Towards a Logical Framework with Intersection and Union Types

Claude Stolze Luigi Liquori
INRIA Sophia-Antipolis Méditerranée, France

Furio Honsell Ivan Scagnetto
Università di Udine, Italy

Plan of the talk

- Proof functional logics vs. Truth functional logics
- The power of intersection and union types *à la* Curry
- *Preludio.* The **Delta-calculus**: $\bigcap$ and $\bigcup$ types *à la* Church

Core 1 Raising the Delta-calculus to the **Delta-framework**: an implementation of the Δ-calculus with dependent-types and relevant arrow-types

Core 2 Encoding of the Delta-calculus in the Delta-framework

- About the current implementation of the Delta-framework
- Related and future works

Proof functional connectives *vs.* (usual) Truth functional connectives

- Intuitionistic logic states that proof should correspond to an object giving all the components of the proof (BHK interpretation): proofs can be encoded in typed λ-calculus
- Pottinger and Lopez-Escobar in the '80 introduced the notion of *proof-functional* connectives ie. operators allow reasoning about the structure of logical proofs
- Logical proofs are raised to the status of *first-class* objects

Intersection and Union are Proof-functional

- An intersection type/formula $\cap$ is a proof-functional connective totally different from a cartesian product $\times$
- *... to assert $\phi \cap \psi$ is to assert that one has a reason (a derivation) for asserting ϕ which is also a reason (a derivation) for asserting ψ*
- Intersection is a “polymorphic" construction, that is, the same evidence can be used as a proof for different sentences

Intersection and Union are Proof-functional

- An intersection type/formula $\cap$ is a proof-functional connective totally different from a cartesian product $\times$
- *... to assert $\phi \cap \psi$ is to assert that one has a reason (a derivation) for asserting ϕ which is also a reason (a derivation) for asserting ψ*
- Intersection is a “polymorphic" construction, that is, the same evidence can be used as a proof for different sentences
- An union type/formula $\cup$ is a proof-functional connective totally different from disjoint union $\vee$
- *... to assert ξ by disjunction on $\phi \cup \psi$ is to assert ξ using the same reason (derivation) in both the cases of the disjunction ϕ or ψ*
- Union types is a polymorphic construction, that is, a proof for ϕ is also a proof for $\phi \cup \psi$
- Union types represent also a form of “uncertain” construction, that is, a proof for $\phi \cup \psi$ “could" be either a proof for ϕ or a proof for ψ

Intersection and Union Types ($\bigcap$ and $\bigcup$)

- Intersection types [Barendregt-Coppo-Dezani,JSL82] are also referred as *ad hoc* polymorphism
- Intersection types characterize the set of strongly normalizable λ-terms
- Girard's *parametric* polymorphism (System F) is equivalent to *ad hoc* polymorphism

$$\forall\alpha.\sigma \triangleq \bigcap_{i=1\ldots\infty} \sigma_i$$

- Union types [McQueen-Plotkin-Sehti] are considered as a dual of intersection types
- Intersection and union types can be used to express conjunctive and disjunctive properties on programs

Type assignment system for $\bigcap$ and $\bigcup$

$$\frac{x{:}\sigma \in B}{B \vdash x : \sigma}\ (Var) \qquad \frac{B \vdash M : \sigma \quad \sigma \leqslant \tau^{\dagger}}{B \vdash M : \tau}\ (\leqslant)$$

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

$$\frac{B \vdash M : \sigma \quad B \vdash M : \tau}{B \vdash M : \sigma \cap \tau}\ ({\cap}I) \qquad \frac{B \vdash M : \sigma_1 \cap \sigma_2 \quad i = 1,2}{B \vdash M : \sigma_i}\ ({\cap}E_i)$$

$$\frac{B \vdash M : \sigma_i \quad i = 1,2}{B \vdash M : \sigma_1 \cup \sigma_2}\ ({\cup}I_i) \qquad \frac{\begin{array}{l} B, x{:}\sigma \vdash M : \rho \\ B, x{:}\tau \vdash M : \rho \quad B \vdash N : \sigma \cup \tau \end{array}}{B \vdash M\{N/x\} : \rho}\ ({\cup}E)$$

---

† Suitable subtyping relation for arrow, intersection, and union

Ex: Type assignment judgments with $\bigcap$ and $\bigcup$

- For intersection types: polymorphic identity and self-application

$$\vdash \lambda x.x : (\sigma \to \sigma) \cap (\tau \to \tau)$$

$$\vdash \lambda x.x\,x : ((\sigma \to \tau) \cap \sigma) \to \tau$$

Ex: Type assignment judgments with $\bigcap$ and $\bigcup$

- For intersection types: polymorphic identity and self-application

$$\vdash \lambda x.x : (\sigma \to \sigma) \cap (\tau \to \tau)$$

$$\vdash \lambda x.x\,x : ((\sigma \to \tau) \cap \sigma) \to \tau$$

- For intersection and union types: the Forsythe code by Pierce:

$$\begin{array}{rcl} \text{Test} & \triangleq & \text{if } b \text{ then } 1 \text{ else } -1 : \mathit{Pos} \cup \mathit{Neg} \\ \text{Is\_0} & : & (\mathit{Neg} \to F) \cap (\mathit{Zero} \to T) \cap (\mathit{Pos} \to F) \\ (\text{Is\_0 Test}) & : & F \end{array}$$

Without union types the best information we can get for (Is_0 Test) is a Boolean type

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

- Intersection and union types were defined as type assignment systems (for pure λ-terms)
- Very elegant presentation but undecidability of type checking
- Many attempts of finding decidable and typed λ-calculi with intersection and union types preserving all the good properties of type assignment

?1 The usual approach (adding types to binders) is problematic for $\bigcap$

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

?2 $M\{N/x\}$ in $(\cup E)$ would make the system non syntax directed

Our solution: use Curry-Howard isomorphism

- Based on Dougherty, Liquori, Ronchi, Stolze papers (see biblio)
- Curry-Howard isomorphism is usually used for encoding a logic into a corresponding typed λ-calculus. For example:

$$\lambda x{:}\phi.M : \phi \to \psi \text{ encodes a derivation tree } \mathcal{D} \text{ for } \phi \supset \psi$$

Our solution: use Curry-Howard isomorphism

- Based on Dougherty, Liquori, Ronchi, Stolze papers (see biblio)
- Curry-Howard isomorphism is usually used for encoding a logic into a corresponding typed λ-calculus. For example:

  $\lambda x{:}\phi.M : \phi \to \psi$ encodes a derivation tree $\mathcal{D}$ for $\phi \supset \psi$

- Our solution: we encode a type assignment derivation into our corresponding typed "Δ-term"
- For example the Δ-term

  $$\langle \lambda x{:}\sigma.x \,,\, \lambda x{:}\tau.x \rangle \quad \text{of type} \quad (\sigma \to \sigma) \cap (\tau \to \tau)$$

  encodes a derivation tree $\mathcal{D}$ for

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

- We call $\lambda x.x$ the *essence* of Δ

Syntax of the Δ-calculus

Δ-terms and types are defined as follows:

$$\begin{array}{rcl} \sigma & ::= & \phi \mid \sigma \to \sigma \mid \sigma \cap \sigma \mid \sigma \cup \sigma \\ \Delta & ::= & x \mid \lambda x{:}\sigma.\Delta \mid \Delta\,\Delta \mid \langle \Delta \,, \Delta \rangle \mid [\Delta \,, \Delta] \mid \\ & & \mathrm{pr}_1\,\Delta \mid \mathrm{pr}_2\,\Delta \mid \mathrm{in}_1^{\sigma}\,\Delta \mid \mathrm{in}_2^{\sigma}\,\Delta \end{array}$$

σ	arrow, intersection and union types
Λ^t	typed λ-calculus enriched with ...
$\langle \Delta \,, \Delta \rangle$	strong pair
$[\Delta \,, \Delta]$	strong sum
pr_i	projections for strong product
in_i^{σ}	injections for strong sum

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 $\wr - \wr$

$$
\begin{array}{rcll}
\wr x \wr & \triangleq & x & \\
\wr \lambda x{:}\sigma.\Delta \wr & \triangleq & \lambda x.\wr \Delta \wr & \\
\wr \Delta_1\, \Delta_2 \wr & \triangleq & \wr \Delta_1 \wr\, \wr \Delta_2 \wr & \\
\wr \mathrm{pr}_i\, \Delta \wr & \triangleq & \wr \Delta \wr & \\
\wr \mathrm{in}_i\, \Delta \wr & \triangleq & \wr \Delta \wr & \\
\wr \langle \Delta_1 , \Delta_2 \rangle \wr & \triangleq & \wr \Delta_1 \wr & \text{if } \wr \Delta_1 \wr \equiv \wr \Delta_2 \wr \\
\wr [\lambda x{:}\sigma.\Delta_1 , \lambda x{:}\tau.\Delta_2]\, \Delta_3 \wr & \triangleq & \wr \Delta_1 \wr \{ \wr \Delta_3 \wr / x \} & \text{if } \wr \Delta_1 \wr \equiv \wr \Delta_2 \wr
\end{array}
$$

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 $\wr - \wr$

$$
\begin{array}{rcll}
\wr x \wr & \triangleq & x & \\
\wr \lambda x{:}\sigma.\Delta \wr & \triangleq & \lambda x.\wr \Delta \wr & \\
\wr \Delta_1\ \Delta_2 \wr & \triangleq & \wr \Delta_1 \wr\, \wr \Delta_2 \wr & \\
\wr \mathsf{pr}_i\ \Delta \wr & \triangleq & \wr \Delta \wr & \\
\wr \mathsf{in}_i\ \Delta \wr & \triangleq & \wr \Delta \wr & \\
\wr \langle \Delta_1 , \Delta_2 \rangle \wr & \triangleq & \wr \Delta_1 \wr & \text{if } \wr \Delta_1 \wr \equiv \wr \Delta_2 \wr \\
\wr [\lambda x{:}\sigma.\Delta_1 , \lambda x{:}\tau.\Delta_2]\ \Delta_3 \wr & \triangleq & \wr \Delta_1 \wr \{ \wr \Delta_3 \wr / x \} & \text{if } \wr \Delta_1 \wr \equiv \wr \Delta_2 \wr
\end{array}
$$

- Example:

$$
\begin{array}{rcl}
\wr \mathsf{pr}_1\ \langle \lambda x{:}\sigma.x , \lambda x{:}\tau.x \rangle \wr & = & \lambda x.x \\
\wr [\lambda y{:}\tau.\mathsf{in}_2^{\sigma}\ y , \lambda y{:}\sigma.\mathsf{in}_1^{\tau}\ y]\ x \wr & = & x
\end{array}
$$

Semantics and properties of the Δ-calculus

- Reduction in the Δ-calculus is the usual β-reduction plus

$$\begin{array}{llllll} \mathrm{pr}_1\,\langle\Delta_1\,,\Delta_2\rangle & \longrightarrow_{\mathrm{pr}_1} & \Delta_1 & [\Delta_1\,,\Delta_2]\,\mathrm{in}_1^{\sigma}\,\Delta_3 & \longrightarrow_{\mathrm{in}_1} & \Delta_1\,\Delta_3 \\ \mathrm{pr}_2\,\langle\Delta_1\,,\Delta_2\rangle & \longrightarrow_{\mathrm{pr}_2} & \Delta_2 & [\Delta_1\,,\Delta_2]\,\mathrm{in}_1^{\sigma}\,\Delta_3 & \longrightarrow_{\mathrm{in}_1} & \Delta_1\,\Delta_3 \end{array}$$

- Type system (rules for intersection and union)

$$\frac{\begin{array}{l}\Gamma \vdash \Delta_1 : \sigma \\ \Gamma \vdash \Delta_2 : \tau \quad \wr\Delta_1\wr \equiv \wr\Delta_2\wr\end{array}}{\Gamma \vdash \langle\Delta_1\,,\Delta_2\rangle : \sigma \cap \tau}\ (\cap I) \qquad \frac{\begin{array}{l}\Gamma, x{:}\sigma \vdash \Delta_1 : \rho \ \ \wr\Delta_1\wr \equiv \wr\Delta_2\wr \\ \Gamma, x{:}\tau \vdash \Delta_2 : \rho \ \ \Gamma \vdash \Delta_3 : \sigma \cup \tau\end{array}}{\Gamma \vdash [\lambda x{:}\sigma.\Delta_1\,,\lambda x{:}\tau.\Delta_2]\,\Delta_3 : \rho}\ (\cup E)$$

- Judgments fully encode pure type assignment derivations $\mathcal{D}$ i.e.

$$B \vdash \Delta : \sigma \qquad \text{iff} \qquad \mathcal{D} : B \vdash M : \sigma$$

- The following properties can be proved: Church-Rosser, subject reduction for parallel reduction, unicity of typing, decidability of type checking and type reconstruction

Core 1 Why a proof-functional logical framework?

- Intuitionistic logic has realizers, but we do not reason about these realizers

- Proof-functional logic allows us to define constraints on the shape of the realizers

- It could give us a better understanding of structures of proofs (theoretical point of view), and a sharper encoding of proofs (practical point of view)

Stratified syntax of the Δ-framework

Kinds	K	$::=$	$\text{Type} \mid \Pi x{:}\sigma.K$	as in LF
Families	σ, τ	$::=$	$a \mid \Pi x{:}\sigma.\tau \mid \sigma\ \Delta \mid$	as in LF
			$\Pi^r x{:}\sigma.\tau \mid$	relevant product
			$\sigma \cap \tau \mid$	intersection
			$\sigma \cup \tau$	union
Objects	Δ	$::=$	$c \mid x \mid \lambda x{:}\sigma.\Delta \mid \Delta\ \Delta \mid$	as in LF
			$\lambda^r x{:}\sigma.\Delta \mid$	relevant λ
			$\langle \Delta , \Delta \rangle \mid$	pairs for intersection
			$[\Delta , \Delta] \mid$	pairs for union
			$\text{pr}_1\ \Delta \mid \text{pr}_2\ \Delta \mid$	projections
			$\text{in}_1^\sigma\ \Delta \mid \text{in}_2^\sigma\ \Delta$	injections

Reduction rules of the Δ-framework

Standard β-reduction

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

$$(\lambda^r x{:}\sigma.\Delta_1)\,\Delta_2 \longrightarrow_\beta \Delta_1\{\Delta_2/x\}$$

Projection rules

$$\mathsf{pr}_1\,\langle\Delta_1\,,\Delta_2\rangle \longrightarrow_{\mathsf{pr}_1} \Delta_1$$

$$\mathsf{pr}_2\,\langle\Delta_1\,,\Delta_2\rangle \longrightarrow_{\mathsf{pr}_2} \Delta_2$$

Injection rules

$$[\Delta_1\,,\Delta_2]\,\mathsf{in}_1^\sigma\,\Delta_3 \longrightarrow_{\mathsf{in}_1} \Delta_1\,\Delta_3$$

$$[\Delta_1\,,\Delta_2]\,\mathsf{in}_2^\sigma\,\Delta_3 \longrightarrow_{\mathsf{in}_2} \Delta_2\,\Delta_3$$

Typing Judgments of the Δ-framework

$$\Sigma \quad \text{sig}$$

$$\Gamma \quad \vdash_{\Sigma}$$

$$\Gamma \quad \vdash_{\Sigma} \quad K$$

$$\Gamma \quad \vdash_{\Sigma} \quad \sigma : K$$

$$\Gamma \quad \vdash_{\Sigma} \quad \Delta : \sigma$$

Essence function (now it depends on Γ and Σ)

$$
\begin{array}{rcll}
\wr c \wr^{\Gamma}_{\Sigma} & \triangleq & c & \\
\wr x \wr^{\Gamma}_{\Sigma} & \triangleq & x & \\
\wr \lambda x{:}\sigma.\Delta \wr^{\Gamma}_{\Sigma} & \triangleq & \lambda x.\wr \Delta \wr^{\Gamma}_{\Sigma} & \\
\wr \lambda^{r} x{:}\sigma.\Delta \wr^{\Gamma}_{\Sigma} & \triangleq & \lambda x.\wr \Delta \wr^{\Gamma, x:\sigma}_{\Sigma} & \text{if } \wr \Delta \wr^{\Gamma, x:\sigma}_{\Sigma} \equiv x \\
\wr \langle \Delta_1 , \Delta_2 \rangle \wr^{\Gamma}_{\Sigma} & \triangleq & \wr \Delta_1 \wr^{\Gamma}_{\Sigma} & \text{if } \wr \Delta_1 \wr^{\Gamma}_{\Sigma} \equiv \wr \Delta_2 \wr^{\Gamma}_{\Sigma} \\
\wr [\lambda x{:}\sigma.\Delta_1 , \lambda x{:}\tau.\Delta_2]\, \Delta_3 \wr^{\Gamma}_{\Sigma} & \triangleq & \wr \Delta_1 \wr^{\Gamma}_{\Sigma} \{ \wr \Delta_3 \wr^{\Gamma}_{\Sigma} / x \} & \text{if } \wr \Delta_1 \wr^{\Gamma}_{\Sigma} \equiv \wr \Delta_2 \wr^{\Gamma}_{\Sigma} \\
\wr [\Delta_1 , \Delta_2] \wr^{\Gamma}_{\Sigma} & \triangleq & \wr \Delta_1 \wr^{\Gamma}_{\Sigma} & \text{if } \wr \Delta_1 \wr^{\Gamma}_{\Sigma} \equiv \wr \Delta_2 \wr^{\Gamma}_{\Sigma} \\
\wr \mathrm{pr}_i\, \Delta \wr^{\Gamma}_{\Sigma} & \triangleq & \wr \Delta \wr^{\Gamma}_{\Sigma} & \\
\wr \mathrm{in}^{\sigma}_i\, \Delta \wr^{\Gamma}_{\Sigma} & \triangleq & \wr \Delta \wr^{\Gamma}_{\Sigma} & \\
\wr \Delta_1\, \Delta_2 \wr^{\Gamma}_{\Sigma} & \triangleq & \begin{cases} \wr \Delta_2 \wr^{\Gamma}_{\Sigma} & \text{if } \Gamma \vdash_{\Sigma} \Delta_1 : \Pi^{r} x{:}\sigma.\tau \\ \wr \Delta_1 \wr^{\Gamma}_{\Sigma}\, \wr \Delta_2 \wr^{\Gamma}_{\Sigma} & \text{otherwise} \end{cases} &
\end{array}
$$

Q? Why $\wr\Delta_1\wr \equiv \wr\Delta_2\wr$ and not $\wr\Delta_1\wr =_\beta \wr\Delta_2\wr$?

- We could try to replace this condition by $\wr\Delta_1\wr =_\beta \wr\Delta_2\wr$
- However, for any pure λ-term, we can find a corresponding well-typed Δ-term
- For instance, in the signature

$$\Sigma \triangleq \sigma{:}\mathrm{Type}, c_1{:}(\sigma \to \sigma) \to^r \sigma, c_2{:}\sigma \to^r (\sigma \to \sigma)$$

  the Δ-term

$$(\lambda x{:}\sigma.(c_2\, x)\, x)(c_1\, (\lambda x{:}\sigma.(c_2\, x)\, x))$$

  has type σ and its essence is

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

- As a consequence, β-equality of essences is undecidable

Valid signatures, contexts, and kinds

Valid Signatures

$$\frac{}{\langle \omega\text{:Type}\rangle \text{ sig}}\ (\omega\Sigma) \qquad \frac{\Sigma \text{ sig} \quad \vdash_\Sigma K \quad a \notin \text{dom}(\Sigma)}{\Sigma, a{:}K \text{ sig}}\ (K\Sigma)$$

$$\frac{\Sigma \text{ sig} \quad \vdash_\Sigma \sigma : \text{Type} \quad c \notin \text{dom}(\Sigma)}{\Sigma, c{:}\sigma \text{ sig}}\ (\sigma\Sigma)$$

Valid Contexts

$$\frac{\Sigma \text{ sig}}{\vdash_\Sigma \langle\,\rangle}\ (\epsilon\Gamma) \qquad \frac{\vdash_\Sigma \Gamma \quad \Gamma \vdash_\Sigma \sigma : \text{Type} \quad x \notin \text{dom}(\Gamma)}{\vdash_\Sigma \Gamma, x{:}\sigma}\ (\sigma\Gamma)$$

Valid Kinds

$$\frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma \text{Type}}\ (\mathit{Type}) \qquad \frac{\Gamma, x{:}\sigma \vdash_\Sigma K}{\Gamma \vdash_\Sigma \Pi x{:}\sigma.K}\ (\Pi K)$$

Valid families

$$\frac{\vdash_\Sigma \Gamma \quad a{:}K \in \Sigma}{\Gamma \vdash_\Sigma a : K} \; (\mathit{Const})$$

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma \tau : \mathrm{Type}}{\Gamma \vdash_\Sigma \Pi x{:}\sigma.\tau : \mathrm{Type}} \; (\Pi I) \qquad \frac{\Gamma, x{:}\sigma \vdash_\Sigma \tau : \mathrm{Type}}{\Gamma \vdash_\Sigma \Pi^r x{:}\sigma.\tau : \mathrm{Type}} \; (\Pi^r I)$$

$$\frac{\Gamma \vdash_\Sigma \sigma : \Pi x{:}\tau.K \quad \Gamma \vdash_\Sigma \Delta : \tau}{\Gamma \vdash_\Sigma \sigma\, \Delta : K\{\Delta/x\}} \; (\Pi E) \qquad \frac{\Gamma \vdash_\Sigma \sigma : \Pi^r x{:}\tau.K \quad \Gamma \vdash_\Sigma \Delta : \tau}{\Gamma \vdash_\Sigma \sigma\, \Delta : K\{\Delta/x\}} \; (\Pi^r E)$$

$$\frac{\Gamma \vdash_\Sigma \sigma : \mathrm{Type} \quad \Gamma \vdash_\Sigma \tau : \mathrm{Type}}{\Gamma \vdash_\Sigma \sigma \cap \tau : \mathrm{Type}} \; (\cap I) \qquad \frac{\Gamma \vdash_\Sigma \sigma : \mathrm{Type} \quad \Gamma \vdash_\Sigma \tau : \mathrm{Type}}{\Gamma \vdash_\Sigma \sigma \cup \tau : \mathrm{Type}} \; (\cup I)$$

$$\frac{\Gamma \vdash_\Sigma \sigma : K_1 \quad \Gamma \vdash_\Sigma K_2 \quad K_1 =_{} K_2}{\Gamma \vdash_\Sigma \sigma : K_2} \; (\mathit{Conv})$$

Valid objects (I)

$$\frac{\vdash_\Sigma \Gamma \quad c{:}\sigma \in \Sigma}{\Gamma \vdash_\Sigma c : \sigma}\ (\mathit{Const}) \qquad \frac{\vdash_\Sigma \Gamma \quad x{:}\sigma \in \Gamma}{\Gamma \vdash_\Sigma x : \sigma}\ (\mathit{Var})$$

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma \Delta : \tau}{\Gamma \vdash_\Sigma \lambda x{:}\sigma.\Delta : \Pi x{:}\sigma.\tau}\ (\Pi I) \qquad \frac{\Gamma \vdash_\Sigma \Delta_1 : \Pi x{:}\sigma.\tau \quad \Gamma \vdash_\Sigma \Delta_2 : \sigma}{\Gamma \vdash_\Sigma \Delta_1\, \Delta_2 : \tau\{\Delta_2/x\}}\ (\Pi E)$$

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma \Delta : \tau \quad \wr\Delta\wr^\Gamma_\Sigma \equiv x}{\Gamma \vdash_\Sigma \lambda^r x{:}\sigma.\Delta : \Pi^r x{:}\sigma.\tau}\ (\Pi^r I) \qquad \frac{\Gamma \vdash_\Sigma \Delta_1 : \Pi^r x{:}\sigma.\tau \quad \Gamma \vdash_\Sigma \Delta_2 : \sigma}{\Gamma \vdash_\Sigma \Delta_1\, \Delta_2 : \tau\{\Delta_2/x\}}\ (\Pi^r E)$$

$$\frac{\Gamma \vdash_\Sigma \Delta : \sigma \quad \Gamma \vdash_\Sigma \tau : \mathrm{Type} \quad \sigma =_\beta \tau}{\Gamma \vdash_\Sigma \Delta : \tau}\ (\mathit{Conv})$$

Valid objects (II)

$$\frac{\Gamma \vdash_\Sigma \Delta_1 : \sigma \quad \Gamma \vdash_\Sigma \Delta_2 : \tau \quad \wr\Delta_1\wr_\Sigma^\Delta \equiv \wr\Delta_2\wr_\Sigma^\Delta}{\Gamma \vdash_\Sigma \langle\Delta_1 , \Delta_2\rangle : \sigma \cap \tau} \; (\cap I)$$

$$\frac{\Gamma \vdash_\Sigma \Delta : \sigma \cap \tau}{\Gamma \vdash_\Sigma \mathrm{pr}_1\, \Delta : \sigma} \; (\cap E_l) \qquad \frac{\Gamma \vdash_\Sigma \Delta : \sigma \cap \tau}{\Gamma \vdash_\Sigma \mathrm{pr}_2\, \Delta : \tau} \; (\cap E_r)$$

$$\frac{\Gamma \vdash_\Sigma \Delta : \sigma \quad \Gamma \vdash_\Sigma \sigma \cup \tau : \mathrm{Type}}{\Gamma \vdash_\Sigma \mathrm{in}_1^\tau\, \Delta : \sigma \cup \tau} \; (\cup I_l) \qquad \frac{\Gamma \vdash_\Sigma \Delta : \tau \quad \Gamma \vdash_\Sigma \sigma \cup \tau : \mathrm{Type}}{\Gamma \vdash_\Sigma \mathrm{in}_2^\sigma\, \Delta : \sigma \cup \tau} \; (\cup I_r)$$

$$\frac{\begin{array}{c}\Gamma \vdash_\Sigma \Delta_1 : \Pi y{:}\sigma.\rho\{\mathrm{in}_1^\tau\, y/x\} \quad \wr\Delta_1\wr_\Sigma^\Gamma \equiv \wr\Delta_2\wr_\Sigma^\Gamma \\ \Gamma \vdash_\Sigma \Delta_2 : \Pi y{:}\tau.\rho\{\mathrm{in}_2^\sigma\, y/x\} \quad \Gamma \vdash_\Sigma \Delta_3 : \sigma \cup \tau\end{array}}{\Gamma \vdash_\Sigma [\Delta_1 , \Delta_2]\, \Delta_3 : \rho\{\Delta_3/x\}} \; (\cup E)$$

Alternative definition for $(\cup E)$

Higher-order unification is undecidable, so we don't know how to infer the type ρ in the rule $(\cup E)$.

$$\frac{\Gamma \vdash_\Sigma \Delta_1 : \Pi y{:}\sigma.\rho\{\mathrm{in}_1^\tau\, y/x\} \quad \wr\Delta_1\wr_\Sigma^\Gamma \equiv \wr\Delta_2\wr_\Sigma^\Gamma \quad \Gamma \vdash_\Sigma \Delta_2 : \Pi y{:}\tau.\rho\{\mathrm{in}_2^\sigma\, y/x\} \quad \Gamma \vdash_\Sigma \Delta_3 : \sigma \cup \tau}{\Gamma \vdash_\Sigma [\Delta_1 , \Delta_2]\, \Delta_3 : \rho\{\Delta_3/x\}}\ (\cup E)$$

Alternative definition for $(\cup E)$

Higher-order unification is undecidable, so we don't know how to infer the type ρ in the rule $(\cup E)$.

$$\frac{\Gamma \vdash_\Sigma \Delta_1 : \Pi y{:}\sigma.\rho\{\mathrm{in}_1^\tau\ y/x\} \quad \wr\Delta_1\wr_\Sigma^\Gamma \equiv \wr\Delta_2\wr_\Sigma^\Gamma \qquad \Gamma \vdash_\Sigma \Delta_2 : \Pi y{:}\tau.\rho\{\mathrm{in}_2^\sigma\ y/x\} \quad \Gamma \vdash_\Sigma \Delta_3 : \sigma \cup \tau}{\Gamma \vdash_\Sigma [\Delta_1 , \Delta_2]\, \Delta_3 : \rho\{\Delta_3/x\}}\ (\cup E)$$

$$\frac{\Gamma \vdash_\Sigma \Delta_3 : \sigma \cup \tau \qquad \Gamma \vdash_\Sigma \Delta_1 : \Pi y{:}\sigma.\rho\,(\mathrm{in}_1^\tau\ y) \quad \wr\Delta_1\wr_\sigma^\Gamma \equiv \wr\Delta_2\wr_\sigma^\Gamma \qquad \Gamma \vdash_\Sigma \Delta_2 : \Pi y{:}\tau.\rho\,(\mathrm{in}_2^\sigma\ y) \quad \Gamma \vdash_\Sigma \rho : \Pi y{:}(\sigma \cup \tau).\mathrm{Type}}{\Gamma \vdash_\Sigma [\Delta_1 , \Delta_2]_\rho\, \Delta_3 : \rho\, \Delta_3}\ (\cup E)_{\mathrm{implemented}}$$

In the implementation, we ask the user to explicitly give ρ (similarly to the return keyword in the Coq match operator)

Exemple: dependent auto-application in the Δ-framework

Let $\Sigma \triangleq \sigma{:}\mathsf{Type}, \tau{:}\sigma \to \mathsf{Type}$

$$
\dfrac{\dfrac{\dfrac{}{x{:}(\Pi y{:}\sigma.\tau\, y) \cap \sigma \vdash_\Sigma x : (\Pi y{:}\sigma.\tau\, y) \cap \sigma}}{x{:}(\Pi y{:}\sigma.\tau\, y) \cap \sigma \vdash_\Sigma \mathsf{pr}_1\, x : \Pi y{:}\sigma.\tau\, y} \quad \dfrac{\dfrac{}{x{:}(\Pi y{:}\sigma.\tau\, y) \cap \sigma \vdash_\Sigma x : (\Pi y{:}\sigma.\tau\, y) \cap \sigma}}{x{:}(\Pi y{:}\sigma.\tau\, y) \cap \sigma \vdash_\Sigma \mathsf{pr}_2\, x : \sigma}}{\dfrac{x{:}(\Pi y{:}\sigma.\tau\, y) \cap \sigma \vdash_\Sigma (\mathsf{pr}_1\, x)\,(\mathsf{pr}_2\, x) : \tau\,(\mathsf{pr}_2\, x)}{\vdash_\Sigma \lambda x{:}(\Pi y{:}\sigma.\tau\, y) \cap \sigma.(\mathsf{pr}_1\, x)\,(\mathsf{pr}_2\, x) : \Pi x{:}(\Pi y{:}\sigma.\tau\, y) \cap \sigma.\tau\,(\mathsf{pr}_2\, x)}}
$$

Core 2

Encoding examples in LF vs. the Δ-framework

Pure LF encoding of the Δ-calculus

- Because of the expressivity of the Edinburgh LF, encoding the Δ-calculus is possible
- We have to face up the encoding of a proof-functional logic
- In particular, the encoding will face up to equality of two essence of Δ-terms (see $\wr\Delta_1\wr \equiv \wr\Delta_2\wr$)
- Because of this, encoding proof-functional logics is not an easy task
- Important. Thanks to isomorphism between Δ-terms and the type assignment systems derivations, the encoding represent also one encoding (the first?) of the intersection and union type assignment systems

LF encoding of the Δ-calculus (spot 1)

$$
\begin{array}{rcl}
o & : & \text{Type} \\
c_{\to} & : & o \to o \to o \\
c_{\cap} & : & o \to o \to o \\
c_{\cup} & : & o \to o \to o \\
obj & : & o \to \text{Type} \\
=_o & : & \Pi s\, t{:}o. obj\ s \to obj\ t \to \text{Type} \\
r_= & : & \Pi s{:}o. \Pi M{:}obj\ s. =_o\ s\, s\, M\, M \\
s_= & : & \Pi s\, t{:}o. \Pi M{:}obj\ s. \Pi N{:}obj\ t. =_o\ s\, t\, M\, N \to =_o\ t\, s\, N\, M \\
t_= & : & \Pi\, s\, t\, r{:}o. \Pi M{:}obj\ s. \Pi N{:}obj\ t. \Pi O{:}obj\ r. =_o\ s\, t\, M\, N \to \\
 & & \quad =_o\ t\, r\, N\, O \to =_o\ s\, r\, M\, O
\end{array}
$$

LF encoding of the Δ-calculus (spot 2)

$$c_{spair} : \Pi s\, t{:}o.\Pi M{:}obj\, s.\Pi N{:}obj\, t.\ =_o\ s\, t\, M\, N \to obj\,(c_\cap\ s\, t)$$

$$c_{\mathrm{pr}_1} : \Pi s\, t{:}o.\Pi M{:}obj\,(c_\cap s\, t).obj\, s$$

$$c_{\mathrm{pr}_2} : \Pi s\, t{:}o.\Pi M{:}obj\,(c_\cap\ s\, t).obj\, t$$

$$c_{=spair} : \Pi s\, t{:}o.\Pi M{:}obj\, s.\Pi N{:}obj\, t.\Pi Z{:}\ =_o\ s\, t\, M\, N.\ =_o\ (c_\cap\ s\, t)\, s\,(c_{spair}\ s\, t\, M\, N\, Z)\, M$$

$$c_{=\mathrm{pr}_1} : \Pi s\, t{:}o.\Pi M{:}obj\,(c_\cap\ s\, t).{=_o}\ (c_\cap\ s\, t)\, s\, M\,(c_{\mathrm{pr}_1}\ s\, t\, M)$$

$$c_{=\mathrm{pr}_2} : \Pi s\, t{:}o.\Pi M{:}obj\,(c_\cap\ s\, t).{=_o}\ (c_\cap\ s\, t)\, t\, M\,(c_{\mathrm{pr}_2}\ s\, t\, M)$$

Full Coq encoding of the Δ-calculus (see paper)

$$
\begin{array}{rl}
o & : \mathrm{Type} \\
c_{\rightarrow} & : o \rightarrow o \rightarrow o \\
c_{\cap} & : o \rightarrow o \rightarrow o \\
c_{\cup} & : o \rightarrow o \rightarrow o \\
obj & : o \rightarrow \mathrm{Type} \\
=_o & : \Pi s\, t{:}o.\, obj\ s \rightarrow obj\ t \rightarrow \mathrm{Type} \\
r_{=} & : \Pi s{:}o.\Pi M{:}obj\ s.\ =_o\ s\ s\ M\ M \\
s_{=} & : \Pi s\, t{:}o.\Pi M{:}obj\ s.\Pi N{:}obj\ t.\ =_o\ s\ t\ M\ N \rightarrow =_o\ t\ s\ N\ M \\
t_{=} & : \Pi\, s\, t\, r{:}o.\Pi M{:}obj\ s.\Pi N{:}obj\ t.\Pi O{:}obj\ r.\ =_o\ s\ t\ M\ N \rightarrow =_o\ t\ r\ N\ O \rightarrow =_o\ s\ r\ M\ O \\
c_{abst} & : \Pi s\, t{:}o.(obj\ s \rightarrow obj\ t) \rightarrow obj\ (c_{\rightarrow}\ s\ t) \\
c_{app} & : \Pi s\, t{:}o.\ obj\ (c_{\rightarrow}\ s\ t) \rightarrow obj\ s \rightarrow obj\ t \\
c_{spair} & : \Pi s\, t{:}o.\Pi M{:}obj\ s.\Pi N{:}obj\ t.\ =_o\ s\ t\ M\ N \rightarrow obj\ (c_{\cap}\ s\ t) \\
c_{\mathrm{pr}_1} & : \Pi s\, t{:}o.\Pi M{:}obj\ (c_{\cap}\ s\ t).obj\ s \\
c_{\mathrm{pr}_2} & : \Pi s\, t{:}o.\Pi M{:}obj\ (c_{\cap}\ s\ t).obj\ t \\
c_{\mathrm{in}_1} & : \Pi s\, t{:}o.\Pi M{:}obj\ s.obj\ (c_{\cup}\ s\ t) \\
c_{\mathrm{in}_2} & : \Pi s\, t{:}o.\Pi M{:}obj\ t.obj\ (c_{\cup}\ s\ t) \\
c_{ssum} & : \Pi s\, t\, r{:}o.\Pi X{:}obj\ (c_{\rightarrow}\ s\ r).\Pi Y{:}obj\ (c_{\rightarrow}\ t\ r).obj\ (c_{\cup}\ s\ t) \rightarrow =_o\ (c_{\rightarrow}\ s\ r)\ (c_{\rightarrow}\ t\ r)\ X\ Y \rightarrow obj\ r \\
c_{=abst} & : \Pi s\, t\, s'\, t'{:}o.\Pi M{:}obj\ s \rightarrow obj\ t.\Pi N{:}obj\ s' \rightarrow obj\ t'. \\
 & \quad (\Pi x{:}obj\ s.\Pi y{:}obj\ s'.\ =_o\ s\ s'\ x\ y \rightarrow =_o\ t\ t'\ (M\ x)\ (N\ y)) \rightarrow \\
 & \quad =_o\ (c_{\rightarrow}\ s\ t)(c_{\rightarrow}\ s'\ t')(c_{abst}\ s\ t\ M)(c_{abst}\ s'\ t'\ N) \\
c_{=app} & : \Pi s\, t\, s'\, t'{:}o.\Pi M{:}obj\ (c_{\rightarrow}\ s\ t).\Pi N{:}obj\ s.\Pi M'{:}obj(c_{\rightarrow}\ s'\ t').\Pi N'{:}obj\ s'. \\
 & \quad =_o\ (c_{\rightarrow}\ s\ t)\ (c_{\rightarrow}\ s'\ t')\ M\ M' \rightarrow =_o\ s\ s'\ N\ N' \rightarrow =_o\ t\ t'(c_{app}\ s\ t\ M\ N)\ (c_{app}\ s'\ t'\ M'\ N') \\
c_{=spair} & : \Pi s\, t{:}o.\Pi M{:}obj\ s.\Pi N{:}obj\ t.\Pi Z{:}\ =_o\ s\ t\ M\ N.\ =_o\ (c_{\cap}\ s\ t)\ s\ (c_{spair}\ s\ t\ M\ N\ Z)\ M \\
c_{=\mathrm{pr}_1} & : \Pi s\, t{:}o.\Pi M{:}obj\ (c_{\cap}\ s\ t).\ =_o\ (c_{\cap}\ s\ t)\ s\ M\ (c_{\mathrm{pr}_1}\ s\ t\ M) \\
c_{=\mathrm{pr}_2} & : \Pi s\, t{:}o.\Pi M{:}obj\ (c_{\cap}\ s\ t).\ =_o\ (c_{\cap}\ s\ t)\ t\ M\ (c_{\mathrm{pr}_2}\ s\ t\ M) \\
c_{=\mathrm{in}_1} & : \Pi s\, t{:}o.\Pi M{:}obj\ s.\ =_o\ (c_{\cup}\ s\ t)\ s\ (c_{\mathrm{in}_1}\ s\ t\ M)\ M \\
c_{=\mathrm{in}_2} & : \Pi s\, t{:}o.\Pi M{:}obj\ t.\ =_o\ (c_{\cup}\ s\ t)\ t\ (c_{\mathrm{in}_2}\ s\ t\ M)\ M \\
c_{=ssum} & : \Pi s\, t\, r{:}o.\Pi\, A{:}obj\ (c_{\rightarrow}\ s\ r).\Pi B{:}obj\ (c_{\rightarrow}\ t\ r).\Pi C{:}obj\ (c_{\cup}\ s\ t). \\
 & \quad \Pi Z{:}\ =_o\ (c_{\rightarrow}\ s\ r)\ (c_{\rightarrow}\ t\ r)\ A\ B.\Pi x{:}obj\ s. \\
 & \quad =_o\ s\ (c_{\cup}\ s\ t)\ x\ C \rightarrow =_o\ r\ r\ (c_{app}\ s\ r\ A\ x)\ (c_{ssum}\ s\ t\ r\ A\ B\ C\ Z)
\end{array}
$$

The Δ-calculus in the Δ-framework (in one slide)

$$o \;:\; \mathrm{Type} \qquad c_{\to}, c_{\to_r}, c_{\cap}, c_{\cup} : o \to o \to o$$

$$obj \;:\; o \to \mathrm{Type}$$

$$c_{abst} \;:\; \Pi s\, t{:}o.(obj\ s \to obj\ t) \to_r obj\,(c_{\to}\ s\ t)$$

$$c_{sabst} \;:\; \Pi s\, t{:}o.(obj\ s \to_r obj\ t) \to_r obj\,(c_{\to_r}\ s\ t)$$

$$c_{app} \;:\; \Pi s\, t{:}o.obj\,(c_{\to}\ s\ t) \to_r obj\ s \to obj\ t$$

$$c_{sapp} \;:\; \Pi s\, t{:}o.obj\,(c_{\to_r}\ s\ t) \to_r obj\ s \to_r obj\ t$$

$$c_{\mathrm{pr}_i} \;:\; \Pi s\, t{:}o.obj\,(c_{\cap}\ s\ t) \to_r (obj\, s \cap obj\, t)$$

$$c_{\mathrm{in}_i} \;:\; \Pi s\, t{:}o.(obj\, s \cup obj\, t) \to_r obj\,(c_{\cup}\ s\ t)$$

$$c_{spair} \;:\; \Pi s\, t{:}o.(obj\, s \cap obj\, t) \to_r obj\,(c_{\cap}\ s\ t)$$

$$c_{ssum} \;:\; \Pi s\, t{:}o.obj\,(c_{\cup}\ s\ t) \to_r (obj\, s \cup obj\, t)$$

Ex 1: encoding polymorphic identity in the Δ-framework

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

This derivation is faithfully encoded by the Δ-term

$$\langle \lambda x{:}\sigma.x \,,\, \lambda x{:}\tau.x \rangle$$

and a shallow and compact encoding is

$$c_{spair}\,(c_{\to}\;\sigma\;\sigma)\,(c_{\to}\;\tau\;\tau)\,\langle c_{abst}\;\sigma\;\sigma\;(\lambda x{:}obj\;\sigma.x))\,,\,c_{abst}\;\tau\;\tau\;(\lambda x{:}obj\;\tau.x))\rangle$$

Note that a deep encoding in pure LF would be

$$c_{spair}\,(c_{\to}\;\sigma\;\sigma)\,(c_{\to}\;\tau\;\tau)\,(c_{abst}\;\sigma\;\sigma\;(\lambda x{:}obj\;\sigma.x))\,(c_{abst}\;\tau\;\tau\;(\lambda x{:}obj\;\tau.x))$$
$$(c_{=abst}\;\sigma\;\sigma\;\tau\;\tau\;(\lambda x{:}obj\;\sigma.x)\,(\lambda x{:}obj\;\tau.x)\,(\lambda x{:}obj\;\sigma.\lambda y{:}obj\;\tau.\lambda z{:}=_o\;\sigma\;\tau\;x\;y).z))$$

Ex 2: encoding commutativity of union in the Δ-framework

$$
\dfrac{\dfrac{\dfrac{}{x{:}\sigma\cup\tau, y{:}\sigma \vdash y : \sigma}}{x{:}\sigma\cup\tau, y{:}\sigma \vdash y : \tau\cup\sigma} \quad \dfrac{\dfrac{}{x{:}\sigma\cup\tau, y{:}\tau \vdash y : \tau}}{x{:}\sigma\cup\tau, y{:}\tau \vdash y : \tau\cup\sigma} \quad \dfrac{}{x{:}\sigma\cup\tau \vdash x : \sigma\cup\tau}}{\dfrac{x{:}\sigma\cup\tau \vdash x : \tau\cup\sigma \qquad \wr x \wr \equiv x}{\vdash \lambda^r x.x : (\sigma\cup\tau) \to^r (\tau\cup\sigma)}}
$$

This derivation is faithfully encoded by the Δ-term

$$\lambda^r x{:}\sigma{\cup}\tau.[\lambda y{:}\sigma.\mathrm{in}_2^{\tau}\, y\,,\, \lambda y{:}\tau.\mathrm{in}_1^{\sigma}\;\, y]\, x$$

and a shallow compact encoding in the Δ-framework is

$$c_{sabst}\,(c_\cup\,\sigma\,\tau)\,(c_\cup\,\tau\,\sigma)\,(\lambda^r x{:}obj\,(c_\cup\,\sigma\,\tau).$$

$$[\lambda y{:}obj\,\sigma.c_{\mathrm{in}_i}\,(\mathrm{in}_2^{obj\,\tau}\;\, y)\,,\, \lambda y{:}obj\,\tau.c_{\mathrm{in}_i}\,(\mathrm{in}_1^{obj\,\sigma}\;\, y)]\,(c_{ssum}\,\sigma\,\tau\,x))$$

Source code

- Prototype implementation of a type reconstruction algorithm in ocaml, with a simple CLI REPL
- Standard tools (lex+yacc, de Bruijn indices. . .)
- We use the PTS syntax

```
> Axiom A : Type.
A is assumed.
> Axiom B : forall x : A, Type.
B is assumed.
> Definition foo :=
  fun x : (forall y : A, B y) & A => (proj_l x) (proj_r x).
foo is defined.
> Print foo.
  fun x : (forall y : A, B y) & A => proj_l x proj_r x :
forall x : (forall y : A, B y) & A, B proj_r x
  essence = fun x => x x :
forall x : (forall y : A, B y) & A, B x
```

Agenda

- Adding subtyping to the Δ-framework, with the corresponding algorithm
- Studying the metatheory of the Δ-framework
  - Church-Rosser
  - Subject reduction
  - Strong normalization
  - ...
- Study the impact of proof-functional operators in *refiners*.
  A refiner takes a term with unification meta-variables, and tries to fill or to generate a proof obligation for the meta-variables

$$\langle \Delta_1 \, , ? \rangle$$

- Encoding the full power of Anderson-Belnap Relevant Logic [JSL62] and Routley-Meyer Minimal Relevant Logic B^+ [JPL72]

Thanks and visit

https://github.com/cstolze/Bull

EXTRA SLIDES

Reductions rules of the Δ-calculus

Standard β-reduction

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

$$(\lambda^r x{:}\sigma.\Delta_1)\,\Delta_2 \longrightarrow_\beta \Delta_1\{\Delta_2/x\}$$

Projection rules

$$\mathrm{pr}_1\,\langle\Delta_1\,,\Delta_2\rangle \longrightarrow_{\mathrm{pr}_1} \Delta_1$$

$$\mathrm{pr}_2\,\langle\Delta_1\,,\Delta_2\rangle \longrightarrow_{\mathrm{pr}_2} \Delta_2$$

Injection rules

$$[\Delta_1\,,\Delta_2]\,\mathrm{in}_1^\sigma\,\Delta_3 \longrightarrow_{\mathrm{in}_1} \Delta_1\,\Delta_3$$

$$[\Delta_1\,,\Delta_2]\,\mathrm{in}_2^\sigma\,\Delta_3 \longrightarrow_{\mathrm{in}_2} \Delta_2\,\Delta_3$$

Reductions rules of the Δ-calculus

Standard β-reduction

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

$$(\lambda^r x{:}\sigma.\Delta_1)\,\Delta_2 \longrightarrow_\beta \Delta_1\{\Delta_2/x\}$$

Projection rules

$$\mathrm{pr}_1\,\langle\Delta_1\,,\Delta_2\rangle \longrightarrow_{\mathrm{pr}_1} \Delta_1$$

$$\mathrm{pr}_2\,\langle\Delta_1\,,\Delta_2\rangle \longrightarrow_{\mathrm{pr}_2} \Delta_2$$

Injection rules

$$[\Delta_1\,,\Delta_2]\,\mathrm{in}_1^\sigma\,\Delta_3 \longrightarrow_{\mathrm{in}_1} \Delta_1\,\Delta_3$$

$$[\Delta_1\,,\Delta_2]\,\mathrm{in}_2^\sigma\,\Delta_3 \longrightarrow_{\mathrm{in}_2} \Delta_2\,\Delta_3$$

In a more ML-like syntax, $[\Delta_1\,,\Delta_2]\,\mathrm{in}_i\,\Delta_3$ would have been written:

match $\mathrm{in}_i\,\Delta_3$ with
| in_1^σ x -> $\Delta_1\,x$
| in_2^σ x -> $\Delta_2\,x$

Typing in Δ-calculus

$$\frac{x{:}\sigma \in \Gamma}{\Gamma \vdash x : \sigma}\ (Var)$$

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

$$\frac{\begin{array}{l}\Gamma \vdash \Delta_1 : \sigma \\ \Gamma \vdash \Delta_2 : \tau \quad \wr\Delta_1\wr \equiv \wr\Delta_2\wr\end{array}}{\Gamma \vdash \langle \Delta_1 , \Delta_2 \rangle : \sigma \cap \tau}\ (\cap I) \qquad \frac{\Gamma \vdash \Delta : \sigma_1 \cap \sigma_2 \quad i \in \{1,2\}}{\Gamma \vdash \mathrm{pr}_i\,\Delta : \sigma_i}\ (\cap E_i)$$

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

Subtyping rules (Ξ type 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) $\sigma \leqslant \omega$

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

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

(13) $\omega \leqslant \omega \to \omega$

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

Subtyping rules (Ξ type 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) $\sigma \leqslant \omega$

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

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

(13) $\omega \leqslant \omega \to \omega$

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

- We have defined a functional-style algorithm with exponential complexity
- Deciding subtyping is easy when types are in normal form
- Well established domain of set constraints (see eg. Aiken)

Subtyping algorithm

- Syntax of normal forms

$$
\begin{array}{rcl}
A & ::= & \omega \mid \phi \mid (A \cap \ldots \cap A) \to (A \cup \ldots \cup A) \\
CNF & ::= & (A \cup \ldots \cup A) \cap \ldots \cap (A \cup \ldots \cup A) \\
DNF & ::= & (A \cap \ldots \cap A) \cup \ldots \cup (A \cap \ldots \cap A)
\end{array}
$$

- Sketch of the algorithm
  - Any judgement $\sigma \leqslant \tau$ can be reduced to a judgement whose syntax is $DNF \leqslant CNF$
  - A judgement whose syntax is $DNF \leqslant CNF$ can be reduced to multiple judgements whose syntax is $A \leqslant A$
  - A judgement whose syntax is $A \leqslant A$ can be easily decided ($\phi \leqslant \omega$, $\omega \nleqslant \phi$, $\phi \leqslant \phi'$ iff $\phi \equiv \phi'$, $\ldots$)

On relevant operators and relevant logics *spoiler*

- Meyer-Routley B^+ relevant logic (with the relevant implication $\supset_r$ connective) forces the proof to use all the hypothesis, therefore making the proof relevant
- ... a proof $\mathcal{D}$ for $\phi \supset_r \psi$ is also proof for $\phi \supset \psi$ whose realizer is the identity function
- Relevant implication $\supset_r$ can be intended as another proof-functional connective
- The typing rule to be added to the Delta-calculus is

$$\frac{\Gamma, x{:}\sigma \vdash \Delta : \tau \quad \wr\Delta\wr \equiv x}{\Gamma \vdash \lambda^r x{:}\sigma.\Delta : \sigma \rightarrow_r \tau} \ (\rightarrow_r I)$$

- As example, in the Delta-calculus with relevant arrow we can prove

$$\phi \cap \psi \supset_r \psi \cap \phi$$

$$\phi \cup \psi \supset_r \psi \cup \phi$$

Example: relevant logic B^+

$$\dfrac{\dfrac{\dfrac{}{x{:}(\sigma \to^r \tau) \cap \sigma \vdash_\Sigma x : (\sigma \to^r \tau) \cap \sigma}}{x{:}(\sigma \to^r \tau) \cap \sigma \vdash_\Sigma \mathrm{pr}_1\, x : \sigma \to^r \tau} \quad \dfrac{\dfrac{}{x{:}(\sigma \to^r \tau) \cap \sigma \cap \sigma \vdash_\Sigma x : (\sigma \to^r \tau) \cap \sigma}}{x{:}(\sigma \to^r \tau) \cap \sigma \vdash_\Sigma \mathrm{pr}_2\, x : \sigma}}{x{:}(\sigma \to^r \tau) \cap \sigma \vdash_\Sigma (\mathrm{pr}_1\, x)\,(\mathrm{pr}_2\, x) : \tau \quad \wr(\mathrm{pr}_1\, x)\,(\mathrm{pr}_2\, x)\wr \equiv x}$$

$$\overline{\vdash_\Sigma \lambda^r x{:}(\sigma \to^r \tau) \cap \sigma.(\mathrm{pr}_1\, x)\,(\mathrm{pr}_2\, x) : ((\sigma \to^r \tau) \cap \sigma) \to^r \tau}$$

The relevant arrow forces us to use all the hypotheses. The proof is therefore relevant.

However, the affixing property

$$(\sigma \to^r \tau) \to^r ((\rho \to^r \sigma) \to^r (\rho \to^r \tau))$$

of the relevant logic B^+ is not encodable. We could try

$$\lambda^r f{:}(\sigma \to^r \tau).\lambda^r g{:}\rho \to^r \sigma.\lambda^r x{:}\rho.f\,(g\,x)$$

However, the essence of $\lambda^r g{:}\rho \to^r \sigma.\lambda^r x{:}\rho.f\,(g\,x)$ is $\lambda g.\lambda x.x$, which is not the identity.

Pierce example

- Pierce example:

$$x\,(\underbrace{(\mathsf{I}\,y)\,z}_{\beta})\,(\underbrace{(\mathsf{I}\,y)}_{\beta}\,z)\ \begin{array}{l} \nearrow^{\beta}\ x\,(y\,z)\,\overbrace{((\mathsf{I}\,y)\,z)}^{\beta}\ \searrow_{\beta} \\ \searrow_{\beta}\ x\,\underbrace{((\mathsf{I}\,y)\,z)}_{\beta}\,(y\,z)\ \nearrow^{\beta} \end{array}\ x\,(y\,z)\,(y\,z)$$

- In the context where $x{:}(\sigma_1 \to \sigma_1 \to \tau) \cap (\sigma_2 \to \sigma_2 \to \tau), y{:}\rho \to \sigma_1 \cup \sigma_2, z{:}\rho$ the corresponding Δ-term is

$$\Delta \triangleq [\underbrace{(\lambda v{:}\sigma_1.(\mathrm{pr}_1\, x)\, v\, v)}_{\Delta_1}, \underbrace{(\lambda v{:}\sigma_2.(\mathrm{pr}_2\, x)\, v\, v)}_{\Delta_2}]\,(\underbrace{(\lambda v{:}\rho \to \sigma_1 \cup \sigma_2.v)}_{\Delta_3}\ y\, z)$$

- The only applicable parallel redex is $\Delta_3\, y$ and that gives

$$[\Delta_1\,, \Delta_2]\,(y\,z)$$

Compact encoding of [BDdL] in the extended LF

- Because of the *shallow* encoding, source language and target language are "mostly" overlapped

$$
\begin{array}{rcl}
o & : & \text{Type} \qquad c_{\to}, c_{\to_r}, c_{\cap}, c_{\cup} : o \to o \to o \\
obj & : & o \to \text{Type} \\
c_{abst} & : & \Pi s\, t{:}o.(obj\ s \to obj\ t) \to_r obj(c_{\to}\ s\ t) \\
c_{sabst} & : & \Pi s\, t{:}o.(obj\ s \to_r obj\ t) \to_r obj(c_{\to_r}\ s\ t) \\
c_{app} & : & \Pi s\, t{:}o.obj(c_{\to}\ s\ t) \to_r obj\ s \to obj\ t \\
c_{sapp} & : & \Pi s\, t{:}o.obj(c_{\to_r}\ s\ t) \to_r obj\ s \to_r obj\ t \\
c_{pr_i} & : & \Pi s\, t{:}o.obj\,(c_{\cap}\ s\ t) \to_r (obj\ s) \cap (obj\ t) \\
c_{in_i} & : & \Pi s\, t{:}o.(obj\ s) \cup (obj\ t) \to_r obj\,(c_{\cup}\ s\ t) \\
c_{spair} & : & \Pi s\, t{:}o.(obj\ s) \cap (obj\ t) \to_r obj\,(c_{\cap}\ s\ t) \\
c_{ssum} & : & \Pi s\, t{:}o.obj\,(c_{\cup}\ s\ t) \to_r (obj\ s) \cup (obj\ t)
\end{array}
$$

- By extending the logical framework, we eliminate the need of encoding the essence side conditions via many lines of pure LF code (see Honsell LF encoding)

Mints realizers

- First-order predicate NJ logic with subject beta-conversion

$$
\begin{array}{rcl}
r_{\phi}[x] & \equiv & \mathbf{P}_{\phi}(x) \\
r_{\sigma_1 \to \sigma_2}[x] & \equiv & \forall y . r_{\sigma_1}[y] \supset r_{\sigma_2}[x\, y] \\
r_{\sigma_1 \cap \sigma_2}[x] & \equiv & r_{\sigma_1}[x] \wedge r_{\sigma_2}[x] \\
r_{\sigma_1 \cup \sigma_2}[x] & \equiv & r_{\sigma_1}[x] \vee r_{\sigma_2}[x]
\end{array}
$$

- it is more stronger than the Barbanera-Dezani-de'Liguoro type assignement system

Properties of the Δ-calculus

- Judgments fully encode pure type assignment derivations $\mathcal{D}$ i.e.

$$B \vdash \Delta : \sigma \qquad \text{iff} \qquad \mathcal{D} : B \vdash M : \sigma$$

Example: the Δ-term $\langle \lambda x{:}\sigma.x \,, \lambda x{:}\tau.x \rangle$ of type $\sigma \to \sigma \cap \tau \to \tau$ encodes the type assignment derivation

$$\dfrac{\dfrac{\overline{x{:}\sigma \vdash x : \sigma}}{\vdash I : \sigma \to \sigma} \quad \dfrac{\overline{x{:}\tau \vdash x : \tau}}{\vdash I : \tau \to \tau}}{I : \sigma \to \sigma \cap \tau \to \tau}$$

- Subject reduction for parallel reduction $\to_{\parallel}$
- Strong normalization of ω-free typable terms
- Unicity of typing
- Decidability of type checking and type reconstruction

Splash

`Help.`	
`List of commands:`	
`Help.`	show this list of commands
`Load file.`	for loading a script file
`Axiom term : type.`	define a constant or an axiom
`Definition name [: type] := term.`	define a term
`Print name.`	print the definition of name
`Printall.`	print all the signature (axioms and definitions)
`Compute name.`	normalize name and print the result
`Quit.`	quit

Subtyping

- Many of the basic properties of intersection and unions can be derived
- However, distributivity of intersection over union (and *vice versa*) is not derivable

$$x{:}\sigma \cap (\tau \cup \rho) \nvdash x : (\sigma \cap \tau) \cup (\sigma \cap \rho)$$

- Therefore, we need a subtyping axiom for distributivity

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

More examples (opt)

- Union commutativity

$$\dfrac{\dfrac{}{x{:}\sigma \cup \tau, y{:}\sigma \vdash y : \sigma}}{x{:}\sigma \cup \tau, y{:}\sigma \vdash y : \tau \cup \sigma} \quad \dfrac{\dfrac{}{x{:}\sigma \cup \tau, y{:}\tau \vdash y : \tau}}{x{:}\sigma \cup \tau, y{:}\tau \vdash y : \tau \cup \sigma} \quad \dfrac{}{x{:}\sigma \cup \tau \vdash x : \sigma \cup \tau}$$
$$\overline{\qquad x{:}\sigma \cup \tau \vdash x : \tau \cup \sigma \qquad}$$

More examples (opt)

- Union commutativity

$$\dfrac{\dfrac{\overline{x{:}\sigma \cup \tau, y{:}\sigma \vdash y : \sigma}}{x{:}\sigma \cup \tau, y{:}\sigma \vdash y : \tau \cup \sigma} \quad \dfrac{\overline{x{:}\sigma \cup \tau, y{:}\tau \vdash y : \tau}}{x{:}\sigma \cup \tau, y{:}\tau \vdash y : \tau \cup \sigma} \quad \overline{x{:}\sigma \cup \tau \vdash x : \sigma \cup \tau}}{x{:}\sigma \cup \tau \vdash x : \tau \cup \sigma}$$

- Intersection commutativity

$$\dfrac{\dfrac{\overline{x{:}\sigma \cap \tau \vdash x : \sigma \cap \tau}}{x{:}\sigma \cap \tau \vdash x : \tau} \quad \dfrac{\overline{x{:}\sigma \cap \tau \vdash x : \sigma \cap \tau}}{x{:}\sigma \cap \tau \vdash x : \sigma}}{x{:}\sigma \cap \tau \vdash x : \tau \cap \sigma}$$

More examples (opt)

- Union commutativity

$$\dfrac{\dfrac{\overline{x{:}\sigma\cup\tau, y{:}\sigma \vdash y : \sigma}}{x{:}\sigma\cup\tau, y{:}\sigma \vdash y : \tau\cup\sigma} \quad \dfrac{\overline{x{:}\sigma\cup\tau, y{:}\tau \vdash y : \tau}}{x{:}\sigma\cup\tau, y{:}\tau \vdash y : \tau\cup\sigma} \quad \overline{x{:}\sigma\cup\tau \vdash x : \sigma\cup\tau}}{x{:}\sigma\cup\tau \vdash x : \tau\cup\sigma}$$

- Intersection commutativity

$$\dfrac{\dfrac{\overline{x{:}\sigma\cap\tau \vdash x : \sigma\cap\tau}}{x{:}\sigma\cap\tau \vdash x : \tau} \quad \dfrac{\overline{x{:}\sigma\cap\tau \vdash x : \sigma\cap\tau}}{x{:}\sigma\cap\tau \vdash x : \sigma}}{x{:}\sigma\cap\tau \vdash x : \tau\cap\sigma}$$

- Self-application

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

Reductions in Δ-calculus

- $\langle(\lambda x{:}\sigma.x)\,c\,,(\lambda x{:}\sigma.x)\,c\rangle$ is typable

$$\frac{c{:}\sigma \vdash (\lambda x{:}\sigma.x)\,c : \sigma \quad c{:}\sigma \vdash (\lambda x{:}\sigma.x)\,c : \sigma \quad {\color{blue}(\lambda x.x)\,c \equiv (\lambda x.x)\,c}}{c{:}\sigma \vdash \langle(\lambda x{:}\sigma.x)\,c\,,(\lambda x{:}\sigma.x)\,c\rangle : \sigma \cap \sigma}$$

- $\langle c\,,(\lambda x{:}\sigma.x)\,c\rangle$ is not typable

$$\frac{c{:}\sigma \vdash c : \sigma \quad c{:}\sigma \vdash (\lambda x{:}\sigma.x)\,c : \sigma \quad {\color{red}c \not\equiv (\lambda x.x)\,c}}{c{:}\sigma \nvdash \langle c\,,(\lambda x{:}\sigma.x)\,c\rangle : \sigma \cap \sigma}$$