modal Calculi

NACHIAPPAN VALLIAPPAN, Chalmers University of Technology, Sweden
FABIAN RUCH, Unaffiliated, Sweden
CARLOS TOMÉ CORTIÑAS, Chalmers University of Technology, Sweden

2022

Two-Dimensional Kripke Semantics I: Presheaves

G. A. Kavvos
University of Bristol, United Kingdom

The st
seman
secon
two c

2012

2024

Two-dimensional Kripke Semantics II: Stability and Completeness*

G. A. Kavvos[a,1]

Takeaway: "propositions as types" extends to boxes in intuitionistic modal logic

$(\mathcal{W}, R_i, R_m)$

$\Box$, 🔒

propositions as types

3.2. intuitionistic diamonds

$\Diamond\varphi$: “possibly phi”

On Intuitionistic Diamonds (and Lack Thereof)

Anupam Das and Sonia Marin(✉)

University of Birmingham, Birmingham, UK
{a.das,s.marin}@bham.ac.uk

2022

Semantical Analysis of Intuitionistic Modal Logics between CK and IK

Jim de Groot
Mathematical Institute
University of Bern
Bern, Switzerland
https://orcid.org/0000-0003-1375-6758

Ian Shillito
School of Computer Science
University of Birimingham
Birmingham, UK
https://orcid.org/0009-0009-1529-2679

Ranald Clouston
School of Computing
Australian National University
Canberra, Australia
ranald.clouston@anu.edu.au

2025

intuitionistic diamonds are under-developed it makes little sense to ask for “diamond” types it makes little sense to ask for “diamond” types

*currently**

but we may ask for “monadic” modalities

$\mathrm{M}A$: “computation of A”

INFORMATION AND COMPUTATION **93**, 55–92 (1991)

Notions of Computation and Monads

EUGENIO MOGGI*

Department of Computer Science, University of Edinburgh, Edinburgh EH9 3JZ, UK

The λ-calculus is considered a useful mathematical tool in the study of programming languages, since programs can be *identified* with λ-terms. However, if one goes further and uses $\beta\eta$-conversion to prove equivalence of programs, then a gross simplification is introduced (programs are identified with total functions from *values* to *values*) that may jeopardise the applicability of theoretical results. In this paper we introduce calculi, based on a categorical semantics for *computations*, that provide a correct basis for proving equivalence of programs for a wide range of *notions of computation*.

INTRODUCTION

This paper is about logics for reasoning about programs, in particular for proving equivalence of programs. Following a consolidated tradition in

1991

1997–98

INFORMATION AND COMPUTATION **137**, 1–33 (1997)
ARTICLE NO. IC972627

Propositional Lax Logic

Matt Fairtlough

Department of Computer Science, University of Sheffield, Regent Court, Sheffield S1 4DP, United Kingdom
E-mail: m.fairtlough@dcs.shef.ac.uk

and

Michael Mendler

Department of Computer Science, University of Passau, Innstrasse 33, D-94032 Passau, Germany
E-mail: mendler@fmi.uni-passau.de

We investigate a peculiar intuitionistic modal logic, called Propo

J. Functional Programming **8** (2): 177–193, March 1998. Printed in the United Kingdom 177

Computational types from a logical perspective

P. N. BENTON
Persimmon IT Inc., Cambridge, UK

G. M. BIERMAN
Gonville and Caius College, Cambridge, UK

V. C. V. DE PAIVA
School of Computer Science, University of Birmingham, Birmingham, UK

Abstract

Moggi's computational lambda calculus is a metalanguage for denotational semantics which arose from the observation that many different notions of computation have the categorical structure of a strong monad on a cartesian closed category. In this paper we show that the computational lambda calculus also arises naturally as the term calculus corresponding (by the Curry–Howard correspondence) to a novel intuitionistic modal propositional logic. We give natural deduction, sequent calculus and Hilbert-style presentations of this logic and prove strong normalisation and confluence results.

Capsule Review

This is a short, concise and well-written paper that addresses Moggi's Computational Lambda Calculus (CLC) from an interesting but little explored perspective.

The authors take the CLC, extend it with coproducts and apply the Curry–Howard correspondence to obtain a propositional intuitionistic modal logic which they call CL-logic.

If $\vdash \varphi \Rightarrow \psi$ then $\Gamma \vdash \Diamond\varphi \Rightarrow \Diamond\psi$

$$
\left.
\begin{array}{l}
(\varphi \Rightarrow \psi) \Rightarrow \Diamond\varphi \Rightarrow \Diamond\psi \\
\varphi \Rightarrow \Diamond\varphi \\
\Diamond\Diamond\varphi \Rightarrow \Diamond\varphi
\end{array}
\right\} \text{PLL}
$$

lax propositions	*as*	monadic types
proofs in **PLL**	*as*	programs in **ML**
??	*as*	$\rightsquigarrow$ prog.

$$\text{If} \vdash \varphi \Rightarrow \psi \text{ then } \Gamma \vdash \Diamond\varphi \Rightarrow \Diamond\psi$$

$$\left.\begin{array}{l} (\varphi \Rightarrow \psi) \Rightarrow \Diamond\varphi \Rightarrow \Diamond\psi \\ \varphi \Rightarrow \Diamond\varphi \\ \Diamond\Diamond\varphi \Rightarrow \Diamond\varphi \end{array}\right\} \text{SL, SRL, SJL, PLL}$$

Lax Modal Lambda Calculi

Nachiappan Valliappan
University of Edinburgh, United Kingdom

Abstract

Intuitionistic modal logic (IML) is the study of extending intuitionistic propositional logic with the box and diamond modalities. Advances in IML have led to a plethora of useful applications in programming languages via the development of corresponding type theories with modalities. Until recently, IMLs with diamonds have been misunderstood as somewhat peculiar and unstable, causing the development of type theories with diamonds to lag behind type theories with boxes. In this article, we develop a family of typed-lambda calculi corresponding to sublogics of a peculiar IML with diamonds known as Lax logic. These calculi provide a modal logical foundation for various strong functors in typed-functional programming. We present possible-world and categorical semantics for these calculi and constructively prove normalization, equational completeness and proof-theoretic inadmissibility results. Our key results have been formalized using the proof assistant Agda.

2012 ACM Subject Classification Replace ccsdesc macro with valid one

2026

Takeaway: “propositions as types” extends to boxes and a special class of diamonds known as lax modalities

4. onwards

[Theory] Ongoing: Interaction with sum types

$$\Diamond(\varphi \vee \psi) \Leftrightarrow \Diamond\varphi \vee \Diamond\psi$$

[Application] Ongoing: Modular normalization

▶ **Theorem 11** (Correctness of normalization). *For all terms* $\Gamma \vdash t : A$ *in* $\lambda_{SL}/\lambda_{SRL}/\lambda_{SJL}/\lambda_{LL}$, *there exists a normal form* $\Gamma \vdash_{NF} n : A$ *such that* $t \sim n$.

[Application] Future: Semantics for modalities in CbV

2020

Recovering Purity with Comonads and Capabilities

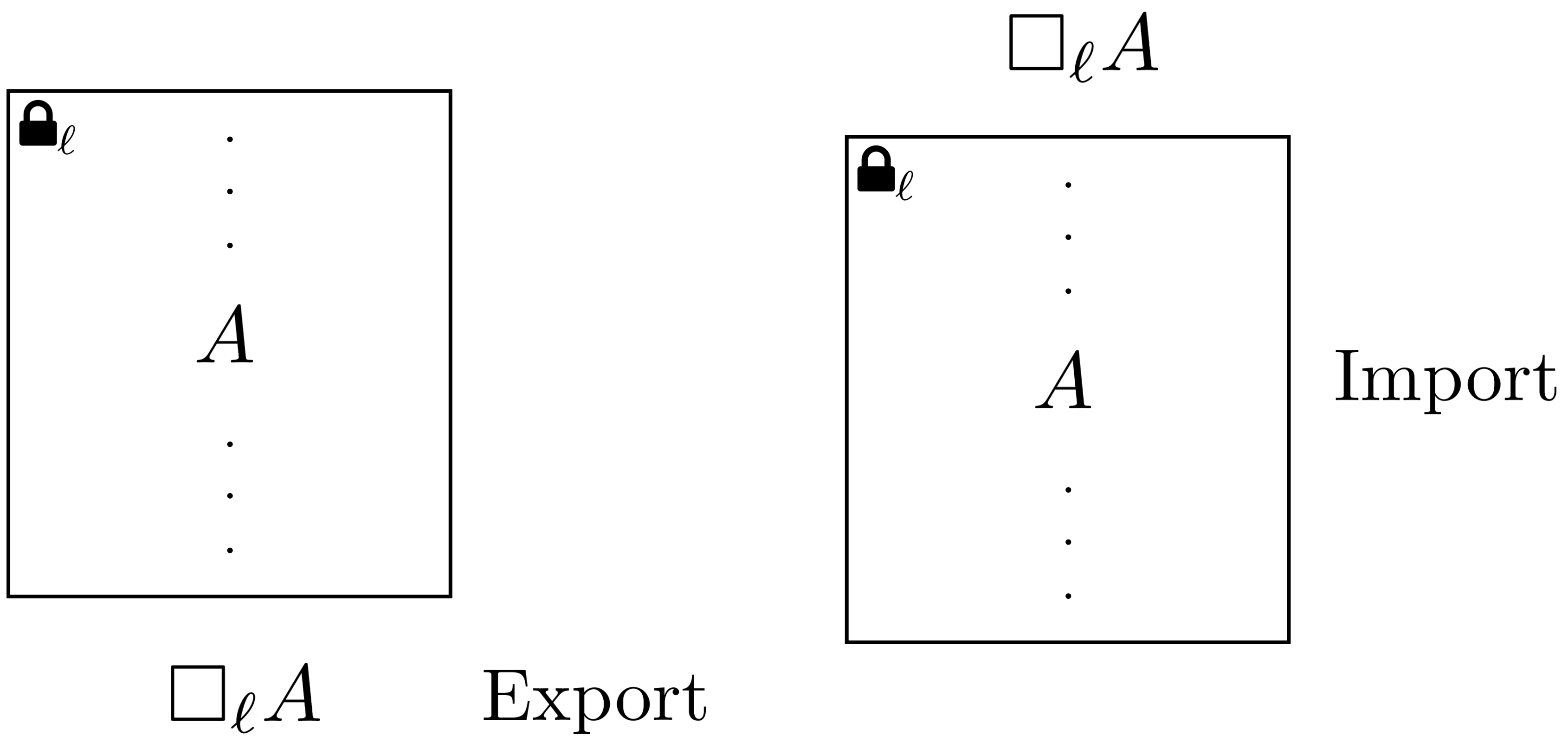
VIKRAMAN CHOUDHURY, Indiana University, USA and University of Cambridge, UK
NEEL KRISHNASWAMI, University of Cambridge, UK

In this paper, we take a pervasively effectful (in the style of ML) typed lambda calculus, and show how to *extend* it to permit capturing pure expressions with types. Our key observation is that, just as the pure simply-typed lambda calculus can be extended to support effects with a monadic type discipline, an impure typed lambda calculus can be extended to support purity with a *comonadic* type discipline.

We establish the correctness of our type system via a simple denotational model, which we call the *capability space* model. Our model formalises the intuition common to systems programmers that the ability to perform effects should be controlled via access to a permission or capability, and that a program is *capability-safe* if it performs no effects that it does not have a runtime capability for. We then identify the axiomatic categorical structure that the capability space model validates, and use these axioms to give a categorical semantics for our comonadic type system. We then give an equational theory (substitution and the call-by-value β and η laws) for the imperative lambda calculus, and show its soundness relative to this semantics.

Finally, we give a translation of the pure simply-typed lambda calculus into our comonadic imperative calculus, and show that any two terms which are $\beta\eta$-equal in the STLC are equal in the equational theory of the comonadic calculus, establishing that pure programs can be mapped in an equation-preserving way into

[Application] Future: IFC in an impure language



Compositional Normalisation with Modal Types

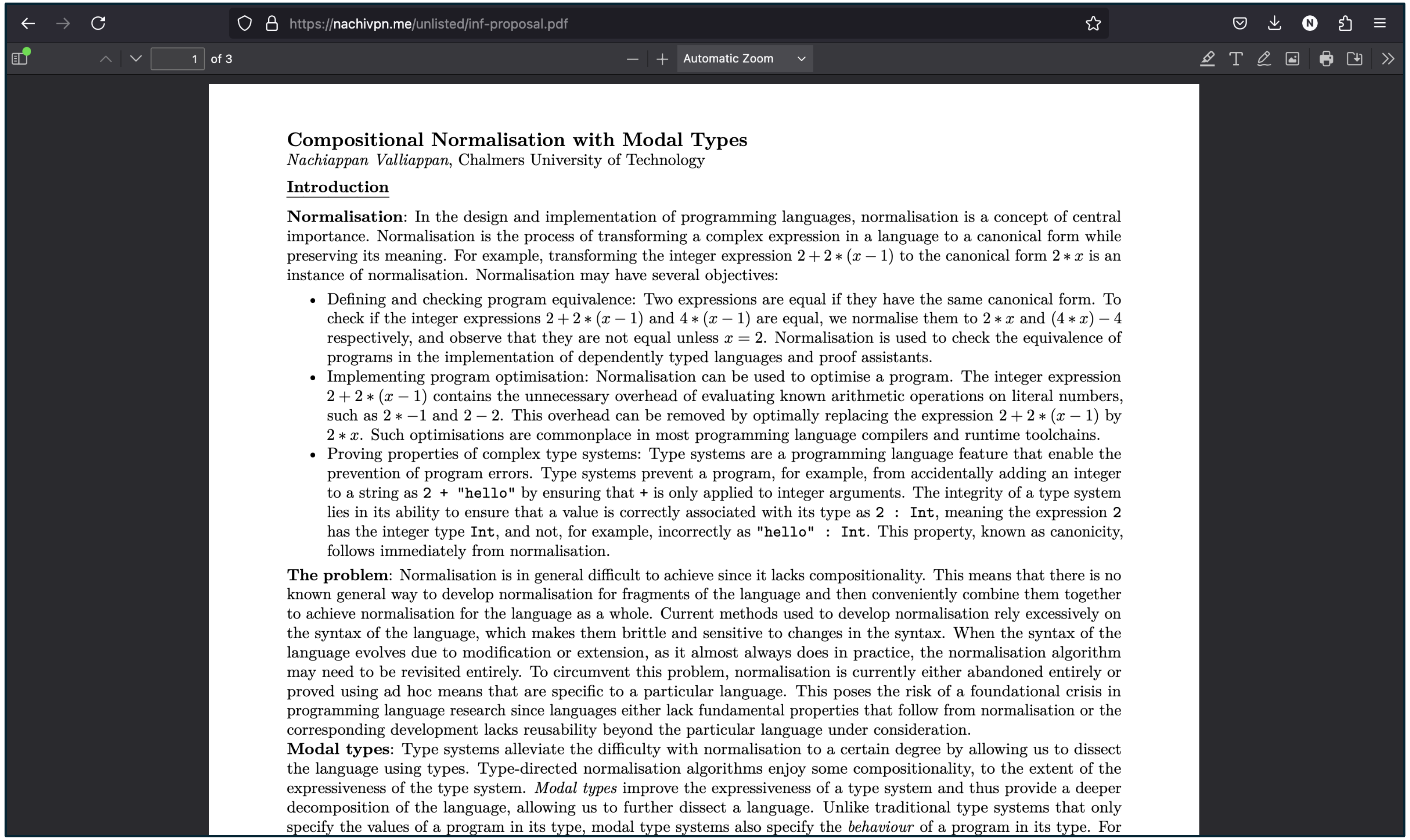
Valliappan, Nachi (Principal Investigator), Lindley, Sam (Sponsor)

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