Intersection, Union, Dependent Types and SubType Systems

Why after more than 40 yy the topic is still alive ...

Luigi Liquori, Inria, Sophia Antipolis Méditerranée

Intersection Types for pure λ-calculus [$\sigma \bigcap \tau$]

- *Ad hoc* polymorphism for the pure λ-calculus
- 40 years history: characterization of strongly normalizing λ-terms, λ-models, object-oriented programming, automatic type inference, type inhabitation, type unification, software product lines, etc
- Type inference is undecidable, subtyping is decidable

$$\frac{B \vdash M : \sigma \quad B \vdash M : \tau}{B \vdash M : \sigma \cap \tau} \ (\cap I) \qquad \frac{B \vdash M : \sigma \cap \tau}{B \vdash M : \sigma \ (\mathit{resp.}\ \tau)} \ (\cap E_{l/r})$$

$$\frac{B \vdash M : \sigma \quad \sigma \leqslant_{\mathcal{T}} \tau}{B \vdash M : \tau} \ (\leqslant) \qquad \frac{x{:}\sigma \in B}{B \vdash x : \sigma} \ (\mathit{Var})$$

$$\frac{B, x{:}\sigma \vdash M : \tau}{B \vdash \lambda x.M : \sigma \to \tau} \ (\to I) \qquad \frac{B \vdash M : \sigma \to \tau \quad B \vdash N : \sigma}{B \vdash M\,N : \tau} \ (\to E)$$

Intersection Types as a Theoretical Swiss Knife

Examples

- Polymorphic identity

$$\dfrac{\dfrac{x{:}\sigma \vdash x : \sigma}{\vdash \lambda x.x : \sigma \to \sigma} \qquad \dfrac{x{:}\tau \vdash x : \tau}{\vdash \lambda x.x : \tau \to \tau}}{\vdash \lambda x.x : (\sigma \to \sigma) \cap (\tau \to \tau)}$$

- Self-application $\lambda x.x\,x$

$$\dfrac{\dfrac{\dfrac{}{x{:}(\sigma \to \tau) \cap \sigma \vdash x : (\sigma \to \tau) \cap \sigma}}{x{:}(\sigma \to \tau) \cap \sigma \vdash x : \sigma \to \tau} \qquad \dfrac{\dfrac{}{x{:}(\sigma \to \tau) \cap \sigma \vdash x : (\sigma \to \tau) \cap \sigma}}{x{:}(\sigma \to \tau) \cap \sigma \vdash x : \sigma}}{\dfrac{x{:}(\sigma \to \tau) \cap \sigma \vdash x\,x : \tau}{\vdash \lambda x.x\,x : ((\sigma \to \tau) \cap \sigma) \to \tau}}$$

Examples (continued)

6 $\cap$ AS A CONNECTIVE

It was remarked in section 1 that $\cap$ behaves quite differently from &. This will now be made apparent.

A is a theorem iff, for some t, $\vdash t : A$ is derivable. This amounts to saying that A is a theorem iff A is realized by a closed member of TERM.

Given theorem 4.12 and 5.5, it is easy to show that the following formulas are not theorems: $p \rightarrow .\, q \rightarrow p \cap q$, $p \rightarrow q \rightarrow .\, p \rightarrow r \rightarrow .\, p \rightarrow q \cap r$, $p \cap q \rightarrow r \rightarrow .\, p \rightarrow .\, q \rightarrow r$. On the other hand, the following sequents are derivable.

$\vdash \lambda x.xx : A \cap (A \rightarrow B) \rightarrow B$

$\vdash \lambda x.\lambda y.xy : (A \rightarrow B) \cap (A \rightarrow C) \rightarrow .\, A \rightarrow B \cap C$

$\vdash \lambda x.\lambda y, xy : A \rightarrow B \cap C \rightarrow .\, (A \rightarrow B) \cap (A \rightarrow C)$

$\vdash \lambda x.\lambda y.xy : A \rightarrow C \rightarrow .\, A \cap B \rightarrow C$

$\vdash \lambda x.\lambda y.x : A \cap B \rightarrow .\, A \rightarrow B$

$\vdash \lambda x.\lambda y, xyy : A \rightarrow (B \rightarrow C) \rightarrow .\, A \cap B \rightarrow C$

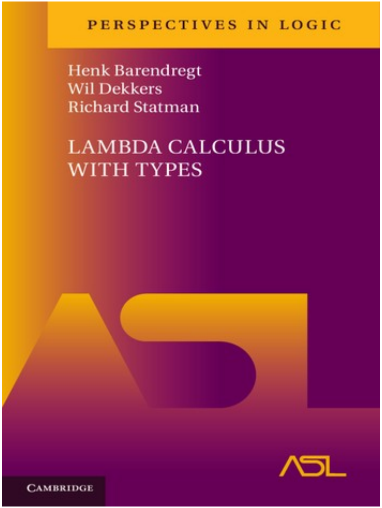
$\vdash \lambda x.x : A \cap B \rightarrow A$

$\vdash \lambda x.x : A \rightarrow A \cap A$

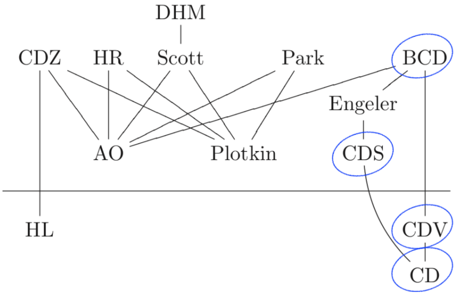
$\vdash \lambda x.x : A \cap B \rightarrow B \cap A$

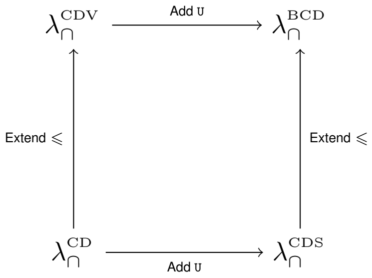
$\vdash \lambda x.x : A \cap (B \cap C) \rightarrow (A \cap B) \cap C$

The Reference Book: part 3 is dedicated to Intersection Types



Taxonomy of 13 $\bigcap$-systems, pp. 601 [BDS13]





4 historical systems	$\lambda_\cap^{\mathcal{T}}$	$\mathcal{T}$
Coppo-Dezani '78	$\lambda_\cap^{\mathrm{CD}}$	CD
Coppo-Dezani-Sallé '79	$\lambda_\cap^{\mathrm{CDS}}$	CDS
Coppo-Dezani-Venneri '81	$\lambda_\cap^{\mathrm{CDV}}$	CDV
Barendregt-Coppo-Dezani '83	$\lambda_\cap^{\mathrm{BCD}}$	BCD

Intersection Type Theories $\mathcal{T}$

Minimal type theory $\leqslant_{\text{min}}$

$(\text{refl}) \quad \sigma \leqslant \sigma$

$(\text{incl}) \quad \sigma \cap \tau \leqslant \sigma \quad \sigma \cap \tau \leqslant \tau$

$(\text{glb}) \quad \rho \leqslant \sigma \,\&\, \rho \leqslant \tau \Rightarrow \rho \leqslant \sigma \cap \tau$

$(\text{trans}) \quad \sigma \leqslant \tau \,\&\, \tau \leqslant \rho \Rightarrow \sigma \leqslant \rho$

Axiom schemes

$(\mathtt{U}_{top}) \quad \sigma \leqslant \mathtt{U}$ [Universal type]

$(\mathtt{U}_{\to}) \quad \mathtt{U} \leqslant \sigma \to \mathtt{U}$

$(\to\cap) \quad (\sigma \to \tau) \cap (\sigma \to \rho) \leqslant \sigma \to (\tau \cap \rho)$

Rule scheme

$(\to) \quad \sigma_2 \leqslant \sigma_1 \,\&\, \tau_1 \leqslant \tau_2 \Rightarrow \sigma_1 \to \tau_1 \leqslant \sigma_2 \to \tau_2$

4 *historical systems*	$\lambda_\cap^{\mathcal{T}}$	$\mathcal{T}$	$\leqslant_{\text{min}}$ plus	U?
Coppo-Dezani '78	$\lambda_\cap^{\text{CD}}$	CD	–	No
Coppo-Dezani-Sallé '79	$\lambda_\cap^{\text{CDS}}$	CDS	$(\mathtt{U}_{top})$	Yes
Coppo-Dezani-Venneri '81	$\lambda_\cap^{\text{CDV}}$	CDV	$(\to), (\to\cap)$	No
Barendregt-Coppo-Dezani '83	$\lambda_\cap^{\text{BCD}}$	BCD	$(\to), (\to\cap), (\mathtt{U}_{top}), (\mathtt{U}\to)$	Yes

Subtyping in programming languages 1/3

- Subtyping, denoted by $\leqslant$, is a form of implicit polymorphism (aka implicit type conversion or implicit type coercion)

Subtyping in programming languages 1/3

- Subtyping, denoted by $\leqslant$, is a form of implicit polymorphism (aka implicit type conversion or implicit type coercion)

- Subtyping allows us to implicitly and safely promote some variable of some type into another type

```
  int x = 3;              x is an integer
float y = 4.0;            y is a float
float z = x + y;          x is implicitly coerced into a float
                          // the result is 7.0
```

Subtyping in programming languages 1/3

- Subtyping, denoted by $\leqslant$, is a form of implicit polymorphism (aka implicit type conversion or implicit type coercion)

- Subtyping allows us to implicitly and safely promote some variable of some type into another type

```
  int x = 3;                          x is an integer
float y = 4.0;                        y is a float
float z = x + y;                      x is implicitly coerced into a float
                                      // the result is 7.0
```

- Subtyping is not an explicit type conversion (aka type casting)

```
float x = 3.0;                        x is an integer
float y = 4.0;                        y is an double
  int z = (int)x + (int)y;            x and y are casted into integers
                                      // the result is 7
```

Subtyping in programming languages 2/3

Subtyping hierarchy in C

long double
↑
double
↑
float
↑
unsigned long long
↑
long long
↑
unsigned long
↑
long
↑
unsigned int
↑
int

Subtyping in OO programming languages 3/3

- Subtyping lurks also in object-oriented programming

  "An object of class T may be substituted with any object of a subclass S" *(©Barbara Liskov)*

- Inheritance as subtyping

- Subtyping hierarchy in Java

```
Class Point {int x = 0; int y = 0}
Class ColPoint extends Point with {string col = red}

   Point p = new Point();
ColPoint q = new ColPoint()
```

`p = q`	accept
~~`q = p`~~	reject
`q = (ColPoint) p`	accept (explicit cast)

Parametric *vs. ad hoc* polymorphism (1/2)

- Parametric (ML)

```
> fun x -> x : 'a -> 'a
```

'a is a type variable

- Ad hoc (C)

```
  int a, b;
float x, y;
printf(``%d %f'', a+b, x+y);
```

- The type of the operator + is

$$+ : (\texttt{int -> int}) \cap (\texttt{float -> float})$$

- Girard's parametric polymorphism (System F) is "equivalent" to *ad hoc* polymorphism

$$\forall \alpha.\sigma \overset{\text{conj}}{=} \bigcap_{i=1\ldots\infty} \sigma_i$$

$$\forall \alpha.\alpha \to \alpha \sim (int \to int) \cap (nat \to nat) \cap (real \to real) \cap \ldots$$

Parametric *vs. ad hoc* (2/2)

Intersection types can type every strongly normalizing term of the λ-calculus... which is not the case in System F (or F_{omega}) ... this "monster" λ-term is strongly normalizing

$$\lambda x.z\,(x\,(\lambda f.\lambda u.f\,u))\,(x\,(\lambda v.\lambda g.g\,v))\,(\lambda y.y\,y\,y)$$

is not typable in Girard's F_{omega} but it is in Coppo-Dezani $\lambda_{\cap}^{\mathrm{CD}}$ (rank 3)

[Urzyczyn, MSCS'97]

3 Strongly normalizable but untypable

The aim of this section is to show our first main result: the type inference rules of $\mathbf{F}_\omega$ do not suffice to type all strongly normalizable terms. Our counter-example is the following term:

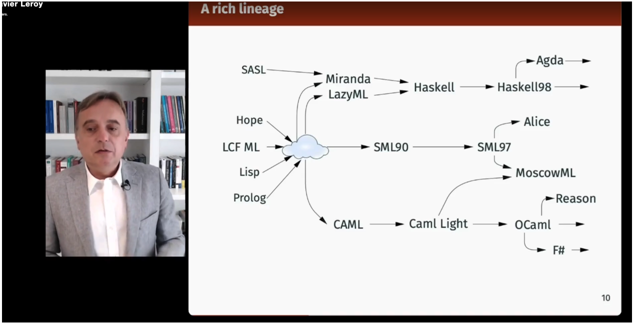
$$(*) \qquad M \equiv (\lambda x.\, z(x1)(x1'))(\lambda y.\, yyy)$$

where $1 \equiv \lambda fu.\, fu$ and $1' \equiv \lambda vg.\, gv$. Clearly, M is strongly normalizable, and it is an easy exercise to see that it becomes typable in $\mathbf{F}_\omega$ after just one reduction step.

Theorem 3.1 *The above term M cannot be typed in $\mathbf{F}_\omega$, and thus the class of typable terms is a proper subclass of the class of all strongly normalizable terms.*

Many attempts to adopt Intersection Types, Programming and Proof Languages

- Languages *à la* ML (Type Inference): Failure because of the HUGE literature on the difficulty to find a *Principal Type System*, see Damas-Milner seminal algorithm $\mathcal{W}$ [POPL82]



- Languages *à la* Algol, C, Java (Type Checking): Failure because of the HUGE literature to find expressive Fully Typed presentation with Decidable Type Checking. A small "galleria" follows...

Why a typed calculus with $\bigcap$ is so complicated?

- Intersection (and union types) were defined as type assignment systems (for pure λ-terms)
- Very elegant presentation but undecidability of type inference
- Many attempts of finding decidable and typed λ-calculi with intersection (and union types) preserving all the good properties of type assignment
- The usual approach (adding types to binders) is problematic

GOAL: find a suitable Intersection type systems *à la* Church

- with DECIDABILITY of Type Checking while preserving expressivity of the Type Assignment characterising ALL STRONGLY NORMALISING TERMS

$$\dfrac{\dfrac{\overline{x{:}\sigma \vdash_\cap x : \sigma}}{\vdash_\cap \lambda x.x : \sigma \to \sigma} \qquad \dfrac{\overline{x{:}\tau \vdash_\cap x : \tau}}{\vdash_\cap \lambda x.x : \tau \to \tau}}{\vdash_\cap \lambda x.x : (\sigma \to \sigma) \cap (\tau \to \tau)}\ (\cap I)$$

$$\dfrac{\dfrac{\overline{x{:}\sigma \vdash x : \sigma}}{\vdash \lambda x{:}\sigma.x : \sigma \to \sigma} \qquad \dfrac{\overline{x{:}\tau \vdash x : \tau}}{\vdash \lambda x{:}\tau.x : \tau \to \tau}}{\vdash \lambda x{:}???.x : (\sigma \to \sigma) \cap (\tau \to \tau)}\ (\cap I)$$

Reynolds' FORSYTHE '88

Reynolds annotates a λ-abstraction with types as in

$$\frac{B, x{:}\sigma_i \vdash M : \tau \quad i \in 1 \ldots n}{B \vdash \lambda x{:}\sigma_1|\cdots|\sigma_n.M : \sigma_i \to \tau}$$

However, we cannot type a typed term, whose type erasure is the combinator

$$\mathsf{K} \equiv \lambda x.\lambda y.x$$

with the intersection type

$$(\sigma \to \sigma \to \sigma) \cap (\tau \to \tau \to \tau)$$

Pierce's PhD '91

Pierce improves Forsythe by using a `for` construct to build *ad hoc* polymorphic typing, as in

$$\frac{B \vdash M[\sigma_i/\alpha] : \tau_i \quad i \in 1 \ldots n}{B \vdash \texttt{for}\, \alpha \in \{\sigma_1 \ldots \sigma_n\}.M : \tau_i}$$

However, we cannot type a typed term, whose type erasure is

$$\lambda x.\lambda y.\lambda z.(x\, y\, , x\, z)$$

with the intersection type

$$((\sigma \to \rho) \cap (\tau \to \rho') \to \sigma \to \tau \to \rho \times \rho')$$
$$\cap$$
$$((\sigma \to \sigma) \cap (\sigma \to \sigma) \to \sigma \to \sigma \to \sigma \times \sigma)$$

Pfenning&Friedman: Refinement Types '91

Refinement types are subtypes of standard types

We can only intersect types which are refinements of the same ML type

Subtype *ground* refine the ML type *boolexp*: variables cannot be of type *ground*

$$
\begin{array}{rcl}
\mathit{ground} & \sqsubseteq & \mathit{boolexp} \\
\mathit{Var} & : & \mathit{boolexp} \\
\mathit{True}, \mathit{False} & : & \mathit{ground} \cap \mathit{boolexp} \\
\mathit{Not} & : & \mathit{ground} \cap \mathit{boolexp} \to \mathit{ground} \cap \mathit{boolexp} \\
\mathit{And} & : & (\mathit{boolexp} * \mathit{boolexp} \to \mathit{boolexp}) \\
 & & \cap \\
 & & (\mathit{ground} * \mathit{ground} \to \mathit{ground})
\end{array}
$$

Miquel's Implicit Constructions PhD '01

Extends Coquand-Huet's CC with the ternary operator

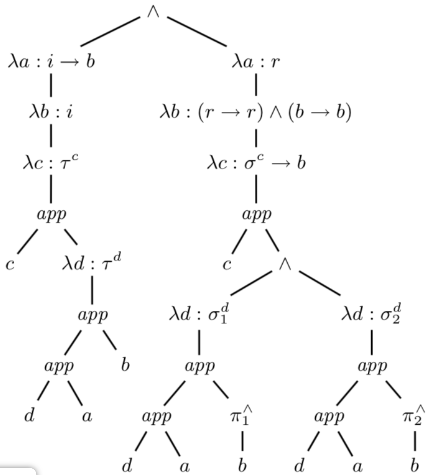
$$b?M\,;N \text{ of type } \Pi b{:}bool.\sigma\,;\tau$$

$$true?M\,;N \longrightarrow_{IC} M \qquad false?M\,;N \longrightarrow_{IC} N$$

Unfortunately, not all terms typed by intersection types have an equivalent in ICC, for instance $\lambda x.x : ((\sigma \cap \tau) \to \sigma) \cap (\rho \to \rho))$ appears to be problematic

Wells *et al.* $\vdash_{\cap}^{\mathcal{T}_{\mathrm{CD}}} \lambda a.\lambda b.\lambda c.c\,(\lambda d.d\,a\,b) : \tau \cap \sigma$

JFP '02. Explicitly Typed Intermediate Languages (TILs) facilitate the safe and efficient compilation of programming languages

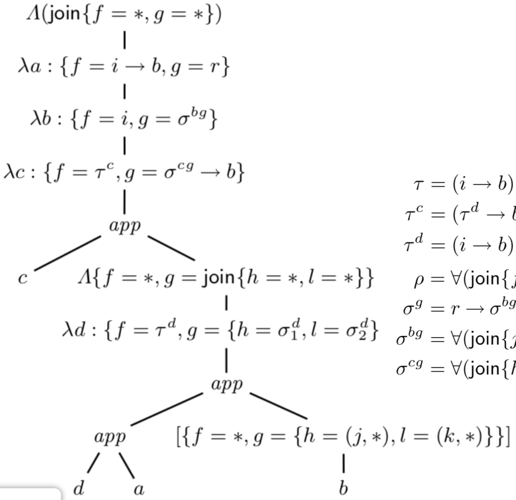


$$
\begin{aligned}
\tau &= (i \to b) \to i \to \tau^c \to b \\
\tau^c &= (\tau^d \to b) \to b \\
\tau^d &= (i \to b) \to i \to b
\end{aligned}
$$

$$
\begin{aligned}
\sigma &= r \to ((r \to r) \wedge (b \to b)) \to (\sigma^c \to b) \to b \\
\sigma^c &= ((\sigma_1^d \to b) \wedge (\sigma_2^d \to b)) \\
\sigma_1^d &= r \to (r \to r) \to b \\
\sigma_2^d &= r \to (b \to b) \to b
\end{aligned}
$$

Wells & Haak $\vdash^{\mathcal{T}\mathrm{CD}}_{\cap} \lambda a.\lambda b.\lambda c.c\,(\lambda d.d\,a\,b) : \tau \cap \sigma$

ESOP '02. The first system with the power of $\lambda^{\mathrm{CD}}_{\cap}$



$$
\begin{aligned}
\tau &= (i \to b) \to i \to \tau^c \to b \\
\tau^c &= (\tau^d \to b) \to b \\
\tau^d &= (i \to b) \to i \to b \\
\rho &= \forall(\mathsf{join}\{f = *, g = *\}).\,\{f = \tau, g = \sigma^g\} \\
\sigma^g &= r \to \sigma^{bg} \to (\sigma^{cg} \to b) \to b \\
\sigma^{bg} &= \forall(\mathsf{join}\{j = *, k = *\}).\,\{j = r \to r, k = b \to b\} \\
\sigma^{cg} &= \forall(\mathsf{join}\{h = *, l = *\}).\,\{h = \sigma_1^d \to b, l = \sigma_2^d \to b\}
\end{aligned}
$$

Frisch, Castagna, Benzaken, JSL '08

- Types as sets and subtyping as subsets
- $\sigma \cap \tau \leqslant \sigma$ is interpreted as $[\![\sigma]\!] \cap [\![\tau]\!] \subseteq [\![\sigma]\!]$
- $([\![\sigma]\!] \cap [\![\tau]\!]) \cap \overline{[\![\sigma]\!]} = \emptyset$

$$\frac{t = \bigwedge_{i=1..n} (t_i \to s_i) \wedge \boxed{\bigwedge_{j=1..m} \neg(t'_j \to s'_j)} \quad \boxed{t \not\simeq \mathbb{0}} \qquad \forall i = 1..n. \Gamma, (f : t), (x : t_i) \vdash e : s_i}{\Gamma \vdash \mu f(t_1 \to s_1; \ldots; t_n \to s_n).\lambda x.e : t} \; (abstr)$$

Many attempts to find an INTUITIONISTIC LOGIC corresponding to Intersection

A “scientific consensus” on the existence of a

Curry-Howard Isomorphism for $\bigcap$

is still an OPEN PROBLEM

Curry-Howard Isomorphism

Types as

Logical Propositions

(Formulas)

and

Typed λ-terms as

Logical Proofs

Logic vs Intersection (and Union) Types

$\cap$ is not $\wedge$

The dual type of intersection is Union:

$\cup$ is not $\vee$

Since the meaning of $\cap$ is reasonably clear (to claim that $A \cap B$ is to claim that one has a reason for asserting A which is also a reason for asserting B), it would obviously be of interest to figure out how to add $\cap$ to intuitionist logic and then consider the analysis of intuitionist mathematical reasoning in the light of the resulting system.

NB: Usual *intuitionistic logics* do not apply for intersection and union

Proof-functional logics for INTERSECTION

Pottinger '80 conjectured a "logical" interpretation of intersection as an *intuitionistic connective*, stating that:

CONJUNCTION: "To assert $A \wedge B$ is to assert that one has a pair of reasons, the first of which is a reason for asserting A and the second (<u>[possibly different from the first]</u>) of which is a reason for asserting B"

... (while) ...

INTERSECTION: "To assert $A \cap B$ is to assert that one has a reason for asserting A <u>which is also</u> a reason for asserting B"

$$\frac{\begin{array}{cc}\mathcal{P}_1 & \mathcal{P}_2 \\ \Downarrow & \Downarrow \\ A & B\end{array}}{A \wedge B} \qquad\qquad \frac{\begin{array}{cc}\mathcal{P} & \mathcal{P} \\ \Downarrow & \Downarrow \\ A & B\end{array}}{A \cap B}$$

Many attempts to find a LOGIC corresponding to Intersection

As the time the existence of a

Curry-Howard Isomorphism

is still an OPEN PROBLEM

Curry-Howard Isomorphism

Types as

Logical Propositions

(Formulas)

and

Typed λ-terms as

Logical Proofs

Logic vs Intersection (and Union) Types

$$\cap \text{ is not } \wedge$$

The dual type of intersection is Union:

$$\cup \text{ is not } \vee$$

Since the meaning of $\cap$ is reasonably clear (to claim that $A \cap B$ is to claim that one has a reason for asserting A which is also a reason for asserting B), it would obviously be of interest to figure out how to add $\cap$ to intuitionist logic and then consider the analysis of intuitionist mathematical reasoning in the light of the resulting system.

NB: Usual *intuitionistic logics* do not apply for intersection and union

Proof-functional logics for INTERSECTION

Pottinger '80 conjectured a "logical" interpretation of intersection as an *intuitionistic connective*, stating that:

CONJUNCTION: "To assert $A \wedge B$ is to assert that one has a pair of reasons, the first of which is a reason for asserting A and the second ([possibly different from the first]) of which is a reason for asserting B"

... (while) ...

INTERSECTION: "To assert $A \cap B$ is to assert that one has a reason for asserting A which is also a reason for asserting B"

$$\frac{\begin{array}{c}\mathcal{P}_1\\ \curlyvee\\ A\end{array} \qquad \begin{array}{c}\mathcal{P}_2\\ \curlyvee\\ B\end{array}}{A \wedge B} \qquad\qquad \frac{\begin{array}{c}\mathcal{P}\\ \curlyvee\\ A\end{array} \qquad \begin{array}{c}\mathcal{P}\\ \curlyvee\\ B\end{array}}{A \cap B}$$

A proposal of a Church-style calculus with Intersection Types

The Δ-calculus: Syntax and Types

Luigi Liquori **Claude Stolze** [1]
Université Côte d'Azur, Inria, France
[Luigi.Liquori,Claude.Stolze]@inria.fr

Syntax of the Generic Δ-calculus

σ	$::=$	$\phi \mid \sigma \to \sigma \mid \sigma \cap \sigma \mid \mathtt{U}$	types
Δ	$::=$	$x \mid \lambda x{:}\sigma.\Delta \mid \Delta\,\Delta \mid$	typed λ-calculus
		$\langle \Delta\,,\Delta \rangle \mid$	strong pairs
		$pr_1\,\Delta \mid pr_2\,\Delta \mid$	projections
		$\Delta^{\sigma} \mid$	explicit coercions
		u_Δ	indexed constants

A strong pair $\langle \Delta\,,\Delta \rangle$ is *a kind of* cartesian pair

An explicit coercion is Δ^{σ} is a Δ-term annotated with a type

u_Δ is an infinite set of constants indexed by Δ-terms

Typed Δ *vs.* Untyped λ: the essence function

- Essence is an *erasing function* transforming a typed Δ-term into an untyped λ-term.

$$
\begin{array}{rcl}
\wr\, x \,\wr & \stackrel{def}{=} & x \\
\wr\, \lambda x{:}\sigma.\Delta \,\wr & \stackrel{def}{=} & \lambda x.\wr\, \Delta \,\wr \\
\wr\, \Delta_1\ \Delta_2 \,\wr & \stackrel{def}{=} & \wr\, \Delta_1 \,\wr\wr\, \Delta_2 \,\wr \\
\wr\, \Delta^{\sigma} \,\wr & \stackrel{def}{=} & \wr\, \Delta \,\wr \\
\wr\, u_{\Delta} \,\wr & \stackrel{def}{=} & \wr\, \Delta \,\wr \\
\wr\, pr_i\, \Delta \,\wr & \stackrel{def}{=} & \wr\, \Delta \,\wr \\
\wr\, \langle \Delta_1 \,, \Delta_2 \rangle \,\wr & \stackrel{def}{=} & \wr\, \Delta_1 \,\wr
\end{array}
$$

Reduction semantics

- **(Substitution)** Substitution on Δ-terms is defined as usual, with the additional rules:

$$u_{\Delta_1}[\Delta_2/x] \stackrel{def}{=} u_{(\Delta_1[\Delta_2/x])}$$

$$\Delta_1^{\sigma}[\Delta_2/x] \stackrel{def}{=} (\Delta_1[\Delta_2/x])^{\sigma}$$

- **(One-step reduction)** Reduction rules:

$$\begin{array}{rcll} (\lambda x{:}\sigma.\Delta_1)\,\Delta_2 & \longrightarrow_{\beta} & \Delta_1[\Delta_2/x] & \\ pr_i\,\langle \Delta_1\,,\Delta_2\rangle & \longrightarrow_{pr_i} & \Delta_i & \text{for } i \in \{1,2\} \\ \lambda x{:}\sigma.\Delta\,x & \longrightarrow_{\eta} & \Delta & \text{if } x \notin FV(\Delta) \end{array}$$

2 NB and 1 EX

- NB.1: $(\lambda x{:}\sigma.\Delta_1)^{\tau}\ \Delta_2$ is not a redex

- NB.2: $u_{(\lambda x:\sigma.\Delta_1)\ \Delta_2}$ is not a redex

- EX: $(\lambda x{:}\sigma\to\sigma.u_{(x\ x)})\ (\lambda x{:}\sigma.x) \longrightarrow_{\beta} u_{((\lambda x:\sigma.x)\ (\lambda x:\sigma.x))}$

Synchronization in reductions

Desynchronization inside a strong pair can produce “exotic” Δ-terms

$$\langle(\lambda x{:}\sigma.x)\,pr_1\,y\,,(\lambda x{:}\tau.x)\,pr_2\,y\rangle \begin{array}{l} \nearrow^{\beta}\ \langle(\lambda x{:}\sigma.x)\,pr_1\,y\,,pr_2\,y\rangle \searrow_{\beta} \\ \searrow_{\beta}\ \langle pr_1\,y\,,(\lambda x{:}\sigma.x)\,pr_2\,y\rangle \nearrow^{\beta} \end{array} \langle pr_1\,y\,,pr_2\,y\rangle$$

Synchronicity in operational semantics

$$\frac{\Delta_1 \longrightarrow^{\parallel} \Delta_1' \quad \Delta_2 \longrightarrow^{\parallel} \Delta_2' \quad \wr\Delta_1'\wr \equiv \wr\Delta_2'\wr}{\langle\Delta_1\,,\Delta_2\rangle \longrightarrow^{\parallel} \langle\Delta_1'\,,\Delta_2'\rangle}\ (\mathit{Sync})$$

Church-Rosser property

- Reduction is confluent

$$\begin{tikzcd}[column sep=small]
& \Delta_1 \arrow[dl, twoheadrightarrow] \arrow[dr, twoheadrightarrow] & \\
\Delta_2 \arrow[dr, dashed, twoheadrightarrow] & & \Delta_3 \arrow[dl, dashed, twoheadrightarrow] \\
& \Delta_4 &
\end{tikzcd}$$

- Synchronous reduction is also confluent

$$\begin{tikzcd}[column sep=small]
& \Delta_1 \arrow[dl, twoheadrightarrow, "\|"'] \arrow[dr, twoheadrightarrow, "\|"] & \\
\Delta_2 \arrow[dr, dashed, twoheadrightarrow, "\|"] & & \Delta_3 \arrow[dl, dashed, twoheadrightarrow, "\|"'] \\
& \Delta_4 &
\end{tikzcd}$$

The Generic Δ-calculus type system $\vdash^{\mathcal{T}}_{\mathcal{R}}$

The Type Checker depends on 2 parameters:

1. The subtyping relation $\leqslant_{\mathcal{T}}$ in $\mathcal{T} \in \{\mathrm{CD}, \mathrm{CDV}, \mathrm{CDS}, \mathrm{BCD}\}$
2. The synchronicity relation $\mathcal{R}$ on pure λ-terms, $\mathcal{R} \in \{\equiv, =_\beta, =_{\beta\eta}\}$

$$\frac{x{:}\sigma \in \Gamma}{\Gamma \vdash^{\mathcal{T}}_{\mathcal{R}} x : \sigma}\ (Var)$$

$$\frac{\Gamma \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma}{\Gamma \vdash^{\mathcal{T}}_{\mathcal{R}} u_\Delta : \mathtt{U}}\ (\mathtt{U}) \qquad \frac{\Gamma \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma \quad \sigma \leqslant_{\mathcal{T}} \tau}{\Gamma \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta^\tau : \tau}\ (\leqslant)$$

$$\frac{\begin{array}{l}\Gamma \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta_1 : \sigma \\ \Gamma \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta_2 : \tau \quad \wr \Delta_1 \wr \ \mathcal{R}\ \wr \Delta_2 \wr\end{array}}{\Gamma \vdash^{\mathcal{T}}_{\mathcal{R}} \langle \Delta_1 , \Delta_2 \rangle : \sigma \cap \tau}\ (\cap I) \qquad \frac{\Gamma \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma_1 \cap \sigma_2 \quad i \in \{1,2\}}{\Gamma \vdash^{\mathcal{T}}_{\mathcal{R}} pr_i\, \Delta : \sigma_i}\ (\cap E_i)$$

$$\frac{\Gamma \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta_1 : \sigma \to \tau \quad \Gamma \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta_2 : \sigma}{\Gamma \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta_1\, \Delta_2 : \tau}\ (\to E) \qquad \frac{\Gamma, x{:}\sigma \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \tau}{\Gamma \vdash^{\mathcal{T}}_{\mathcal{R}} \lambda x{:}\sigma.\Delta : \sigma \to \tau}\ (\to I)$$

Examples

Auto-application $(\lambda x.x\,x)$ à la Curry can be typed à la Church as follows:

$$\dfrac{\dfrac{x{:}(\sigma \to \tau) \cap \sigma \vdash x{:}(\sigma \to \tau) \cap \sigma}{x{:}(\sigma \to \tau) \cap \sigma \vdash pr_1\, x : \sigma \to \tau} \qquad \dfrac{x{:}(\sigma \to \tau) \cap \sigma \vdash x{:}(\sigma \to \tau) \cap \sigma}{x{:}(\sigma \to \tau) \cap \sigma \vdash pr_2\, x : \sigma}}{\dfrac{x{:}(\sigma \to \tau) \cap \sigma \vdash (pr_1\, x)\,(pr_2\, x) : \tau}{\vdash \lambda x{:}(\sigma \to \tau) \cap \sigma.(pr_1\, x)\,(pr_2\, x) : ((\sigma \to \tau) \cap \sigma) \to \tau}}$$

Examples

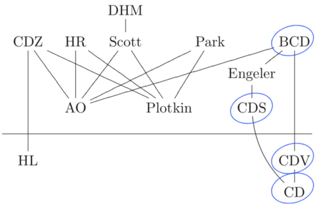
For $\Delta^{\mathrm{CDS}}_{\mathcal{R}}$ and $\Delta^{\mathrm{BCD}}_{\mathcal{R}}$:

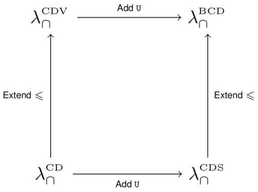
Fixpoint combinator $Y \stackrel{def}{=} \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))$

$$\dfrac{\dfrac{\dfrac{f{:}\mathtt{U} \to \sigma, x{:}\mathtt{U} \vdash f : \mathtt{U} \to \sigma \quad f{:}\mathtt{U} \to \sigma, x{:}\mathtt{U} \vdash u_{(x\,x)} : \mathtt{U}}{f{:}\mathtt{U} \to \sigma, x{:}\mathtt{U} \vdash f\,u_{(x\,x)} : \sigma}}{f{:}\mathtt{U} \to \sigma \vdash \lambda x{:}\mathtt{U}.f\,u_{(x\,x)} : \mathtt{U} \to \sigma} \quad f{:}\mathtt{U} \to \sigma \vdash u_{(\lambda x:\mathtt{U}.f\,u_{(x\,x)})} : \mathtt{U}}{\dfrac{f{:}\mathtt{U} \to \sigma \vdash (\lambda x{:}\mathtt{U}.f\,u_{(x\,x)})\,u_{(\lambda x:\mathtt{U}.f\,u_{(x\,x)})} : \sigma}{\vdash \lambda f{:}\mathtt{U} \to \sigma.(\lambda x{:}\mathtt{U}.f\,u_{(x\,x)})\,u_{(\lambda x:\mathtt{U}.f\,u_{(x\,x)})} : (\mathtt{U} \to \sigma) \to \sigma}}$$

Replay

Taxonomy of 13 $\bigcap$-systems, pp. 601 [BDS13]





4 *historical systems*	$\lambda_\cap^{\mathcal{T}}$	$\mathcal{T}$	$\leqslant_{\text{min}}$ plus	U?
Coppo-Dezani '78	$\lambda_\cap^{CD}$	CD	–	No
Coppo-Dezani-Sallé '79	$\lambda_\cap^{CDS}$	CDS	$(\mathtt{U}_{top})$	Yes
Coppo-Dezani-Venneri '81	$\lambda_\cap^{CDV}$	CDV	$(\to), (\to\cap)$	No
Barendregt-Coppo-Dezani '83	$\lambda_\cap^{BCD}$	BCD	$(\to), (\to\cap), (\mathtt{U}_{top}), (\mathtt{U}\to)$	Yes

The Δ-chair

$$\begin{tikzcd}
\lambda^{\mathrm{CDV}}_{\cap} \arrow[r, "\text{Add } \cup"] & \lambda^{\mathrm{BCD}}_{\cap} \\
\lambda^{\mathrm{CD}}_{\cap} \arrow[u, "\text{Extend } \leqslant"] \arrow[r, "\text{Add } \cup"'] & \lambda^{\mathrm{CDS}}_{\cap} \arrow[u, "\text{Extend } \leqslant"']
\end{tikzcd}$$

$$\begin{tikzcd}
 & \Delta^{\mathrm{CDV}}_{=_{\beta\eta}} \arrow[rr] & & \Delta^{\mathrm{BCD}}_{=_{\beta\eta}} \\
 & \Delta^{\mathrm{CDV}}_{=_{\beta}} \arrow[u] \arrow[rr] & & \Delta^{\mathrm{BCD}}_{=_{\beta}} \arrow[u] \\
\Delta^{\mathrm{CD}}_{=_{\beta}} \arrow[ur] \arrow[rr] & & \Delta^{\mathrm{CDS}}_{=_{\beta}} \arrow[ur] & \\
 & \Delta^{\mathrm{CDV}}_{\equiv} \arrow[uu] \arrow[rr] & & \Delta^{\mathrm{BCD}}_{\equiv} \arrow[uu] \\
\Delta^{\mathrm{CD}}_{\equiv} \arrow[uu] \arrow[ur] \arrow[rr] & & \Delta^{\mathrm{CDS}}_{\equiv} \arrow[uu] \arrow[ur] &
\end{tikzcd}$$

$$\mathcal{R} \in \{\equiv, =_{\beta}, =_{\beta\eta}\}$$

Isomorphism property of $\Delta^{\mathcal{T}}_{\mathcal{R}}$

(Soundness)	$(\Delta^{\mathcal{T}}_{\mathcal{R}} \triangleright \lambda^{\mathcal{T}}_{\cap})$	$\Gamma \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma \Longrightarrow \Gamma \vdash^{\mathcal{T}}_{\cap} \wr \Delta \wr : \sigma$
(Completeness)	$(\Delta^{\mathcal{T}}_{\mathcal{R}} \triangleleft \lambda^{\mathcal{T}}_{\cap})$	$\Gamma \vdash^{\mathcal{T}}_{\cap} M : \sigma \Longrightarrow \exists \Delta. \wr \Delta \wr \equiv M$ and $\Gamma \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma$
(Isomorphism)	$(\Delta^{\mathcal{T}}_{\mathcal{R}} \sim \lambda^{\mathcal{T}}_{\cap})$	$\Delta^{\mathcal{T}}_{\mathcal{R}} \triangleright \lambda^{\mathcal{T}}_{\cap}$ and $\Delta^{\mathcal{T}}_{\mathcal{R}} \triangleleft \lambda^{\mathcal{T}}_{\cap}$

Isomorphism property of $\Delta^{\mathcal{T}}_{\mathcal{R}}$

(Soundness) $(\Delta^{\mathcal{T}}_{\mathcal{R}} \triangleright \lambda^{\mathcal{T}}_{\cap})$ $\quad \Gamma \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma \Longrightarrow \Gamma \vdash^{\mathcal{T}}_{\cap} \wr \Delta \wr : \sigma$

(Completeness) $(\Delta^{\mathcal{T}}_{\mathcal{R}} \triangleleft \lambda^{\mathcal{T}}_{\cap})$ $\quad \Gamma \vdash^{\mathcal{T}}_{\cap} M : \sigma \Longrightarrow \exists \Delta. \wr \Delta \wr \equiv M$ and $\Gamma \vdash^{\mathcal{T}}_{\mathcal{R}} \Delta : \sigma$

(Isomorphism) $(\Delta^{\mathcal{T}}_{\mathcal{R}} \sim \lambda^{\mathcal{T}}_{\cap})$ $\quad \Delta^{\mathcal{T}}_{\mathcal{R}} \triangleright \lambda^{\mathcal{T}}_{\cap}$ and $\Delta^{\mathcal{T}}_{\mathcal{R}} \triangleleft \lambda^{\mathcal{T}}_{\cap}$

$$\begin{tikzcd}
 & \Delta^{\mathrm{CDV}}_{=_{\beta\eta}} \arrow[rr] & & \Delta^{\mathrm{BCD}}_{=_{\beta\eta}} \\
 & \Delta^{\mathrm{CDV}}_{=_{\beta}} \arrow[u] \arrow[rr] & & \Delta^{\mathrm{BCD}}_{=_{\beta}} \arrow[u] \\
\Delta^{\mathrm{CD}}_{=_{\beta}} \arrow[ur] \arrow[rr] & & \Delta^{\mathrm{CDS}}_{=_{\beta}} \arrow[ur] & \\
 & \Delta^{\mathrm{CDV}}_{\equiv} \arrow[uu] \arrow[rr] & & \Delta^{\mathrm{BCD}}_{\equiv} \arrow[uu] \\
\Delta^{\mathrm{CD}}_{\equiv} \arrow[uu] \arrow[ur] \arrow[rr] & & \Delta^{\mathrm{CDS}}_{\equiv} \arrow[uu] \arrow[ur] &
\end{tikzcd}$$

$\Delta^{\mathcal{T}}_{\mathcal{R}}$	$\Delta^{\mathcal{T}}_{\mathcal{R}} \triangleright \lambda^{\mathcal{T}}_{\cap}$	$\Delta^{\mathcal{T}}_{\mathcal{R}} \triangleleft \lambda^{\mathcal{T}}_{\cap}$
$\Delta^{\mathrm{CD}}_{\equiv}$	$\surd$	$\surd$
$\Delta^{\mathrm{CDV}}_{\equiv}$	$\surd$	$\surd$
$\Delta^{\mathrm{CDS}}_{\equiv}$	$\surd$	$\surd$
$\Delta^{\mathrm{BCD}}_{\equiv}$	$\surd$	$\surd$
$\Delta^{\mathrm{CD}}_{=_{\beta}}$	$\times$	$\surd$
$\Delta^{\mathrm{CDV}}_{=_{\beta}}$	$\times$	$\surd$
$\Delta^{\mathrm{CDS}}_{=_{\beta}}$	$\surd$	$\surd$
$\Delta^{\mathrm{BCD}}_{=_{\beta}}$	$\surd$	$\surd$
$\Delta^{\mathrm{CDV}}_{=_{\beta\eta}}$	$\times$	$\surd$
$\Delta^{\mathrm{BCD}}_{=_{\beta\eta}}$	$\times$	$\surd$

Counter-example for $\Delta^{\mathrm{CDV}}_{=_{\beta\eta}}$ and $\Delta^{\mathrm{BCD}}_{=_{\beta\eta}}$

- Let in $\Delta^{\mathrm{CDV/BCD}}_{=_{\beta\eta}}$

$$pr_2\,\langle \lambda y{:}\sigma.(pr_1\,x)\,y\,,pr_2\,x\rangle$$

- We have that

$$x{:}(\sigma\to\tau)\cap\rho\vdash^{CDV/BCD}_{=_{\beta\eta}} pr_2\,\langle \lambda y{:}\sigma.(pr_1\,x)\,y\,,pr_2\,x\rangle : \rho$$

- The essence is

$$\lambda y.x\,y$$

- ...but in $\lambda^{\mathrm{CDV}}_{\cap}, \lambda^{\mathrm{BCD}}_{\cap}$

$$x{:}(\sigma\to\tau)\cap\rho\nvdash^{CDV/BCD}_{\cap} \lambda y.x\,y : \rho$$

Counter-example for $\Delta^{\mathrm{CD}}_{=_\beta}$ and $\Delta^{\mathrm{CDV}}_{=_\beta}$

- $\mathsf{S} \stackrel{\mathit{def}}{=} \lambda x.\lambda y.\lambda z.x\, z\, (y\, z)$
- $\mathsf{K} \stackrel{\mathit{def}}{=} \lambda x.\lambda y.x$
- $\mathsf{S\,K\,S} =_\beta \lambda x.x$
- In $\Delta^{\mathrm{CD/CDV}}_{=_\beta}$, you can construct a term Δ such that

$$\wr \Delta \wr \equiv \mathsf{S\,K\,S}$$

- We have that

$$\vdash^{\mathit{CD/CDV}}_{=_\beta} \mathit{pr}_2 \, \langle \Delta \, , \lambda x{:}\sigma.x \rangle : \sigma \to \sigma$$

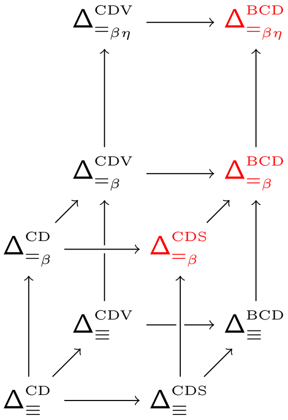
- the essence is

$$\mathsf{S\,K\,S}$$

- ...but in $\lambda^{\mathrm{CD}}_{\cap}, \lambda^{\mathrm{CDV}}_{\cap}$

$$\nvdash^{\mathit{CD/CDV}}_{\cap} \mathsf{S\,K\,S} : \sigma \to \sigma$$

Decidability of type checking/reconstruction



$\Delta^{\mathcal{T}}_{\mathcal{R}}$	TC/TR
$\Delta^{\mathrm{CD}}_{\equiv}$	$\surd$
$\Delta^{\mathrm{CDV}}_{\equiv}$	$\surd$
$\Delta^{\mathrm{CDS}}_{\equiv}$	$\surd$
$\Delta^{\mathrm{BCD}}_{\equiv}$	$\surd$
$\Delta^{\mathrm{CD}}_{=\beta}$	$\surd$
$\Delta^{\mathrm{CDV}}_{=\beta}$	$\surd$
$\Delta^{\mathrm{CDS}}_{=\beta}$	$\times$
$\Delta^{\mathrm{BCD}}_{=\beta}$	$\times$
$\Delta^{\mathrm{CDV}}_{=\beta\eta}$	$\surd$
$\Delta^{\mathrm{BCD}}_{=\beta\eta}$	$\times$

Why? $\langle u_{\Delta_1}\ , u_{\Delta_2}\rangle$ is typable if and only if $\wr\,\Delta_1\,\wr =_{\beta,\beta\eta} \wr\,\Delta_2\,\wr$

UNION TYPES

- Invented by Mc Queen, Plotkin and Sethi *et al.* in ’86
- Dual to Intersection Types
- Same features and drawbacks of Intersection Types
- Conjectures with its relation with Intuitionistic Logic
- Not clear Curry-Howard isomorphism
- Relation with Pottinger’s *Proof Functional Logic*

Type Assignment Rules for Union and Intersection all together

$$\frac{x{:}\sigma \in B}{B \vdash x : \sigma}\ (Var) \qquad \frac{B, x{:}\sigma \vdash M : \tau}{B \vdash \lambda x.M : \sigma \to \tau}\ ({\to}I) \qquad \frac{B \vdash M : \sigma \to \tau \quad B \vdash N : \sigma}{B \vdash M\,N : \tau}\ ({\to}E)$$

$$\frac{B \vdash M : \sigma \quad \sigma \leqslant \tau}{B \vdash M : \tau}\ (\leqslant) \qquad \frac{B \vdash M : \sigma \quad B \vdash M : \tau}{B \vdash M : \sigma \cap \tau}\ (\cap I)$$

$$\frac{B \vdash M : \sigma \cap \tau}{B \vdash M : \sigma}\ (\cap E_l) \qquad \frac{B \vdash M : \sigma \cap \tau}{B \vdash M : \tau}\ (\cap E_r)$$

$$\frac{B \vdash M : \sigma}{B \vdash M : \sigma \cup \tau}\ (\cup I_l) \qquad \frac{B \vdash M : \tau}{B \vdash M : \sigma \cup \tau}\ (\cup I_r)$$

$$\frac{B, x{:}\sigma \vdash M : \rho \quad B, x{:}\tau \vdash M : \rho \quad B \vdash N : \sigma \cup \tau}{B \vdash M[N/x] : \rho}\ (\cup E)$$

Subtyping rules (Ξ subtype theory in [BDdL])

(1) $\sigma \leqslant \sigma \cap \sigma$

(2) $\sigma \cup \sigma \leqslant \sigma$

(3) $\sigma \cap \tau \leqslant \sigma, \sigma \cap \tau \leqslant \tau$

(4) $\sigma \leqslant \sigma \cup \tau, \tau \leqslant \sigma \cup \tau$

(5) $\sigma \leqslant \mathtt{U}$

(6) $\sigma \leqslant \sigma$

(7) $\sigma_1 \leqslant \sigma_2, \tau_1 \leqslant \tau_2 \Rightarrow$
$\sigma_1 \cap \tau_1 \leqslant \sigma_2 \cap \tau_2$

(8) $\sigma_1 \leqslant \sigma_2, \tau_1 \leqslant \tau_2 \Rightarrow \sigma_1 \cup \tau_1 \leqslant \sigma_2 \cup \tau_2$

(9) $\sigma \leqslant \tau, \tau \leqslant \rho \Rightarrow \sigma \leqslant \rho$

(10) $\sigma \cap (\tau \cup \rho) \leqslant (\sigma \cap \tau) \cup (\sigma \cap \rho)$

(11) $(\sigma \rightarrow \tau) \cap (\sigma \rightarrow \rho) \leqslant \sigma \rightarrow (\tau \cap \rho)$

(12) $(\sigma \rightarrow \rho) \cap (\tau \rightarrow \rho) \leqslant (\sigma \cup \tau) \rightarrow \rho$

(13) $\mathtt{U} \leqslant \mathtt{U} \rightarrow \mathtt{U}$

(14) $\sigma_2 \leqslant \sigma_1, \tau_1 \leqslant \tau_2 \Rightarrow$
$\sigma_1 \rightarrow \tau_1 \leqslant \sigma_2 \rightarrow \tau_2$

Subtyping rules 1/4

A subtyping relation is a preorder, ie. a reflexive and transitive order with $\mathtt{U}$ is a universal type, corresponding to the $\top$ constant in the lattice of types (with $\cup$ as $\sqcup$ and $\cap$ as $\sqcap$)

$$\sigma \leqslant \sigma$$ Reflexivity

$$\sigma \leqslant \tau, \tau \leqslant \rho \Rightarrow \sigma \leqslant \rho$$ Transitivity

$$\sigma \leqslant \mathrm{U}$$ Universal type

$$\mathrm{U} \leqslant \mathrm{U} \to \mathrm{U}$$ Universal type is also a function

Subtyping rules 2/4

Main rules for intersection:

$$\sigma \leqslant \sigma \cap \sigma$$

$$\sigma \cap \tau \leqslant \sigma$$

$$\sigma \cap \tau \leqslant \tau$$

$$\sigma_1 \leqslant \sigma_2, \tau_1 \leqslant \tau_2 \Rightarrow \sigma_1 \cap \tau_1 \leqslant \sigma_2 \cap \tau_2$$ Inter. compositionality

Main rules for union:

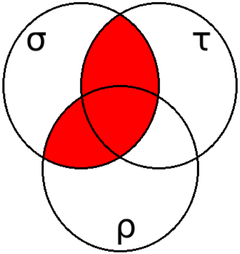
$$\sigma \cup \sigma \leqslant \sigma$$

$$\sigma \leqslant \sigma \cup \tau$$

$$\tau \leqslant \sigma \cup \tau$$

$$\sigma_1 \leqslant \sigma_2, \tau_1 \leqslant \tau_2 \Rightarrow \sigma_1 \cup \tau_1 \leqslant \sigma_2 \cup \tau_2$$ Union compositionality

Subtyping rules 3/4



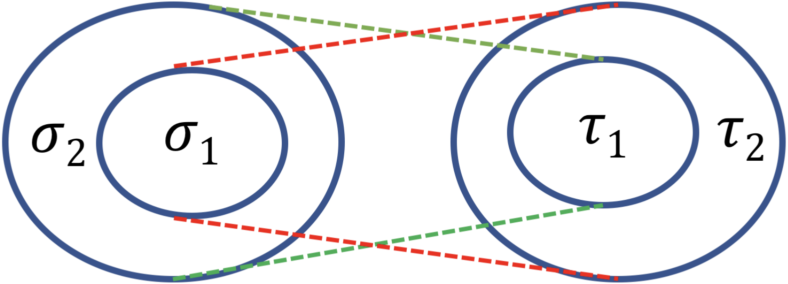
$$\sigma \cap (\tau \cup \rho) \leqslant (\sigma \cap \tau) \cup (\sigma \cap \rho)$$ Distr. of inter over union

$$(\sigma \to \tau) \cap (\sigma \to \rho) \leqslant \sigma \to (\tau \cap \rho)$$ Codomain factorization

$$(\sigma \to \rho) \cap (\tau \to \rho) \leqslant (\sigma \cup \tau) \to \rho$$ Domain factorization

Distributivity of union over intersection can be inferred, so there is no need for another distributivity axiom

Subtyping rules 4/4



Domain contravariance and codomain variance

$$\sigma_1 \leqslant \sigma_2, \tau_1 \leqslant \tau_2 \Rightarrow \sigma_2 \to \tau_1 \leqslant \sigma_1 \to \tau_2$$

Union types as a dual of intersection types

- Union types $\bigcup$ [MacQueen-Plotkin-Sethi '85] are considered as a dual of intersection types

$$\frac{\Gamma, x{:}\sigma \vdash M : \rho \quad \Gamma, x{:}\tau \vdash M : \rho \quad \Gamma \vdash N : \sigma \cup \tau}{\Gamma \vdash M[N/x] : \rho} \; (\cup E)$$

- Union corresponds "roughly" to OCaml Sum types (via `match`)

`type 'a or = In1 of 'a | In2 of 'a ;;`	'a is a type variable
`let f x = match x with`	case analysis on the shape of x
`| In1 y -> "case 1"`	first case
`| In2 y -> "case 2"`	second case
`;;`	

- The big difference between Sum and Union types is that, for Union types, all cases should have the same structure and "set" disjoint

Intersection and union are super expressive

The Forsythe code [by Pierce 91]

$$\text{Is\_0} \stackrel{def}{=} \lambda n.\text{if } n{=}0 \text{ then } \mathit{true} \text{ else } \mathit{false} : \sigma$$

$$\sigma \stackrel{def}{=} (\mathit{Zero} \to \mathit{True}) \cap (\mathit{Neg} \to \mathit{False}) \cap (\mathit{Pos} \to \mathit{False})$$

$$\text{Not\_0} \stackrel{def}{=} \lambda n.\text{if } n \neq 0 \text{ then } 1 \text{ else } -1 : \mathit{Num} \to (\mathit{Pos} \cup \mathit{Neg})$$

$$\text{Is\_0} \ \ (\text{Not\_0} \ \ n) \ : \ \mathit{False}$$

Without union types the best information we can get for Is_0 (Not_0 n) is a Bool type

So intersection and union types allow a restricted form of ABSTRACT INTERPRETATION

REPETITA JUVANT: Propositional Logic vs Intersection (and Union) Types

$\cap$ is not $\wedge$

The dual type of intersection is Union:

$\cup$ is not $\vee$

Since the meaning of $\cap$ is reasonably clear (to claim that $A \cap B$ is to claim that one has a reason for asserting A which is also a reason for asserting B), it would obviously be of interest to figure out how to add $\cap$ to intuitionist logic and then consider the analysis of intuitionist mathematical reasoning in the light of the resulting system.

NB: Usual *intuitionistic logics* do not apply for intersection and union

Proof-functional logics for UNION

We can extend the Pottinger '80 "logical" interpretation of union as an *intuitionistic connective*, by stating that:

DISJUNCTION: "If one has a reason for asserting $A \to C$ and another reason for asserting $B \to C$ and another reason to assert $A \vee B$ then one can assert C"

... (while) ...

UNION: "If one has a reason for asserting both $A \to C$ and $B \to C$ and another reason to assert $A \cup B$ then one can assert C"

$$\frac{\overset{\mathcal{P}_1}{\underset{\Downarrow}{}}\ A \to C \qquad \overset{\mathcal{P}_2}{\underset{\Downarrow}{}}\ B \to C \qquad \overset{\mathcal{P}_3}{\underset{\Downarrow}{}}\ A \vee B}{C} \qquad\qquad \frac{\overset{\mathcal{P}_1}{\underset{\Downarrow}{}}\ A \to C \qquad \overset{\mathcal{P}_1}{\underset{\Downarrow}{}}\ B \to C \qquad \overset{\mathcal{P}_2}{\underset{\Downarrow}{}}\ A \cup B}{C}$$

Extending the Δ-calculus with UNION

σ	$::=$	$\phi \mid \sigma \to \sigma \mid \sigma \cap \sigma \mid \sigma \cup \sigma \mid \mathtt{U}$	types
Δ	$::=$	$x \mid \lambda x{:}\sigma.\Delta \mid \Delta\,\Delta \mid$	typed λ-calculus
		$\langle \Delta , \Delta \rangle \mid pr_1\,\Delta \mid pr_2\,\Delta \mid$	strong pairs and projections
		$[\Delta , \Delta] \mid$	strong sums
		$in_1^\sigma\,\Delta \mid in_2^\sigma\,\Delta \mid$	injections for strong sum
		$\Delta^\sigma \mid$	explicit coercions
		u_Δ	indexed constants

Reconstructing the essence M from a Δ-term

- Fix the relation between pure λ-terms and typed Δ-terms
- Consider the following "erasing" partial function $\wr - \wr$

$$
\begin{aligned}
\wr\, pr_i\, \Delta \,\wr &\stackrel{def}{=} \wr\, \Delta \,\wr \\
\wr\, \langle \Delta_1 \,, \Delta_2 \rangle \,\wr &\stackrel{def}{=} \wr\, \Delta_1 \,\wr \\
\wr\, in_i^{\sigma}\, \Delta \,\wr &\stackrel{def}{=} \wr\, \Delta \,\wr \\
\wr\, [\lambda x{:}\sigma.\Delta_1 \,, \lambda x{:}\tau.\Delta_2]\, \Delta_3 \,\wr &\stackrel{def}{=} \wr\, \Delta_1 \,\wr [\wr\, \Delta_3 \,\wr / x]
\end{aligned}
$$

- Example:

$$\wr\, [\lambda x{:}\sigma.in_2^{\tau} x \,, \lambda x{:}\tau.in_1^{\sigma} x]\, y \,\wr \;=\; y$$

Semantics and properties of the Δ-calculus

Standard β-reduction

$$(\lambda x{:}\sigma.\Delta_1)\,\Delta_2 \quad \longrightarrow_\beta \quad \Delta_1[\Delta_2/x]$$

Projection rules

$$pr_1\,\langle\Delta_1\,,\Delta_2\rangle \quad \longrightarrow_{pr_1} \quad \Delta_1$$

$$pr_2\,\langle\Delta_1\,,\Delta_2\rangle \quad \longrightarrow_{pr_2} \quad \Delta_2$$

Injection rules

$$[\lambda x{:}\sigma.\Delta_1\,,\lambda x{:}\tau.\Delta_2]\,in_1^\sigma\,\Delta_3 \quad \longrightarrow_{\mathrm{in}_1} \quad \Delta_1[\Delta_3/x]$$

$$[\lambda x{:}\sigma.\Delta_1\,,\lambda x{:}\tau.\Delta_2]\,in_2^\tau\,\Delta_3 \quad \longrightarrow_{\mathrm{in}_2} \quad \Delta_2[\Delta_3/x]$$

The usual properties hold: isomorphism wrt the Curry-style system, Church-Rosser, subject reduction for parallel reduction, unicity of typing, decidability of type checking, and type reconstruction

Extending the typing rules of Δ-calculus with Union

$$\frac{\Gamma \vdash \Delta_1 : \sigma \qquad \Gamma \vdash \Delta_2 : \tau \quad \wr \Delta_1 \wr \equiv \wr \Delta_2 \wr}{\Gamma \vdash \langle \Delta_1 \,, \Delta_2 \rangle : \sigma \cap \tau} \; (\cap I) \qquad \frac{\Gamma \vdash \Delta : \sigma_1 \cap \sigma_2 \quad i \in \{1,2\}}{\Gamma \vdash pr_i\, \Delta : \sigma_i} \; (\cap E_i)$$

$$\frac{\Gamma \vdash \Delta : \sigma_i \quad i \in \{1,2\}}{\Gamma \vdash in_i^{\sigma_j} \Delta : \sigma_1 \cup \sigma_2} \; (\cup I_i) \qquad \frac{\Gamma, x{:}\sigma \vdash \Delta_1 : \rho \;\; \wr \Delta_1 \wr \equiv \wr \Delta_2 \wr \qquad \Gamma, x{:}\tau \vdash \Delta_2 : \rho \;\; \Gamma \vdash \Delta_3 : \sigma \cup \tau}{\Gamma \vdash [\lambda x{:}\sigma.\Delta_1 \,, \lambda x{:}\tau.\Delta_2]\, \Delta_3 : \rho} (\cup E)$$

Subtyping rules of theory Ξ from [BDdL]

The subtyping relation from [BDdL] defines a lattice with $\mathtt{U}$ as the top, $\cup$ as the join operator and $\cap$ as the meet operator

$$\sigma \leqslant \sigma \cap \sigma$$

$$\sigma \cup \sigma \leqslant \sigma$$

$$\sigma \cap \tau \leqslant \sigma \text{ and } \sigma \cap \tau \leqslant \tau$$

$$\sigma \leqslant \sigma \cup \tau \text{ and } \tau \leqslant \sigma \cup \tau$$

$$\sigma \leqslant \mathtt{U}$$

$$\sigma \leqslant \sigma$$

$$\sigma_1 \leqslant \sigma_2 \text{ and } \tau_1 \leqslant \tau_2 \Rightarrow \sigma_1 \cap \tau_1 \leqslant \sigma_2 \cap \tau_2$$

$$\sigma_1 \leqslant \sigma_2 \text{ and } \tau_1 \leqslant \tau_2 \Rightarrow \sigma_1 \cup \tau_1 \leqslant \sigma_2 \cup \tau_2$$

$$\sigma \leqslant \tau \text{ and } \tau \leqslant \rho \Rightarrow \sigma \leqslant \rho$$

$$\sigma \cap (\tau \cup \rho) \leqslant (\sigma \cap \tau) \cup (\sigma \cap \rho)$$

$$(\sigma \to \tau) \cap (\sigma \to \rho) \leqslant \sigma \to (\tau \cap \rho)$$

$$(\sigma \to \rho) \cap (\tau \to \rho) \leqslant (\sigma \cup \tau) \to \rho$$

$$\mathtt{U} \leqslant \mathtt{U} \to \mathtt{U}$$

$$\sigma_2 \leqslant \sigma_1 \text{ and } \tau_1 \leqslant \tau_2 \Rightarrow \sigma_1 \to \tau_1 \leqslant \sigma_2 \to \tau_2$$

Raising the Δ-calculus to a Δ-framework

- Adding union types as dual types to intersection types
- Adding dependent types *à la* LF and found a Curry-Howard Isomorphism
- States a Curry-Howard isomorphism for Union and Intersection
- *Proof as Δ-terms and Intersection/Union types as Logical Formulae*
- NB: usual Intuitionistic Logics do not apply to Union and Intersection

The Δ-framework (*à la* Edinburgh LF)

Kinds	K	$::=$	$\mathsf{Type} \mid \Pi x{:}\sigma.K$	as in LF
Families	σ, τ	$::=$	$a \mid \Pi x{:}\sigma.\tau \mid \sigma\,\Delta \mid$	as in LF
			$\sigma \to^{\mathsf{r}} \tau \mid$	relevant arrow
			$\sigma \cap \tau \mid$	intersection
			$\sigma \cup \tau$	union
Objects	Δ	$::=$	$c \mid x \mid \lambda x{:}\sigma.\Delta \mid \Delta\,\Delta \mid$	as in LF
			$\lambda^{\mathsf{r}} x{:}\sigma.\Delta \mid \Delta\,{}^{\mathsf{r}}\Delta \mid$	relevant λ
			$\langle \Delta\,,\Delta \rangle \mid$	pairs for intersection
			$[\Delta\,,\Delta] \mid$	pairs for union
			$pr_1\,\Delta \mid pr_2\,\Delta \mid$	projections
			$in_1^{\sigma}\,\Delta \mid in_2^{\sigma}\,\Delta$	injections

The Bull software

- Bull is an interactive software which implements the Δ-framework
- Developed from scratch in OCaml
- It contains
  - a Read-Eval-Print Loop
  - a typechecker with refinement types
  - an evaluator
  - a decidable and Coq proved and code extracted algorithm for subtyping
  - a higher-order unifier

The language of Bull

$$\begin{array}{rcl} \Delta, \sigma & ::= & s \mid c \mid x \mid \_ \mid ?x[\Delta; \ldots; \Delta] \mid \textbf{let } x{:}\sigma := \Delta \textbf{ in } \Delta \mid \Pi x{:}\sigma.\Delta \mid \\ & & \lambda x{:}\sigma.\Delta \mid \Delta\, S \mid \sigma \cap \sigma \mid \sigma \cup \sigma \mid \langle \Delta \,, \Delta \rangle \mid pr_1\, \Delta \mid pr_2\, \Delta \mid \\ & & \textbf{smatch } \Delta \textbf{ return } \sigma \textbf{ with } [x{:}\sigma \Rightarrow \Delta \mid x{:}\sigma \Rightarrow \Delta] \mid \\ & & \mathrm{in}_1\, \sigma\, \Delta \mid \mathrm{in}_2\, \sigma\, \Delta \mid \textbf{coe}\, \sigma\, \Delta \end{array}$$

- Applications use spines, as in [Cervesato-Pfenning], *i.e.* lists of arguments

$$S ::= () \mid (S; \Delta)$$

- Meta-variables $?x[\Delta; \ldots; \Delta]$ use suspended substitutions: if we know that

$$z{:}\sigma \vdash ?y : \tau$$

- We have

$$(\lambda x{:}\sigma.?y[z := x])\, \Delta \quad \longrightarrow_\beta \quad ?y[z := \Delta] : \tau[z := \Delta]$$

Interacting with Bull

```
Welcome to Bull 1.0, an experimental LF−based proof checker
    based on union, intersection, and relevant types.
Enter "Help." for help.

> Axiom s : Type.
s is assumed.

> Definition id : (s −> s) & (s −> s −> s −> s) :=
  let id1 x := x in let id2 x := x in <id1, id2>.
```

Interacting with Bull

```
Welcome to Bull 1.0, an experimental LF-based proof checker
    based on union, intersection, and relevant types.
Enter "Help." for help.

> Axiom s : Type.
s is assumed.

> Definition id : (s -> s) & (s -> s -> s -> s) :=
  let id1 x := x in let id2 x := x in <id1, id2>.

Definition id : (s -> s) & (s -> s -> s -> s) :=
  let id1 x := x in let id2 x := x in <id1, id2>.
                                           ^^^

Error: the term "id2" has type "?t[] -> ?t[]"
  while it is expected to have type "s -> s -> s -> s".
```

Interacting with Bull

```
Welcome to Bull 1.0, an experimental LF-based proof checker
    based on union, intersection, and relevant types.
Enter "Help." for help.

> Axiom s : Type.
s is assumed.

> Definition id : (s -> s) & ((s -> s) -> s -> s) :=
  let id1 x := x in let id2 x := x in <id1, id2>.
id is defined.
```

Interacting with Bull

```
Welcome to Bull 1.0, an experimental LF−based proof checker
    based on union, intersection, and relevant types.
Enter "Help." for help.

> Axiom s : Type.
s is assumed.

> Definition id : (s -> s) & ((s -> s) -> s -> s) :=
  let id1 x := x in let id2 x := x in <id1, id2>.
id is defined.

> Definition auto_app (f : (s -> s) & ((s -> s) -> s -> s)) := proj_r f proj_l f.
auto_app is defined.

> Compute (auto_app id).
fun x : s => x : s -> s
   essence = fun x => x : s -> s
```

Reduction rules of Bull

- β-reduction: $(\lambda x{:}\sigma.\Delta_1)\,\Delta_2 \longrightarrow_\beta \Delta_1[\Delta_2/x]$
- η-reduction: $\lambda x{:}\sigma.\Delta\, x \longrightarrow_\eta \Delta$ if $x \notin FV(\Delta)$
- smatch-reduction:
  smatch $\mathsf{in}_i\,\Delta_3$ **return** σ **with** $[x{:}\tau \Rightarrow \Delta_1 \mid x{:}\rho \Rightarrow \Delta_2] \longrightarrow_{\mathsf{in}_i} \Delta_i[\Delta_3/x]$
- pr_i-reduction: $pr_i\,\langle\Delta_1\,,\Delta_2\rangle \longrightarrow_{pr_i} \Delta_i$
- δ-reduction: if c is defined as Δ, then $c \longrightarrow_\delta \Delta$
- ζ-reduction: **let** $x{:}\sigma := \Delta_1$ **in** $\Delta_2 \longrightarrow_\zeta \Delta_2[\Delta_1/x]$

Easing the work of the programmer

- In this example, the types of `id1` and `id2` are inferred:

```
> Definition id : (s -> s) & ((s -> s) -> s -> s) :=
    let id1 x := x in let id2 x := x in <id1, id2>.
id is defined.
```

- Also, error reports focus precisely on the culprit:

```
Definition id : (s -> s) & (s -> s -> s -> s) :=
    let id1 x := x in let id2 x := x in <id1, id2>.
                                              ^^^

Error: the term "id2" has type "?t[] -> ?t[]"
    while it is expected to have type "s -> s -> s -> s".
```

- The algorithms we use in order to achieve this are a unifier and a refiner

Unification and refinement

- There is a meta-environment for meta-variables and their instanciation

$$\Phi \quad ::= \quad \cdot \mid \Phi, \mathbf{sort}(?x) \mid \Phi, ?x := s \mid \Phi, (\Gamma \vdash ?x : \sigma) \mid$$
$$\Phi, (\Gamma \vdash ?x := \Delta : \sigma) \mid \Phi, \Psi \vdash ?x \mid \Phi, \Psi \vdash ?x := M$$

- We use Higher-Order Pattern Unification. Judgments are

$$\Phi_1; \Sigma; \Gamma \vdash \Delta_1 \stackrel{?}{=} \Delta_2 \stackrel{\mathcal{U}}{\leadsto} \Phi_2$$

- Refinement is done the same way as in Matita. Judgments are

$$\begin{array}{lcl}
\Phi_1; \Sigma; \Gamma & \vdash & \Delta_1 \stackrel{\Uparrow}{\leadsto} \Delta_2 : \sigma; \Phi_2 \\
\Phi_1; \Sigma; \Gamma & \vdash & \sigma_1 \stackrel{\mathcal{F}}{\leadsto} \sigma_2 : \tau; \Phi_2 \\
\Phi_1; \Sigma; \Gamma & \vdash & \Delta_1 : \sigma \stackrel{\Downarrow}{\leadsto} \Delta_2; \Phi_2 \\
\Phi_1; \Sigma; \Psi & \vdash & \Delta \stackrel{\mathcal{E}^{\Uparrow}}{\leadsto} M; \Phi_2 \\
\Phi_1; \Sigma; \Psi & \vdash & M@\Delta \stackrel{\mathcal{E}^{\Downarrow}}{\leadsto} \Phi_2
\end{array}$$

Conclusion and future work

- We have presented different Church-style λ-calculi with intersection, union, relevant arrow, and dependent types
- We have studied their meta-properties
- We have developed Bull, a proof-of-concept logical framework
- Future: logical interpretation of intersection and union
- Future: enhance the Δ-framework with inductive types

$$
\begin{tikzcd}
 & \Delta^{\mathrm{CDV}}_{=_{\beta\eta}} \arrow[rr] & & \Delta^{\mathrm{BCD}}_{=_{\beta\eta}} \\
 & \Delta^{\mathrm{CDV}}_{=_{\beta}} \arrow[u] \arrow[rr] & & \Delta^{\mathrm{BCD}}_{=_{\beta}} \arrow[u] \\
\Delta^{\mathrm{CD}}_{=_{\beta}} \arrow[ur] \arrow[rr] & & \Delta^{\mathrm{CDS}}_{=_{\beta}} \arrow[ur] & \\
 & \Delta^{\mathrm{CDV}}_{\equiv} \arrow[uu] \arrow[rr] & & \Delta^{\mathrm{BCD}}_{\equiv} \arrow[uu] \\
\Delta^{\mathrm{CD}}_{\equiv} \arrow[uu] \arrow[ur] \arrow[rr] & & \Delta^{\mathrm{CDS}}_{\equiv} \arrow[uu] \arrow[ur] &
\end{tikzcd}
$$

Thank you for listening

$$
\begin{tikzcd}
 & \Delta^{\mathrm{CDV}}_{=_{\beta\eta}} \arrow[rr] & & \Delta^{\mathrm{BCD}}_{=_{\beta\eta}} \\
 & \Delta^{\mathrm{CDV}}_{=_{\beta}} \arrow[u] \arrow[rr] & & \Delta^{\mathrm{BCD}}_{=_{\beta}} \arrow[u] \\
\Delta^{\mathrm{CD}}_{=_{\beta}} \arrow[ur] \arrow[rr] & & \Delta^{\mathrm{CDS}}_{=_{\beta}} \arrow[ur] & \\
 & \Delta^{\mathrm{CDV}}_{\equiv} \arrow[uu] \arrow[rr] & & \Delta^{\mathrm{BCD}}_{\equiv} \arrow[uu] \\
\Delta^{\mathrm{CD}}_{\equiv} \arrow[uu] \arrow[ur] \arrow[rr] & & \Delta^{\mathrm{CDS}}_{\equiv} \arrow[uu] \arrow[ur] &
\end{tikzcd}
$$

The Δ-calculus: syntax and types

https://arxiv.org/abs/1803.09660

Joint works with Claude Stolze, Dan Dougherty, Furio Honsell, Ugo de' Liguoro, Ivan Scagnetto

Proudly supported by COST EUTYPES and, hopefully supported by COST EuroProofNet