A reading of justification logic modality in Curry-Howard fashion

Konstantinos Pouliasis

December 3, 2017

CUNY, Graduate Center

Introduction

Curry–Howard Isomorphism

Sparked from the investigation of the connection of explicit proofs in intuitionistic logic and programs of a simple programming language (*Simply Typed Lamda Calculus*)

Central topic of study in the field of type theory

Standard theoretical tool for studying and designing programming languages

CHI and logic

A paradigm that emphasizes proof relevance

The view of propositions–as–types has been extended to many logics

As a result it can capture more complex programming language features. (**scalability**)

Justification logic

A logic of (explicitly) witnessed modality

Started as a solution to the problem of providing constructive semantics to intuitionistic logic via explicit provability

A proof relevant analysis of $S4$ modality

Scales to other systems of modal logic

Applications in epistemology, epistemic game theory and programming language theory

Intuitionistic Logic and STLC

Intuitionism and constructive mathematics

Continuation of Brouwer's program

- ! Notion of truth collides with the notion of proof

Mathematical reasoning and logic is a human faculty

- ! The creative subjects use mathematical language as a means to express their reasoning (proof) constructs
- ! There are open problems
- ! Propositions do not come with a pre-existing truth value
  - !! Default absence of excluded middle
- ! As Heraclitus puts it truth is "an homodoxy (common belief) that each can individually witness"

Logic within constructive mathematics

The object of logic is the study of "sane" proof constructs

The traditional chasm between "syntax" and "semantics" is no longer very fruitful at least when engineering new logics.

Verificationist

The meaning of a connective is the ways that one can prove it

Pragmatist

The meaning of a connective is the way one can use its proofs

Example: rules $\wedge$

$$\dfrac{\begin{array}{cc}\mathcal{D} & \mathcal{E} \\ A & B\end{array}}{A \wedge B}\ \textsc{Intro}$$

$$\dfrac{\begin{array}{c}\mathcal{D} \\ A \wedge B\end{array}}{A}\ \textsc{Elim1} \qquad \dfrac{\begin{array}{c}\mathcal{D} \\ A \wedge B\end{array}}{B}\ \textsc{Elim2}$$

Proof equality and harmony

Proof trees are objects of logic and - as mathematical objects - are equipped with equality

Logic is algebraized

Equalities should provide for the "harmony" of the Introduction and Elimination rules (constructors destructors)

Harmony

Elim rules are not too strong a.k.a Local Soundness

$$\dfrac{\dfrac{\begin{array}{cc}\mathcal{D} & \mathcal{E} \\ A & B\end{array}}{A \wedge B}\,\textsc{Intro}}{A}\,\textsc{Elim1} \quad = \quad \begin{array}{c}\mathcal{D} \\ A\end{array}$$

$$\dfrac{\dfrac{\begin{array}{cc}\mathcal{D} & \mathcal{E} \\ A & B\end{array}}{A \wedge B}\,\textsc{Intro}}{B}\,\textsc{Elim2} \quad = \quad \begin{array}{c}\mathcal{E} \\ B\end{array}$$

Harmony

Elim rules are not too weak a.k.a Local completeness

$$\begin{array}{c}\mathcal{D}\\ A \wedge B\end{array} = \dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ A \wedge B\end{array}}{A}\,\textsc{Elim1} \qquad \dfrac{\begin{array}{c}\mathcal{D}\\ A \wedge B\end{array}}{B}\,\textsc{Elim2}}{A \wedge B}\,\textsc{Intro}$$

Example: $\supset$ rules

$$\dfrac{\dfrac{\overline{x : A}}{\mathcal{D}}}{\dfrac{A}{A \supset B}\,\supset^{x}} \qquad \dfrac{\mathcal{D} \quad \mathcal{E}}{\dfrac{A \supset B \qquad A}{B}}$$

Harmony

Not too strong

$$\dfrac{\dfrac{\overline{x : A}}{\mathcal{D}}}{\dfrac{B}{A \supset B}} \quad \dfrac{\mathcal{E}}{A}}{B} = \begin{array}{c} \mathcal{E} \\ A \\ \mathcal{D} \\ B \end{array}$$

Not too weak

$$\begin{array}{c} \mathcal{D} \\ A \supset B \end{array} = \dfrac{\dfrac{\mathcal{D}}{A \supset B} \quad \dfrac{\overline{x : A}}{}}{B}}{A \supset B}$$

Calculi for free

trees of A with open $\Gamma \mapsto$ terms ($\Gamma \vdash M : A$) (inhabitation judgments)

local soundness $\mapsto$ β reduction (equality judgments),

local completeness $\mapsto$ η expansion (equality judgments)

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A}\ \textsc{Refl} \qquad \frac{}{\Gamma \vdash \langle\rangle : \top}\ \top\mathrm{I}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash M' : B}{\Gamma \vdash \langle M, M' \rangle : A \times B}\ \textsc{Tup}$$

$$\frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \mathsf{fst}(M) : A}\ \mathrm{LPrj} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \mathsf{snd}(M) : B}\ \mathrm{RPrj}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.\ M : A \to B}\ \lambda{-}\textsc{Abs}$$

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash M' : A}{\Gamma \vdash (MM') : B}\ \textsc{App}$$

Plus equality judgments e.g.

$$\frac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash N : A}{\Gamma \vdash (\lambda x : \phi.M)(N) =_\beta [N/x]M : B}\ \beta \supset$$

$$\frac{\Gamma \vdash f : A \supset B}{\Gamma \vdash f =_\eta \lambda x : A.\ (fx) : B}\ \eta \supset$$

plus congruence axiomatization

Metatheory for (almost) free

The good old way

- ! Forget about variables/ terms/ equalities and prove that $\Gamma \vdash A$ has an order theoretic model ($\Gamma^* \leq \phi^*$)

Cut elimination (core of CHI)

- ! Create restricted versions of your calculus/ proof trees:
  - !! do not permit Elim - After- Intro/ β redexes
- ! Prove that your restricted system of proof trees proves the same stuff by inventing a corresponding sequent calculus and proving cut elimination
  - !! consistency (no normal form for bottom)
- !! equality means equi-normalizibility (together with Church Rosser)

JCalc: Motivation

Motivation

$$\frac{\vdash A}{\vdash \Box A}\ \text{NEC}$$

$$\frac{\vdash A \qquad \vdash \mathsf{JA}}{\vdash \Box A}\ \Box_J$$

In JL the motivation is that $\vdash A$ is Int and JA is in CL so J can be seen as an embedding.

In general the (justified) connective can be treated independently of the details of the two deductive systems

"Necessitation stems from internalization/ validation"

Deductive systems

A deductive system is a $\Gamma \vdash \phi$ relation with the following (minimal) properties:

Reflexivity :

* $A \in \Gamma \Longrightarrow \Gamma \vdash_I A$

Compositionality:

* $\Gamma \vdash A$ and $\Gamma', A \vdash B \Longrightarrow \Gamma, \Gamma' \vdash B$

Top element:

* $\Gamma \vdash \top$

Completeness in deductive systems

A deductive system J is complete for I (or a conservative extension) iff there exists mapping $[\![\bullet]\!] : U_I \to U_J$ on propositions of I into propositions of J s.t.

* $\Gamma \vdash_I A \Longrightarrow [\![\Gamma]\!] \vdash_J [\![A]\!]$
* $[\![\top_I]\!] = \top_J$
* For Γ of the form $A_1, \ldots, A_n$:

$$[\![\Gamma]\!] = [\![A_1]\!], \ldots, [\![A_n]\!]$$

$\Box$ as double theoremhood

When J is complete with respect to I then double theoremhood $\langle \vdash A, \vdash [\![A]\!] \rangle$ is a basic modality

! Intuitively $A \vdash B$ implies $[\![A]\!] \vdash [\![B]\!]$ and thus a double proof of A ($\Box A$) gives a double proof of $B(\Box B)$

! The basic rule of K modality is admissible

$$\frac{A \vdash B}{\Box A \vdash \Box B}$$

My thesis in a slide

- A natural deduction with three kinds of judgments:
  - $\Gamma \vdash A$ that follows intuitionistic logic
  - $[\![\Gamma]\!] \vdash [\![A]\!]$ that axiomatizes the notion of conservative extension
  - $\Box\Gamma \vdash \Box A$ for "double theoremhood"
- Its order theoretic semantics
- Analysis of the new connective a la Gentzen to get
  - proof terms/trees and equalities/ reductions
- Its Cut elimination/ normalization
- An implementation in the metatheoretic framework "Makam" (work in progress)
- Some good reasons to think that this idea is extensible

Jcalc: provability (the good old way)

Type Universe

$$\frac{}{P_k \in \mathsf{Prop}_0}\ \textsc{Atom} \qquad \frac{}{\top \in \mathsf{Prop}_0}\ \textsc{Top}$$

$$\frac{A \in \mathsf{Prop}_i \qquad B \in \mathsf{Prop}_i}{A \wedge B \in \mathsf{Prop}_i}\ \textsc{Conj} \qquad \frac{A \in \mathsf{Prop}_0}{\Box A \in \mathsf{Prop}_1}\ \textsc{Box}$$

$$\frac{A \in \mathsf{Prop}_i \qquad B \in \mathsf{Prop}_i}{A \supset B \in \mathsf{Prop}_i}\ \textsc{Arr} \qquad \frac{A \in \mathsf{Prop}_0}{[\![A]\!] \in [\![\mathsf{Prop}_0]\!]}\ \textsc{Brc}$$

Natural deduction for Prop_0

$$\frac{A \in \Gamma}{\Gamma \vdash A}\ \Gamma_0\text{-}\textsc{Refl} \qquad \frac{}{\Gamma \vdash \top}\ \top_0\mathrm{I}$$

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge_0\mathrm{I} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge_0\mathrm{E1} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge_0\mathrm{E2}$$

$$\frac{\Gamma, x : A \vdash B}{\Gamma \vdash A \supset B}\ \supset_0\mathrm{I} \qquad \frac{\Gamma \vdash A \supset B \qquad \Gamma \vdash A}{\Gamma \vdash B}\ \supset_0\mathrm{E}$$

Natural deduction for $[\![\mathsf{Prop}_0]\!]$

$$\frac{[\![A]\!] \in \Delta}{\Delta \vdash [\![A]\!]}\ \Delta\text{-}\textsc{Refl} \qquad \frac{}{\Delta \vdash [\![\top]\!]}\ \mathrm{Ax}_1$$

$$\frac{A, B \in \mathsf{Prop}_0}{\Delta \vdash [\![A \supset (B \supset A)]\!]}\ \mathrm{Ax}_2$$

$$\frac{A, B, C \in \mathsf{Prop}_0}{\Delta \vdash [\![(A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))]\!]}\ \mathrm{Ax}_3$$

$$\frac{A, B \in \mathsf{Prop}_0}{\Delta \vdash [\![A \supset (B \supset A \wedge B)]\!]}\ \mathrm{Ax}_4 \qquad \frac{A, B \in \mathsf{Prop}_0}{\Delta \vdash [\![A \wedge B \supset A]\!]}\ \mathrm{Ax}_5$$

$$\frac{A, B \in \mathsf{Prop}_0}{\Delta \vdash [\![A \wedge B \supset B]\!]}\ \mathrm{Ax}_6$$

$$\frac{\Delta \vdash [\![A \supset B]\!] \qquad \Delta \vdash [\![A]\!]}{\Delta \vdash [\![B]\!]}\ \mathrm{MP}$$

Natural Deduction (Prop_1) with $\Gamma \in \mathrm{Prop}_1$

$$\frac{\Box A \in \Gamma}{\Gamma \vdash \Box A}$$

$$\frac{(\forall A_i \in \Gamma'.\ \Gamma \vdash \Box A_i) \qquad \Gamma' \vdash B \qquad [\![\Gamma']\!] \vdash [\![B]\!]}{\Gamma \vdash \Box B}$$

$$\frac{\Gamma, \Box A \vdash \Box B}{\Gamma \vdash \Box A \supset \Box B}\ \supset_1 \mathrm{I} \qquad \frac{\Gamma \vdash \Box A \supset \Box B \qquad \Gamma \vdash \Box A}{\Gamma \vdash \Box B}\ \supset_1$$

$$\frac{\Gamma \vdash \Box A \qquad \Gamma \vdash \Box B}{\Gamma \vdash \Box A \wedge \Box B}\ \wedge_1 \mathrm{I}$$

$$\frac{\Gamma \vdash \Box A \wedge \Box B}{\Gamma \vdash \Box A}\ \wedge_1 \mathrm{E1} \qquad \frac{\Gamma \vdash \Box A \wedge \Box B}{\Gamma \vdash \Box B}\ \wedge_1 \mathrm{E2}$$

Basic facts

- Admissibility of deduction theorem or, $[\![\bullet]\!]$ preserves $\supset$:
  $[\![\Gamma]\!], [\![A]\!] \vdash [\![B]\!] \Longrightarrow [\![\Gamma]\!] \vdash [\![A \supset B]\!]$,
- Logical completeness:
  $\forall \Gamma, A \in \mathsf{Prop}_0 \;\; \Gamma \vdash A \Longrightarrow [\![\Gamma]\!] \vdash [\![A]\!]$,
- Admissibility of (standard) K rule: $\forall \Gamma, A \in \mathsf{Prop}_0$
  $\Gamma \vdash A \Longrightarrow \Box\Gamma \vdash \Box A$,
- Admissibility of Nec: $\vdash A \Longrightarrow \vdash \Box A$

Jcalc: Order Theory

Semi HAs

- A *(meet) semi-lattice* is a non-empty *partial order* (i.e. reflexive, antisymmetric and transitive) with finite meets.
  - $a \times b \leq a$ and $a \times b \leq b$
  - $a \times b$ is the greatest such element:
    $c \leq a$ and $c \leq b \Longrightarrow c \leq a \times b$
- A *bounded (meet) semi-lattice* $(L, \leq)$ is a (meet) semi-lattice that additionally has
  - a greatest element (we name it 1), which satisfies
    $c \leq 1$ for every x in L

! A *semi-HA* is a bounded (meet) semi-lattice $(L, \leq, 1)$ s.t. for every $a, b \in L$ there exists an *exponential* (we name it $a \to b$) with the properties:

* $a \to b \times a \leq b$
* $a \to b$ is the greatest such element :
  $c \times a \leq b \Longrightarrow c \leq a \to b$

"Natural" order preserving functions

Definition A function F between two (semi)-HAs (H_1, H_2) is order preserving and commutes with top, products and exponentials *iff* for every $\phi, \psi \in H_1$

1. $\phi \leq_{H_1} \psi \Rightarrow F\phi \leq_{H_2} F\psi$
2. $F\top_{H_1} = \top_{H_2}$
3. $F(\phi \times_{H_1} \psi) = F(\phi) \times_{H_2} (F(\psi)$
4. $F(\phi \to_{H_1} \psi) = F(\psi) \to_{H_2} F(\phi)$

Definition

A *Jcalc-triplet* is

1. A semi-Heyting algebra HA
2. A partial order J
3. An order preserving function F from H to J s.t.
   - 3.1 The image $F(HA)$ forms a semi-Heyting Algebra
   - 3.2 F preserves top, products and exponentials

$\Box^F HA$ pair algebra

Definition

Given a *Jcalc*−triplet (HA, F, J) define the induced algebra of F-points: $\Box^F HA$:

1. elements are pairs $\langle A, FA \rangle$ (name them, $\Box^F A$)
2. $\Box^F A \leq \Box^F B$ iff $A \leq B$ and $FA \leq FB$

We can show:

* It is reflexive, transitive and antisymmetric (a *partial* order)
* It is a (semi) Heyting algebra with:
    1. $\Box^F \top$ being the top element (name it $\top_{\Box^F}$)
    2. For any two elements $\Box^F A$, $\Box^F B$: $\Box^F (A \times B)$ is their product (name it, $\Box^F A \times \Box^F B$)
    3. For any two elements $\Box^F A$, $\Box^F B$: $\Box^F (A \to B)$ is their exponential (name it, $\Box^F A \to \Box^B A$)

Jcalc Algebra, Soundness and Completeness

Definition Given a *Jcalc* triplet unionize the $\leq$s (HA, $F(HA)$, $\Box^F HA$). We call the result *Jcalc*-algebra

Theorem

Jcalc is sound and complete w.r.t Jcalc algebras :

$\Gamma \vdash_{Jcalc} \phi$ *iff for any Jcalc Algebra JC* (HA, F, J) $\Gamma^+ \leq \phi^*$

With $(\bullet)*$ any map that extends a mapping of atomic propositions to elements of HA with the following properties:

$$
\begin{aligned}
(\top)* &= \top \\
(A \wedge B \in Prop_0)* &= A* \times_{HA} B* \\
(A \supset B \in Prop_0)* &= A* \to_{HA} B* \\
([\![A]\!])* &= F(A*) \\
(\Box A)* &= \Box^F A* \\
(\Box A \supset \Box B)* &= \Box^F A* \to \Box^F B* \\
(\Box A \wedge \Box B)* &= \Box^F A* \times \Box^F B*
\end{aligned}
$$

$$
\begin{aligned}
(\Gamma, \phi)^+ &= (\Gamma)^+ \times (\phi)* \\
(\circ)^+ &= \top
\end{aligned}
$$

Term calculus for Jcalc

K-$\Box$ rule

One modal rule for elimination and introduction to relate judgments A valid, A true, $\Box A$true:

$$\dfrac{\begin{array}{ccc} & \overline{x : A} & \overline{s : [\![A]\!]} \\ & \mathcal{E} & \mathcal{F} \\ \mathcal{D} & B & [\![B]\!] \\ \Box A & & \end{array}}{\Box B}$$

Modal Rule (Generically)

Or, generalizing for assumptions

$\Gamma := x_1 : A_1, \ldots, x_i : A_i$ and $\Delta := s_1 : [\![A_i]\!], \ldots s_i : [\![A_i]\!]$:

$$\dfrac{\begin{array}{ccc} & \overline{\Gamma} & \overline{\Delta} \\ & \vdots & \vdots \\ \Box A_1, \ldots, \Box A_i & B & [\![B]\!] \end{array}}{\Box B}$$

Modal Rule ($\Box$ Introduction)

We defined the connective negatively but we obtain $\Box$ constuctors by the very same rule for Γ, Δ empty and derivations $\overline{\mathcal{D}}, \overline{\mathcal{E}}$ closed for substitutions.

$$\dfrac{\begin{array}{cc}\overline{\mathcal{D}} & \overline{\mathcal{E}} \\ B & [\![B]\!]\end{array}}{\Box B}$$

Proof term assignment: subcases of the rule

$\Gamma \in \mathsf{Prop}_1$

$$\frac{(\forall A_i \in \Gamma'.\ \Gamma \vdash \Box A_i) \qquad \Gamma' \vdash B \qquad [\![\Gamma']\!] \vdash [\![B]\!]}{\Gamma \vdash \Box B}$$

$$\frac{\vdash B \qquad \vdash [\![B]\!]}{\Gamma \vdash \Box B}\ \Gamma'\ \textsc{empty}$$

$$\frac{\vdash M : B \qquad \vdash J : [\![B]\!]}{\Gamma \vdash M\&J : \Box B}\ \Gamma'\ \textsc{empty}$$

let **bindings**

Standard construct in modern programming languages

! e.g. `let (x,y) = (2,4) in (x+1,y+1)`

  * The expression above has a β-redex and it reduces:
    `let (x,y) = (2,4) in (x+1,y+1)` $\rightarrow_\beta$
    `(x+1,y+1)[2/x][3/y]` $\rightarrow_\beta$
    `(3,4)`

! We obtain similar terms and semantics for the $\Box$ rules applying the slogans

Proof term assignment: subcases of the rule

$\Gamma \in \mathsf{Prop}_1$

$$\frac{(\forall A_i \in \Gamma'.\ \Gamma \vdash \Box A_i) \qquad \Gamma' \vdash B \qquad [\![\Gamma']\!] \vdash [\![B]\!]}{\Gamma \vdash \Box B}$$

$$\frac{\Gamma \vdash \Box A \qquad x : A \vdash B}{\Gamma \vdash \Box B}\ \Gamma' \text{ UNARY}$$

$$\frac{\Gamma \vdash N : \Box A \qquad x : A \vdash M : B \qquad s : [\![A]\!] \vdash \mathsf{J} : [\![B]\!]}{\Gamma \vdash \mathsf{let}\ (x \& s\ \ \mathsf{be}\ N)\ \mathsf{in}\ (M \& \mathsf{J}) : \Box B}\ \Gamma' \text{ UNARY}$$

Local Soundness

For $\overline{\mathcal{D}}, \overline{\mathcal{E}}$ closed for substitutions, the first proof tree is reducible to the second:

$$\dfrac{\dfrac{\overline{\mathcal{D}}}{A} \quad \dfrac{\overline{\mathcal{E}}}{[\![A]\!]}}{\Box A} \quad \dfrac{\overline{x : A}}{\vdots\ B} \quad \dfrac{\overline{s : [\![A]\!]}}{\vdots\ [\![B]\!]}}{\Box B} \Longrightarrow$$

$$\dfrac{\overline{\mathcal{D}}}{\overline{A}}\ \vdots\ B \qquad \dfrac{\overline{\mathcal{E}}}{\overline{[\![A]\!]}}\ \vdots\ [\![B]\!]}{\Box B}$$

β reduction

$$\Box_I \dfrac{\dfrac{\vdash M : A \qquad \vdash \mathsf{J} : [\![A]\!]}{\Gamma \vdash M \& \mathsf{J} : \Box A} \qquad x : A \vdash M' : B \qquad s : [\![A]\!] \vdash \mathsf{J}' : [\![B]\!]}{\Gamma \vdash \mathsf{let}\ \ (x \& s)\ \ \mathsf{be}\ \ (M \& \mathsf{J})\ \ \mathsf{in}\ \ (M' \& \mathsf{J}') : \Box B}$$

$$\Longrightarrow_R$$

$$\dfrac{\vdash M'[M/x] : B \qquad \vdash \mathsf{J}'[\mathsf{J}/s] : [\![B]\!]}{\Gamma \vdash M'[M/x] \& \mathsf{J}'[\mathsf{J}/s] : \Box B}$$

β reduction

$$\Box_I \frac{\dfrac{\vdash M : A \qquad \vdash \mathsf{J} : [\![A]\!]}{\Gamma \vdash M\&\mathsf{J} : \Box A} \qquad x : A \vdash M' : B \qquad s : [\![A]\!] \vdash \mathsf{J}' : [\![B]\!]}{\Gamma \vdash \mathsf{let}\ \ (x\&s)\ \ \mathsf{be}\ \ (M\&\mathsf{J})\ \ \mathsf{in}\ \ (M'\&\mathsf{J}') : \Box B}$$

$$\Longrightarrow_R$$

$$\frac{\vdash M'[M/x] : B \qquad \vdash \mathsf{J}'[\mathsf{J}/s] : [\![B]\!]}{\Gamma \vdash M'[M/x]\&\mathsf{J}'[\mathsf{j}/s] : \Box B}$$

Which gives $\beta-$equality rule:

$$\frac{\vdash M : A \qquad \vdash \mathsf{J} : [\![A]\!] \qquad x : A \vdash M' : B \qquad s : [\![A]\!] \vdash \mathsf{J}' : [\![B]\!]}{\Gamma \vdash \mathsf{let}\ \ (x\&s)\ \ \mathsf{be}\ \ (M\&\mathsf{J})\ \ \mathsf{in}\ \ (M'\&\mathsf{J}') =_\beta M'[M/x]\&\mathsf{J}'[\mathsf{J}/s] : \Box B}$$

Generalizing for the $\Box$ rule

Or, generically, for arbitrary length of Γ':

$$\frac{\forall A_k \in \Gamma'.\ \Gamma \vdash N_k : \Box A_k \qquad \Gamma' \vdash M : B \qquad [\![\Gamma']\!] \vdash \mathsf{J} : [\![B]\!]}{\Gamma \vdash \mathsf{let}^*\ (\Gamma', [\![\Gamma']\!], Ns)\ \mathsf{in}\ (M \& \mathsf{J}) : \Box B}$$

With :

$$\mathsf{let}^*\ (\circ;\ \circ;\ [\,])\ \mathsf{in}\ M := M$$

$$\mathsf{let}^*\ (x_1 : A_1, \ldots, x_i : A_i\ ;\ s_1 : [\![A_1]\!], \ldots, s_i : [\![A_i]\!];\ N_1, \ldots, N_i)\ \mathsf{in}\ M$$

$$:=$$

$$\mathsf{let}\ \{(x_1 \& s_1)\ \mathsf{be}\ N_1, \ldots, (x_i \& s_i)\ \mathsf{be}\ N_i\}\ \mathsf{in}\ M$$

With similar β reduction rule (i double substitutions).

Local Completeness

Any deduction $\mathcal{D} : \Box A$ can be expanded as follows:

$$\begin{array}{c} \mathcal{D} \\ \Box A \end{array} \qquad \Longrightarrow$$

$$\dfrac{\begin{array}{ccc} \mathcal{D} & & \\ \Box A & \overline{x : A} & \overline{s : [\![A]\!]} \end{array}}{\Box A}$$

Which gives the η equality judgment:

$$\dfrac{\Gamma \vdash M : \Box A}{\Gamma \vdash M =_\eta \text{let } (x \& s) \text{ be } M \text{ in } x \& s : \Box A}$$

The rest of the system

Proof terms for the rest of the system are standard

! For the $[\![\mathsf{Prop}_0]\!]$ part we deploy justification logic:

  ! Axioms get assigned to (parametrized) constants

  !! Modus ponens is internalized by $*$ operation on justifications

! For the rest of Prop_1 trivially from the lambda calculus

The implicative fragment of $[\![\mathsf{Prop}_0]\!]$

$$\frac{s : [\![A]\!] \in \Delta}{\Delta \vdash s : [\![A]\!]}\ \Delta\text{-}\textsc{Refl} \qquad \frac{}{\Delta \vdash C_{\top} : [\![\top]\!]}\ \textsc{Ax}_1$$

$$\frac{A, B \in \mathsf{Prop}_0}{\Delta \vdash C_{K^{A,B}} : [\![A \supset (B \supset A)]\!]}\ \textsc{Ax}_2$$

$$\frac{A, B, C \in \mathsf{Prop}_0}{\Delta \vdash C_{S^{A,B,C}} : [\![(A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))]\!]}\ \textsc{Ax}_3$$

$$\frac{\Delta \vdash \mathsf{J} : [\![A \supset B]\!] \qquad \Delta \vdash \mathsf{J}' : [\![A]\!]}{\Delta \vdash \mathsf{J} * \mathsf{J}' : [\![B]\!]}\ \textsc{App}$$

Metatheory for the term system

! We have proven cut elimination for natural deduction (which implies strong normalization of proof trees / termination of β reduction)

! We have proven termination under *call-by-value* semantics seperately

Call by value

$$\frac{}{\lambda x.M \text{ value}} \qquad \frac{}{C_i \text{ value}} \qquad \frac{\mathsf{J}_1 \text{ value} \qquad \mathsf{J}_2 \text{ value}}{\mathsf{J}_1 * \mathsf{J}_2 \text{ value}}$$

$$\frac{M \text{ value} \qquad \mathsf{J} \text{ value}}{M\&\mathsf{J} \text{ value}} \qquad \frac{M \to M'}{M\&\mathsf{J} \to M'\&\mathsf{J}}$$

$$\frac{N_1 \text{ value} \quad \ldots \quad N_{k-1} \text{ value} \qquad N_k \to N_k'}{\begin{array}{c}\text{let}\{(x_1 \& s_1) \text{ be } N_1, \ldots, \ (x_k \& s_k) \text{ be } N_k, \ldots\} \text{ in } M \to \\ \text{let}\{(x_1 \& s_1) \text{ be } N_1, \ldots, \ (x_k \& s_k) \text{ be } N_k', \ldots\} \text{ in } M\end{array}}$$

$$\frac{M_1 \& \mathsf{J}_1 \text{ value} \quad \ldots \quad M_i \& \mathsf{J}_\mathsf{i} \text{ value}}{\begin{array}{c}\text{let}\{(x_1 \& s_1) \text{ be } (M_1 \& \mathsf{J}_1), \ldots, (x_i \& s_i) \text{ be } (M_i \& \mathsf{J}_\mathsf{i})\} \text{ in } (M \& \mathsf{J}) \to \\ M[M_1/x_1, \ldots, M_i/x_i] \& \mathsf{J}[\mathsf{J}_1/s_1, \ldots, \mathsf{J}_\mathsf{i}/s_i]\end{array}}$$

$$\frac{M \to M'}{(MN) \to (M'N)} \qquad \frac{N \to N'}{((\lambda x.M)N) \to ((\lambda x.M)N')}$$

$$\frac{N \text{ value}}{((\lambda x.M)N) \to [N/x]M}$$

Dynamic linking in presence of typed dependencies

Set up

Consider a language I (source) interfacing another language J (target) via a declared signature Σ.

! We assume that the two languages share some basic types e.g. int

* e.g $[\![int_I]\!] = int_J$
* function space $[\![A \to_I B]\!] = [\![A]\!] \to_J [\![B]\!]$
* pair types $[\![A \times_I B]\!] = [\![A]\!] \times_J [\![B]\!]$

Semantics for source terms

A typical interface for a stack will be declared as follows:

```
using type intstack
  empty:  intstack
  push:  int -> intstack -> intstack
  pop:  intstack -> int x intstack
```

- ! Source expressions in such a setting have a dynamic meaning depending on the target implementations
- ! E.g. the term
  $\Sigma; \circ\vdash \text{pop}(\text{push}\ (1+1)\ \text{Empty}) \rightsquigarrow \text{pop}(\text{push}\ 2\ \text{Empty}) : \text{int}$
- ! Its continuation depends on the implementations of the signature

Dynamic linking

- ! A dynamic linker is a metaprogramming construct that consumes a type correct implementation of the signature and produces a type correct residual program sensitive to the implementation.
- ! Avoid type checking of the whole source in presence of new implementations
- ! Our calculus, is a calculus of such constructions.

Dynamic linker construction

Here are the steps towards the dynamic linker construction for the expression:

! Contextualize the use of the signature:

$$\Sigma; \quad x_1 : \text{intstack}, x_2 : \text{int} \to \text{intstack} \to \text{int} \times \text{intstack}, \\ x_3 : \text{intstack} \to \text{int} \vdash x_3(x_2\ 2\ x_1) : \text{int}$$

Dynamic linker construction

! Rewrite the previous judgment assuming references to foreign implementations for the corresponding missing elements.

$$\mathsf{s}_1 : [\![\mathsf{intstack}]\!], \mathsf{s}_2 : [\![\mathsf{int} \to \mathsf{intstack} \to \mathsf{intstack}]\!],$$
$$\mathsf{s}_3 : [\![\mathsf{intstack} \to \mathsf{int} \times \mathsf{intstack}]\!] \vdash \mathsf{s}_3 * (\mathsf{s}_2 * \ \bar{2} \ * \mathsf{s}_1) : [\![\mathsf{int}]\!]$$

! Merge the two judgments to construct a dynamic linker :

$$\Sigma; \mathsf{x}'_1 : \Box\mathsf{intstack}, \mathsf{x}'_2 : \Box(\mathsf{int} \to \mathsf{intstack} \to \mathsf{intstack}),$$
$$\mathsf{x}'_3 : \Box(\mathsf{intstack} \to \mathsf{int} \times \mathsf{instack}) \vdash$$
$$\mathsf{let}\{\mathsf{x}_1 \& \mathsf{s}_1 \text{ be } \mathsf{x}'_1,\ \mathsf{x}_2 \& \mathsf{s}_2 \text{ be } \mathsf{x}'_2,\ \mathsf{x}_3 \& \mathsf{s}_3 \text{ be } \mathsf{x}'_3\} \text{ in}$$
$$(\mathtt{x}_3(\mathtt{x}_2\ 2\ \mathtt{x}_1)\ \&\ \mathtt{s}_3 * (\mathtt{s}_2 * \bar{2} * \mathtt{s}_1)) : \Box\mathtt{int}$$

Dynamic linker constuction

Which gives by λ abstractions:

$$\lambda \mathtt{x}'_1.\ \lambda \mathtt{x}'_2.\lambda \mathtt{x}'_3.$$
$$\mathsf{let}\{\mathtt{x}_1\&\mathtt{s}_1 \text{ be } \mathtt{x}'_1,\ \mathtt{x}_2\&\mathtt{s}_2 \text{ be } \mathtt{x}'_2,\ \mathtt{x}_3\&\mathtt{s}_3 \text{ be } \mathtt{x}'_3\} \text{ in}$$
$$(\mathtt{x}_3(\mathtt{x}_2\ 2\ \mathtt{x}_1)\ \&\ \mathtt{s}_3 * (\mathtt{s}_2 * \bar{2} * \mathtt{s}_1)) :$$
$$\Box(\mathsf{intstack}) \to \Box(\mathsf{int} \to \mathsf{intstack} \to \mathsf{intstack})$$
$$\to \Box(\mathsf{intstack} \to \mathsf{int} \times \mathsf{intstack}) \to \Box\mathsf{int}$$

- ! Such a program behaves correctly consuming **only** conforming implementations of the interface
- ! calculating the program's different futures with respect to different implementations.

  E.g. with an array implementation we get :

$$\begin{aligned}
&\text{create_array}() : \text{intarray} \quad \text{for} \quad \text{empty} \\
&\text{add_array} : \text{int} \rightarrow \text{intarray} \rightarrow \text{intarray} \quad \text{for} \quad \text{push} \\
&\text{remove_last} : \text{intarray} \rightarrow \text{intarray} \times \text{int} \ \text{for} \ \text{pop}
\end{aligned}$$

Conforming with the signature under the interpretation:

$$[\![\text{intstack}]\!] := \text{intarray}$$
$$[\![\text{int} \to \text{intstack} \to \text{instack}]\!] := \text{int} \to \text{intarray} \to \text{intarray}$$
$$[\![\text{intstack} \to (\text{int} \times \text{intstack})]\!] := \text{int} \to (\text{int} \times \text{intarray})$$

Hence we obtain values of $\Box$ types (bindings):

$$\text{empty\&\ create\_array}() : \Box\text{intstack}$$
$$\text{push\&\ add\_array} : \Box(\text{int} \to \text{intstack} \to \text{intstack})$$
$$\text{pop\&\ remove\_last} : \Box(\text{intarray} \to \text{intarray} \times \text{int})$$

If we apply these values to the dynamic linker we obtain the assignment for the whole expression: after reducing all redexes (λ and let):

$$\begin{array}{l}\mathsf{pop}(\mathsf{push}\ 2\ \mathsf{empty})\&\\ \mathsf{remove\_last} * (\mathsf{add\_array} * \ \bar{2} * \mathsf{create\_array}())\end{array}$$

As required in systems with separate compilation, the same process will compute a different residual with a different set of conforming implementations

implementation

Tools for free

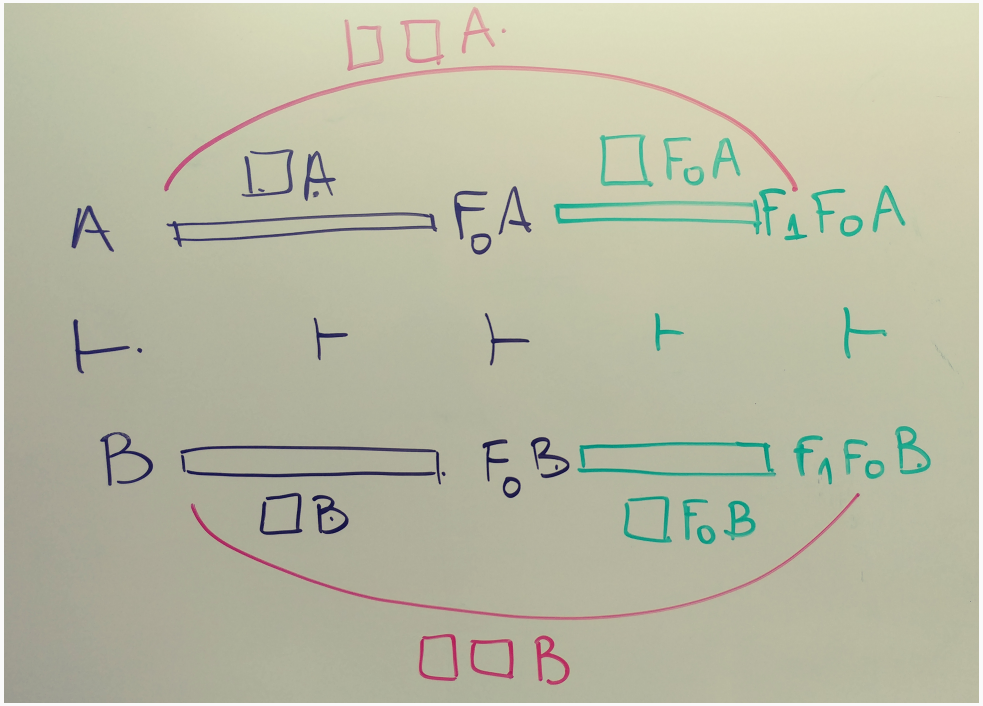
- ! Coq, LF are logic focused
- ! Makam, Beluga permit work from a programming language design point of view
- ! Makam is a meta programming framework for quick language prototyping

Makam implementation

- ! Antonis Stampoulis and I have encoded *Jcalc* in Makam without much effort!
- ! We are able to typecheck programs like the ones presented
- ! You can check the code on github: `https://github.com/KPouliasis/makamjs/blob/as-check-in-makam-js/just.md`

The future

Higher □ types?



Yes! Takes a bit more syntax and using the bang operator of justification logic

What about stronger modality?

A Galois connection is a pair of order preserving functions $L : \mathcal{D} \to C$, $R : \mathcal{C} \to D$ such that there is an isomorphism:
$\forall d \in D, c \in C.\ Ld \leq c \longleftrightarrow d \leq Rc$

Take the relation of classical and intuitionistic logic as a prototype:

$$\frac{[\![\Gamma]\!] \vdash_{class} \phi}{\Gamma \vdash_I \neg\neg\phi}\ \downarrow\uparrow$$

This is a typical case of a Galois connection (or an adjunction)
To mimic such a notion in *Jcalc* one should add another function symbol to correspond to R and add the rule:

$$\frac{[\![\Gamma]\!] \vdash_J \mathsf{j} : \phi}{\Gamma \vdash \mathsf{eval\_return}\ \ \mathsf{j} : R\phi}\ R$$

This is enough to obtain a generalized notion of Factivity that is:

$$\frac{\Gamma \vdash M : \Box\phi}{\Gamma \vdash \text{let } \_\_ \,\&\, \text{s be M} \;\; \text{in eval\_return(s)} : \text{R}[\![\phi]\!]}\, R$$

Ideas from the Artemov, Bonelli *Intensional Lambda calculus* could be utilized to add evaluation at the level of justifications

Concluding

Justification logic has rich higher order features and reflection!

I see this as a first part of a project for utilizing it to provide foundations of *typed programming language interaction*

Thank you!