Realizability and parametricity in pure type systems

Jean-Philippe Bernardy – Chalmers university
Marc Lasson – École Normale Supérieure de Lyon

February 15, 2011

Parametric polymorphism

```
let rec f = function
   | [] -> 1
   | hd::tl -> 2 * (f tl)

val f : ∀α, α list → int
```

Parametricity polymorphism: parametric types behave uniformly over abstracted types.

If $\vdash_{\mathcal{F}} f : \forall\alpha, \alpha\texttt{list} \rightarrow \texttt{int}$ and $|l| = |l'|$ then $f\ l = f\ l'$.

Parametricity relations

- Tool introduced by Reynolds to study polymorphism.

Parametricity relations

- Tool introduced by Reynolds to study polymorphism.

In System F

We define a relation $s \sim_\tau t$ by induction on τ

$$t_1 \sim_{\sigma\to\tau} t_2 \equiv \forall x_1\, x_2. x_1 \sim_\sigma x_2 \to (t_1\, x_1) \sim_\tau (t_2\, x_2)$$

$$t_1 \sim_\alpha t_2 \equiv R_\alpha\, t_1\, t_2$$

$$t_1 \sim_{\forall\alpha,\tau} t_2 \equiv \forall R_\alpha . t_1 \sim_\tau t_2$$

Two related functions map related inputs to related outputs.

Parametricity relations

- Tool introduced by Reynolds to study polymorphism.

In System F

We define a relation $s \sim_\tau t$ by induction on τ

$$t_1 \sim_{\sigma \to \tau} t_2 \equiv \forall x_1\, x_2 . x_1 \sim_\sigma x_2 \to (t_1\, x_1) \sim_\tau (t_2\, x_2)$$

$$t_1 \sim_\alpha t_2 \equiv R_\alpha\, t_1\, t_2$$

$$t_1 \sim_{\forall \alpha, \tau} t_2 \equiv \forall R_\alpha . t_1 \sim_\tau t_2$$

Two related functions map related inputs to related outputs.

Abstraction theorem

If $\vdash t : \tau$ then we can prove that $t \sim_\tau t$.

Example

$\forall \alpha . \alpha \to \alpha$

$$f \sim_{\forall \alpha . \alpha \to \alpha} g$$

$$\equiv$$

$$\forall R . \forall xy . xRy \to (f\ x) R (g\ y)$$

Example

$\forall \alpha . \alpha \to \alpha$

$$f \sim_{\forall \alpha . \alpha \to \alpha} g$$

$$\equiv$$

$$\forall R . \forall x y . x R y \to (f\ x) R (g\ y)$$

$\forall \alpha \beta . \alpha \to \beta \to \alpha$

$$f \sim_{\forall \alpha \beta . \alpha \to \beta \to \alpha} g$$

$$\equiv$$

$$\forall R_1 R_2 . \forall x_1 y_1 . x_1 R_1 y_1 \to \forall x_2 y_2 . x_2 R_2 y_2 \to (f\ x_1\ x_2) R_1 (g\ y_1\ y_2)$$

Parametricity – Abstraction theorem

Abstraction theorem

If $\vdash t : \tau$ then we can prove that $t \sim_\tau t$.

Application : Theorems for free!

- Let t be such that

$$\vdash t : \forall \alpha . \alpha \to \alpha$$

Parametricity – Abstraction theorem

Abstraction theorem

If $\vdash t : \tau$ then we can prove that $t \sim_\tau t$.

Application : Theorems for free!

- Let t be such that

$$\vdash t : \forall\alpha.\alpha \to \alpha$$

- By the abstraction theorem, you obtain

$$t \sim_{\forall\alpha.\alpha\to\alpha} t$$

Parametricity – Abstraction theorem

Abstraction theorem

If $\vdash t : \tau$ then we can prove that $t \sim_\tau t$.

Application : Theorems for free!

- Let t be such that

$$\vdash t : \forall \alpha . \alpha \to \alpha$$

- By the abstraction theorem, you obtain

$$t \sim_{\forall \alpha . \alpha \to \alpha} t$$

- By unfolding the definition of $\sim_{\forall \alpha . \alpha \to \alpha}$,

$$\forall R^{\alpha, \beta} \quad x : \alpha \quad y : \beta . x R y \to (t_\alpha\, x) R (t_\beta\, y)$$

Parametricity – Abstraction theorem

Abstraction theorem

If $\vdash t : \tau$ then we can prove that $t \sim_\tau t$.

Application : Theorems for free!

- Let t be such that

$$\vdash t : \forall\alpha.\alpha \to \alpha$$

- By the abstraction theorem, you obtain

$$t \sim_{\forall\alpha.\alpha\to\alpha} t$$

- By unfolding the definition of $\sim_{\forall\alpha.\alpha\to\alpha}$,

$$\forall\, R^{\alpha,\beta} \quad x : \alpha \quad y : \beta.xRy \to (t_\alpha\, x)R(t_\beta\, y)$$

- For all $g : \alpha \to \beta$, if you take to be $R\,x\,y \Leftrightarrow (g\,x) = y$, you have

$$\forall g : \alpha \to \beta.\forall x : \alpha.g\,(t_\alpha\, x) = t_\beta\,(g\,x)$$

Parametricity – Abstraction theorem

Abstraction theorem

If $\vdash t : \tau$ then we can prove that $t \sim_\tau t$.

Application : Theorems for free!

- Let t be such that
$$\vdash t : \forall \alpha . \alpha \to \alpha$$
- By the abstraction theorem, you obtain
$$t \sim_{\forall \alpha . \alpha \to \alpha} t$$
- By unfolding the definition of $\sim_{\forall \alpha . \alpha \to \alpha}$,
$$\forall R^{\alpha,\beta} \quad x : \alpha \quad y : \beta . xRy \to (t_\alpha\, x) R (t_\beta\, y)$$
- For all $g : \alpha \to \beta$, if you take to be $R\, x\, y \Leftrightarrow (g\, x) = y$, you have
$$\forall g : \alpha \to \beta . \forall x : \alpha . g\, (t_\alpha\, x) = t_\beta\, (g\, x)$$
- By extensionality, it's equivalent to
$$\forall g : \alpha \to \beta . g \circ t_\alpha = t_\beta \circ g$$

Parametricity – Abstraction theorem

Abstraction theorem

If $\vdash t : \tau$ then we can prove that $t \sim_\tau t$.

Application : Theorems for free!

- Let t be such that
$$\vdash t : \forall\alpha.\alpha \to \alpha$$
- By the abstraction theorem, you obtain
$$t \sim_{\forall\alpha.\alpha\to\alpha} t$$
- By unfolding the definition of $\sim_{\forall\alpha.\alpha\to\alpha}$,
$$\forall\, R^{\alpha,\beta} \quad x : \alpha \quad y : \beta . xRy \to (t_\alpha\, x) R (t_\beta\, y)$$
- For all $g : \alpha \to \beta$, if you take to be $R\, x\, y \Leftrightarrow (g\, x) = y$, you have
$$\forall g : \alpha \to \beta . \forall x : \alpha . g\,(t_\alpha\, x) = t_\beta\,(g\, x)$$
- By extensionality, it's equivalent to
$$\forall g : \alpha \to \beta . g \circ t_\alpha = t_\beta \circ g$$
- Which is equivalent to the fact that t is the identity function

Realizability

Slogan

Specifying programs with formulas
or
giving computational content to formula.

Realizability

Slogan

Specifying programs with formulas

or

giving computational content to formula.

We define "p realizes a formula F" ($p \Vdash F$) by induction on F.

Key case of the definition

$$t \Vdash P \to Q \equiv \forall x. x \Vdash P \to (t\, x) \Vdash Q$$

Realizability

Slogan

Specifying programs with formulas

or

giving computational content to formula.

We define "p realizes a formula F" ($p \Vdash F$) by induction on F.

Key case of the definition

$$t \Vdash P \to Q \equiv \forall x. x \Vdash P \to (t\, x) \Vdash Q$$

Adequacy theorem

If there exists a proof π of P, then there exists a program p_π and a proof π' of $p_\pi \Vdash P$.

Realizability – Applications

- Proving that axioms (e.g. excluded middle) are not derivable

Realizability – Applications

- Proving that axioms (e.g. excluded middle) are not derivable
- Studying programs extracted from proofs:

Realizability – Applications

- Proving that axioms (e.g. excluded middle) are not derivable
- Studying programs extracted from proofs:

Existence property

If $\forall x \exists y, \varphi(x, y)$ is a theorem, then there exists a program f such that $\forall x, \varphi(x, f(x))$.

Realizability – Applications

- Proving that axioms (e.g. excluded middle) are not derivable
- Studying programs extracted from proofs:

Existence property

If $\forall x \exists y, \varphi(x, y)$ is a theorem, then there exists a program f such that $\forall x, \varphi(x, f(x))$.

Representation theorem

Functions definable in system F are exactly those provably total in second-order arithmetic.

Pure type systems – Generalities

- A family of λ-calculi where types and terms are unified
- Provide a framework for studying dependent types
- Contains many famous type-systems:
  - simply typed λ-calculus,
  - Girard and Reynolds polymorphic λ-calculus (system F),
  - Huet-Coquand's Calculus Of Constructions ...
- It even contains inconsistent calculus ($\texttt{Type} : \texttt{Type}$)

- A PTS P is defined by a specification $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ where
  - $\mathcal{S}$ is a set of sorts,
  - $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ a set of axioms,
  - $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ a set of rules.
- Typing judgement $\Gamma \vdash_P A : B$ of the PTS $P = (\mathcal{S}, \mathcal{A}, \mathcal{R})$.

Pure type systems – Terms and typing rules

Terms

$$A, B \quad := \quad s \mid x \mid (A\,B) \mid \lambda x : A.B \mid \forall x : A.B \quad \text{(with } s \in \mathcal{S}\text{)}$$

Pure type systems – Terms and typing rules

Terms

$$A, B \quad := \quad s \mid x \mid (A\,B) \mid \lambda x : A.B \mid \forall x : A.B \quad \text{(with } s \in \mathcal{S}\text{)}$$

$A \to B$ is a notation for $\forall x : A.B$ with $x \notin B$

Pure type systems – Terms and typing rules

Terms

$$A, B \quad := \quad s \mid x \mid (A\,B) \mid \lambda x : A.B \mid \forall x : A.B \quad (\text{with } s \in \mathcal{S})$$

$A \to B$ is a notation for $\forall x : A.B$ with $x \notin B$

$$\textsc{Axiom}\ \frac{}{\vdash s_1 : s_2}\ (s_1, s_2) \in \mathcal{A}$$

$$\textsc{Abstraction}\ \frac{\Gamma, x : A \vdash C : B \qquad \Gamma \vdash (\forall x : A.B) : s}{\Gamma \vdash (\lambda x : A.C) : (\forall x : A.B)}$$

$$\textsc{Product}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\forall x : A.B) : s_3}\ (s_1, s_2, s_3) \in \mathcal{R}$$

Pure type systems – Terms and typing rules

Terms

$$A, B \quad := \quad s \mid x \mid (A\,B) \mid \lambda x : A.B \mid \forall x : A.B \quad (\text{with } s \in \mathcal{S})$$

$A \to B$ is a notation for $\forall x : A.B$ with $x \notin B$

$$\textsc{Axiom}\ \frac{}{\vdash s_1 : s_2}\ (s_1, s_2) \in \mathcal{A}$$

$$\textsc{Abstraction}\ \frac{\Gamma, x : A \vdash C : B \qquad \Gamma \vdash (\forall x : A.B) : s}{\Gamma \vdash (\lambda x : A.C) : (\forall x : A.B)}$$

$$\textsc{Product}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\forall x : A.B) : s_3}\ (s_1, s_2, s_3) \in \mathcal{R}$$

+ Application + Start + Weakening

System F

System F

The PTS F has the following specification

$$\mathcal{S}_F = \{\star, \Box\} \qquad \mathcal{A}_F = \{(\star, \Box)\} \qquad \mathcal{R}_F = \{(\star, \star, \star), (\Box, \star, \star)\}$$

System F

System F

The PTS F has the following specification

$$\mathcal{S}_F = \{\star, \Box\} \qquad \mathcal{A}_F = \{(\star, \Box)\} \qquad \mathcal{R}_F = \{(\star, \star, \star), (\Box, \star, \star)\}$$

Only two kinds of product :

- Arrow type $(\sigma \to \tau)$: $(\star, \star, \star)$
- Type quantification $(\forall \alpha, \tau)$: $(\Box, \star, \star)$

$$\Gamma \vdash t : \tau : \star$$

System F

System F

The PTS F has the following specification

$\mathcal{S}_F = \{\star, \Box\}$ $\mathcal{A}_F = \{(\star, \Box)\}$ $\mathcal{R}_F = \{(\star,\star,\star), (\Box,\star,\star)\}$

Only two kinds of product :

- Arrow type $(\sigma \to \tau)$: $(\star,\star,\star)$
- Type quantification $(\forall\alpha, \tau)$: $(\Box,\star,\star)$

$$\Gamma \vdash t : \tau : \star$$

$$\frac{\Gamma \vdash \sigma : \star \qquad \Gamma, x : \sigma \vdash \tau : \star}{\Gamma \vdash \forall x : \sigma.\tau : \star} (\star,\star,\star) \in \mathcal{R}_F$$

- We can prove that $\Gamma \vdash \tau : \star$ and $\Gamma \vdash x : \sigma : \star$ then $x \notin \tau$. Therefore $\forall x : \sigma.\tau$ can always be written $\sigma \to \tau$.

System F

System F

The PTS F has the following specification

$$\mathcal{S}_F = \{\star, \square\} \qquad \mathcal{A}_F = \{(\star, \square)\} \qquad \mathcal{R}_F = \{(\star, \star, \star), (\square, \star, \star)\}$$

Only two kinds of product :

- Arrow type $(\sigma \to \tau)$: $(\star, \star, \star)$
- Type quantification $(\forall \alpha, \tau)$: $(\square, \star, \star)$

$$\Gamma \vdash t : \tau : \star$$

$$\frac{\Gamma \vdash \sigma : \star \qquad \Gamma \qquad \vdash \tau : \star}{\Gamma \vdash \sigma \to \tau : \star} (\star, \star, \star) \in \mathcal{R}_F$$

- We can prove that $\Gamma \vdash \tau : \star$ and $\Gamma \vdash x : \sigma : \star$ then $x \notin \tau$. Therefore $\forall x : \sigma . \tau$ can always be written $\sigma \to \tau$.

System F

System F

The PTS F has the following specification

$$\mathcal{S}_F = \{\star, \square\} \qquad \mathcal{A}_F = \{(\star, \square)\} \qquad \mathcal{R}_F = \{(\star,\star,\star), (\square,\star,\star)\}$$

Only two kinds of product :

- Arrow type $(\sigma \to \tau)$: $(\star,\star,\star)$
- Type quantification $(\forall \alpha, \tau)$: $(\square,\star,\star)$

$$\Gamma \vdash t : \tau : \star$$

$$\frac{\Gamma \vdash \sigma : \star \qquad \Gamma \qquad \vdash \tau : \star}{\Gamma \vdash \sigma \to \tau : \star} \, (\star,\star,\star) \in \mathcal{R}_F$$

- We can prove that $\Gamma \vdash \tau : \star$ and $\Gamma \vdash x : \sigma : \star$ then $x \notin \tau$. Therefore $\forall x : \sigma.\tau$ can always be written $\sigma \to \tau$.
- We can also prove that inhabitants of $\star$ are either :
  $$\alpha,\ \sigma \to \tau \text{ or } \forall \alpha : \star.\tau.$$

System F – Examples

Example

- $\mathsf{Nat} \equiv \forall\alpha : \star.(\alpha \to \alpha) \to (\alpha \to \alpha)$
- $0 \equiv \lambda(\alpha : \star)(f : \alpha \to \alpha)(x : \alpha).x$
- $\mathsf{Succ} \equiv \lambda(n : \mathsf{Nat})(\alpha : \star)(f : \alpha \to \alpha)(x : \alpha).f\,(n\,\alpha\,f\,x)$

System F – Examples

Example

- $\mathsf{Nat} \equiv \forall\alpha : \star.(\alpha \to \alpha) \to (\alpha \to \alpha)$
- $0 \equiv \lambda(\alpha : \star)(f : \alpha \to \alpha)(x : \alpha).x$
- $\mathsf{Succ} \equiv \lambda(n : \mathsf{Nat})(\alpha : \star)(f : \alpha \to \alpha)(x : \alpha).f\,(n\,\alpha\,f\,x)$

- $\vdash \mathsf{Nat} : \star$

System F – Examples

Example

- $\mathsf{Nat} \equiv \forall\alpha : \star.(\alpha \to \alpha) \to (\alpha \to \alpha)$
- $0 \equiv \lambda(\alpha : \star)(f : \alpha \to \alpha)(x : \alpha).x$
- $\mathsf{Succ} \equiv \lambda(n : \mathsf{Nat})(\alpha : \star)(f : \alpha \to \alpha)(x : \alpha).f\,(n\,\alpha\,f\,x)$

- $\vdash \mathsf{Nat} : \star$
- $\vdash 0 : \mathsf{Nat}$

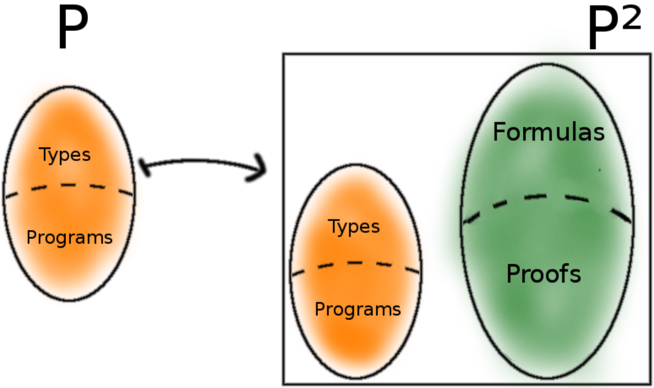
System F – Examples

Example

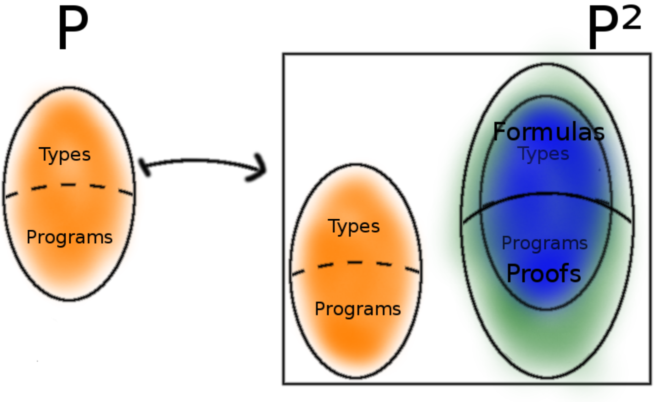
- $\text{Nat} \equiv \forall \alpha : \star.(\alpha \to \alpha) \to (\alpha \to \alpha)$
- $0 \equiv \lambda(\alpha : \star)(f : \alpha \to \alpha)(x : \alpha).x$
- $\text{Succ} \equiv \lambda(n : \text{Nat})(\alpha : \star)(f : \alpha \to \alpha)(x : \alpha).f\,(n\,\alpha\,f\,x)$

- $\vdash \text{Nat} : \star$
- $\vdash 0 : \text{Nat}$
- $\vdash \text{Succ} : \text{Nat} \to \text{Nat}$

From P to P^2 – From realizers to logic



From P to P^2 – From realizers to logic



From P to P^2 – Definitions

Given a PTS $P = (\mathcal{S}, \mathcal{A}, \mathcal{R})$, we define $P^2 = (\mathcal{S}^2, \mathcal{A}^2, \mathcal{R}^2)$ by

$$\begin{array}{rcl}
\mathcal{S}^2 & = & \mathcal{S} \cup \{\, \lceil \mathsf{s} \rceil \;\mid s \in \mathcal{S}\} \\
\mathcal{A}^2 & = & \mathcal{A} \cup \{\, (\lceil \mathsf{s}_1 \rceil,\, \lceil \mathsf{s}_2 \rceil) \;\mid (s_1, s_2) \in \mathcal{A}\} \\
\mathcal{R}^2 & = & \mathcal{R} \cup \{\, (\lceil \mathsf{s}_1 \rceil,\, \lceil \mathsf{s}_2 \rceil,\, \lceil \mathsf{s}_3 \rceil)\,,\ (\mathsf{s}_1, \lceil \mathsf{s}_3 \rceil, \lceil \mathsf{s}_3 \rceil) \;\mid (s_1, s_2, s_3) \in \mathcal{R}\} \\
 & & \cup \{\, (\mathsf{s}_1,\, \lceil \mathsf{s}_2 \rceil, \lceil \mathsf{s}_2 \rceil) \;\mid (s_1, s_2) \in \mathcal{A}\}
\end{array}$$

From P to P^2 – Definitions

Given a PTS $P = (\mathcal{S}, \mathcal{A}, \mathcal{R})$, we define $P^2 = (\mathcal{S}^2, \mathcal{A}^2, \mathcal{R}^2)$ by

$$
\begin{array}{rcl}
\mathcal{S}^2 & = & \mathcal{S} \cup \{ \lceil \mathsf{s} \rceil \mid s \in \mathcal{S}\} \\
\mathcal{A}^2 & = & \mathcal{A} \cup \{ (\lceil \mathsf{s}_1 \rceil, \lceil \mathsf{s}_2 \rceil) \mid (s_1, s_2) \in \mathcal{A}\} \\
\mathcal{R}^2 & = & \mathcal{R} \cup \{ (\lceil \mathsf{s}_1 \rceil, \lceil \mathsf{s}_2 \rceil, \lceil \mathsf{s}_3 \rceil), (\mathsf{s}_1, \lceil \mathsf{s}_3 \rceil, \lceil \mathsf{s}_3 \rceil) \mid (s_1, s_2, s_3) \in \mathcal{R}\} \\
 & & \cup \{ (\mathsf{s}_1, \lceil \mathsf{s}_2 \rceil, \lceil \mathsf{s}_2 \rceil) \mid (s_1, s_2) \in \mathcal{A}\}
\end{array}
$$

- For each sort s we add a copy $\lceil s \rceil$,
- For each axiom (s_1, s_2) we add the axiom $(\lceil s_1 \rceil, \lceil s_2 \rceil)$.
- Beside the original rules, we allow three new quantifications :
  1. We lift constructs of realizer at the level of the logic,

From P to P^2 – Definitions

Given a PTS $P = (\mathcal{S}, \mathcal{A}, \mathcal{R})$, we define $P^2 = (\mathcal{S}^2, \mathcal{A}^2, \mathcal{R}^2)$ by

$$
\begin{array}{rcl}
\mathcal{S}^2 & = & \mathcal{S} \cup \{ \lceil s \rceil \mid s \in \mathcal{S} \} \\
\mathcal{A}^2 & = & \mathcal{A} \cup \{ (\lceil s_1 \rceil, \lceil s_2 \rceil) \mid (s_1, s_2) \in \mathcal{A} \} \\
\mathcal{R}^2 & = & \mathcal{R} \cup \{ (\lceil s_1 \rceil, \lceil s_2 \rceil, \lceil s_3 \rceil), (s_1, \lceil s_3 \rceil, \lceil s_3 \rceil) \mid (s_1, s_2, s_3) \in \mathcal{R} \} \\
 & & \cup \{ (s_1, \lceil s_2 \rceil, \lceil s_2 \rceil) \mid (s_1, s_2) \in \mathcal{A} \}
\end{array}
$$

- For each sort s we add a copy $\lceil s \rceil$,
- For each axiom (s_1, s_2) we add the axiom $(\lceil s_1 \rceil, \lceil s_2 \rceil)$.
- Beside the original rules, we allow three new quantifications :
  1. We lift constructs of realizer at the level of the logic,
  2. We allow quantification over programs,

From P to P^2 – Definitions

Given a PTS $P = (\mathcal{S}, \mathcal{A}, \mathcal{R})$, we define $P^2 = (\mathcal{S}^2, \mathcal{A}^2, \mathcal{R}^2)$ by

$$\begin{array}{rcl} \mathcal{S}^2 & = & \mathcal{S} \cup \{ \lceil \mathsf{s} \rceil \mid s \in \mathcal{S} \} \\ \mathcal{A}^2 & = & \mathcal{A} \cup \{ (\lceil \mathsf{s}_1 \rceil, \lceil \mathsf{s}_2 \rceil) \mid (s_1, s_2) \in \mathcal{A} \} \\ \mathcal{R}^2 & = & \mathcal{R} \cup \{ (\lceil \mathsf{s}_1 \rceil, \lceil \mathsf{s}_2 \rceil, \lceil \mathsf{s}_3 \rceil), (\mathsf{s}_1, \lceil \mathsf{s}_3 \rceil, \lceil \mathsf{s}_3 \rceil) \mid (s_1, s_2, s_3) \in \mathcal{R} \} \\ & & \cup \{ (\mathsf{s}_1, \lceil \mathsf{s}_2 \rceil, \lceil \mathsf{s}_2 \rceil) \mid (s_1, s_2) \in \mathcal{A} \} \end{array}$$

- For each sort s we add a copy $\lceil s \rceil$,
- For each axiom (s_1, s_2) we add the axiom $(\lceil s_1 \rceil, \lceil s_2 \rceil)$.
- Beside the original rules, we allow three new quantifications :
  1. We lift constructs of realizer at the level of the logic,
  2. We allow quantification over programs,
  3. We allow the formation of predicates.

A bit of vocabulary

- a type inhabits an original sort s

$$\Gamma \vdash A : s$$

- a formula inhabits a lifted sort $\lceil s \rceil$

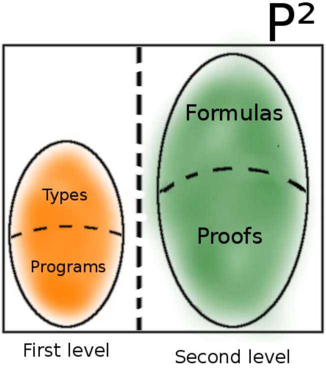
$$\Gamma \vdash A : \lceil s \rceil$$

- a program inhabits a type

$$\Gamma \vdash A : B : s$$

- a proof inhabits a formula

$$\Gamma \vdash A : B : \lceil s \rceil$$



- types & programs are first-level terms
- formulas & proofs are second-level terms

Second-order logic F^2

The PTS F^2 has the following specification:

$$\begin{array}{rcll}
\mathcal{S}_F^2 & = & \{ & \star, \square, \lceil\star\rceil, \lceil\square\rceil \quad \} \\
\mathcal{A}_F^2 & = & \{ & (\star, \square), (\lceil\star\rceil, \lceil\square\rceil) \quad \} \\
\mathcal{R}_F^2 & = & \{ & (\star,\star,\star), (\square,\star,\star), (\lceil\star\rceil,\lceil\star\rceil,\lceil\star\rceil), (\lceil\square\rceil,\lceil\star\rceil,\lceil\star\rceil) \\
 & & & (\star,\lceil\square\rceil,\lceil\square\rceil), (\star,\lceil\star\rceil,\lceil\star\rceil), (\square,\lceil\star\rceil,\lceil\star\rceil) \quad \}.
\end{array}$$

The logic F^2 is a second-order logic with higher-order typed individuals (FA_2 with higher-order individuals).

Second-order logic F^2

The PTS F^2 has the following specification:

$$\begin{array}{rcll}
\mathcal{S}_F^2 & = & \{ & \star, \square, \lceil\star\rceil, \lceil\square\rceil \quad \} \\
\mathcal{A}_F^2 & = & \{ & (\star, \square), (\lceil\star\rceil, \lceil\square\rceil) \quad \} \\
\mathcal{R}_F^2 & = & \{ & (\star, \star, \star), (\square, \star, \star), (\lceil\star\rceil, \lceil\star\rceil, \lceil\star\rceil), (\lceil\square\rceil, \lceil\star\rceil, \lceil\star\rceil) \\
 & & & (\star, \lceil\square\rceil, \lceil\square\rceil), (\star, \lceil\star\rceil, \lceil\star\rceil), (\square, \lceil\star\rceil, \lceil\star\rceil) \quad \}.
\end{array}$$

The logic F^2 is a second-order logic with higher-order typed individuals (FA_2 with higher-order individuals).

- $\lceil\star\rceil$ is the sort of formulas (like `Prop` in Coq).

Second-order logic F^2

The PTS F^2 has the following specification:

$$
\begin{array}{rcll}
\mathcal{S}_F^2 & = & \{ & \star, \Box, \lceil\star\rceil, \lceil\Box\rceil \quad \} \\
\mathcal{A}_F^2 & = & \{ & (\star, \Box), (\lceil\star\rceil, \lceil\Box\rceil) \quad \} \\
\mathcal{R}_F^2 & = & \{ & (\star,\star,\star), (\Box,\star,\star), (\lceil\star\rceil,\lceil\star\rceil,\lceil\star\rceil), (\lceil\Box\rceil,\lceil\star\rceil,\lceil\star\rceil) \\
 & & & (\star,\lceil\Box\rceil,\lceil\Box\rceil), (\star,\lceil\star\rceil,\lceil\star\rceil), (\Box,\lceil\star\rceil,\lceil\star\rceil) \quad \}.
\end{array}
$$

The logic F^2 is a second-order logic with higher-order typed individuals (FA_2 with higher-order individuals).

- $\lceil\star\rceil$ is the sort of formulas (like `Prop` in Coq).
- $(\lceil\star\rceil, \lceil\star\rceil, \lceil\star\rceil)$ allows to build implication $P \to Q$.

Second-order logic F^2

The PTS F^2 has the following specification:

$$\begin{array}{rcll}
\mathcal{S}_F^2 & = & \{ & \star, \Box, \lceil\star\rceil, \lceil\Box\rceil \quad \} \\
\mathcal{A}_F^2 & = & \{ & (\star, \Box), (\lceil\star\rceil, \lceil\Box\rceil) \quad \} \\
\mathcal{R}_F^2 & = & \{ & (\star,\star,\star), (\Box,\star,\star), (\lceil\star\rceil,\lceil\star\rceil,\lceil\star\rceil), (\lceil\Box\rceil,\lceil\star\rceil,\lceil\star\rceil) \\
 & & & (\star,\lceil\Box\rceil,\lceil\Box\rceil), (\star,\lceil\star\rceil,\lceil\star\rceil), (\Box,\lceil\star\rceil,\lceil\star\rceil) \quad \}.
\end{array}$$

The logic F^2 is a second-order logic with higher-order typed individuals (FA_2 with higher-order individuals).

- $\lceil\star\rceil$ is the sort of formulas (like `Prop` in Coq).
- $(\lceil\star\rceil, \lceil\star\rceil, \lceil\star\rceil)$ allows to build implication $P \to Q$.
- $(\star, \lceil\star\rceil, \lceil\star\rceil)$ allows to quantify over programs $\forall x : \tau.P$.

Second-order logic F^2

The PTS F^2 has the following specification:

$$
\begin{array}{lcll}
\mathcal{S}_F^2 & = & \{ & \star, \Box, \lceil\star\rceil, \lceil\Box\rceil \quad \} \\
\mathcal{A}_F^2 & = & \{ & (\star, \Box), (\lceil\star\rceil, \lceil\Box\rceil) \quad \} \\
\mathcal{R}_F^2 & = & \{ & (\star,\star,\star), (\Box,\star,\star), (\lceil\star\rceil, \lceil\star\rceil, \lceil\star\rceil), (\lceil\Box\rceil, \lceil\star\rceil, \lceil\star\rceil) \\
 & & & (\star, \lceil\Box\rceil, \lceil\Box\rceil), (\star, \lceil\star\rceil, \lceil\star\rceil), (\Box, \lceil\star\rceil, \lceil\star\rceil) \quad \}.
\end{array}
$$

The logic F^2 is a second-order logic with higher-order typed individuals (FA_2 with higher-order individuals).

- $\lceil\star\rceil$ is the sort of formulas (like `Prop` in Coq).
- $(\lceil\star\rceil, \lceil\star\rceil, \lceil\star\rceil)$ allows to build implication $P \to Q$.
- $(\star, \lceil\star\rceil, \lceil\star\rceil)$ allows to quantify over programs $\forall x : \tau. P$.
- $(\Box, \lceil\star\rceil, \lceil\star\rceil)$ allows to quantify over types $\forall \alpha. P$.

Second-order logic F^2

The PTS F^2 has the following specification:

$$\begin{array}{lcll} \mathcal{S}_F^2 & = & \{ & \star, \Box, \lceil\star\rceil, \lceil\Box\rceil \} \\ \mathcal{A}_F^2 & = & \{ & (\star, \Box), (\lceil\star\rceil, \lceil\Box\rceil) \} \\ \mathcal{R}_F^2 & = & \{ & (\star,\star,\star), (\Box,\star,\star), (\lceil\star\rceil, \lceil\star\rceil, \lceil\star\rceil), (\lceil\Box\rceil, \lceil\star\rceil, \lceil\star\rceil) \\ & & & (\star, \lceil\Box\rceil, \lceil\Box\rceil), (\star, \lceil\star\rceil, \lceil\star\rceil), (\Box, \lceil\star\rceil, \lceil\star\rceil) \}. \end{array}$$

The logic F^2 is a second-order logic with higher-order typed individuals (FA_2 with higher-order individuals).

- $\lceil\star\rceil$ is the sort of formulas (like `Prop` in Coq).
- $(\lceil\star\rceil, \lceil\star\rceil, \lceil\star\rceil)$ allows to build implication $P \to Q$.
- $(\star, \lceil\star\rceil, \lceil\star\rceil)$ allows to quantify over programs $\forall x : \tau.P$.
- $(\Box, \lceil\star\rceil, \lceil\star\rceil)$ allows to quantify over types $\forall \alpha.P$.
- $(\star, \lceil\Box\rceil, \lceil\Box\rceil)$ is used to build signatures of predicates. They are all of the form $\tau_1 \to \cdots \to \tau_n \to \lceil\star\rceil$.

Second-order logic F^2

The PTS F^2 has the following specification:

$$
\begin{array}{rcl}
\mathcal{S}_F^2 & = & \{ \star, \Box, \lceil\star\rceil, \lceil\Box\rceil \} \\
\mathcal{A}_F^2 & = & \{ (\star, \Box), (\lceil\star\rceil, \lceil\Box\rceil) \} \\
\mathcal{R}_F^2 & = & \{ (\star,\star,\star), (\Box,\star,\star), (\lceil\star\rceil, \lceil\star\rceil, \lceil\star\rceil), (\lceil\Box\rceil, \lceil\star\rceil, \lceil\star\rceil) \\
& & (\star, \lceil\Box\rceil, \lceil\Box\rceil), (\star, \lceil\star\rceil, \lceil\star\rceil), (\Box, \lceil\star\rceil, \lceil\star\rceil) \}.
\end{array}
$$

The logic F^2 is a second-order logic with higher-order typed individuals (FA_2 with higher-order individuals).

- $\lceil\star\rceil$ is the sort of formulas (like `Prop` in Coq).
- $(\lceil\star\rceil, \lceil\star\rceil, \lceil\star\rceil)$ allows to build implication $P \to Q$.
- $(\star, \lceil\star\rceil, \lceil\star\rceil)$ allows to quantify over programs $\forall x : \tau.P$.
- $(\Box, \lceil\star\rceil, \lceil\star\rceil)$ allows to quantify over types $\forall \alpha.P$.
- $(\star, \lceil\Box\rceil, \lceil\Box\rceil)$ is used to build signatures of predicates. They are all of the form $\tau_1 \to \cdots \to \tau_n \to \lceil\star\rceil$.
- $(\lceil\Box\rceil, \lceil\star\rceil, \lceil\star\rceil)$ allows to quantify over predicates $\forall X : \tau_1 \to \cdots \to \tau_n \to \lceil\star\rceil.P$.

Second-order logic F^2 – A stratified presentation

- We can prove that F^2 is equivalent to this presentation:

programs:

$$t, t_1, t_2 \quad := \quad x \quad | \ \lambda x : \tau . t \ | \ \Lambda \alpha . \tau \ | \quad (t_1 \, t_2) \quad | \quad (t \, \tau)$$

types:

$$\tau, \sigma \quad := \quad \alpha \quad | \ \sigma \rightarrow \tau \ | \ \forall \alpha . \tau$$

Second-order logic F^2 – A stratified presentation

- We can prove that F^2 is equivalent to this presentation:

programs:

$$t, t_1, t_2 \quad := \quad x \mid \lambda x : \tau.t \mid \Lambda\alpha.\tau \mid (t_1\, t_2) \mid (t\, \tau)$$

types:

$$\tau, \sigma \quad := \quad \alpha \mid \sigma \to \tau \mid \forall\alpha.\tau$$

formulas:

$$P, Q \quad := \quad X\, t_1 ... t_n \mid P \to Q \mid \forall\alpha.P \mid \forall x : \tau.P \mid \forall X : \tau_1 \to ... \to \tau_n \to \mathtt{Prop}.P$$

Second-order logic F^2 – A stratified presentation

- We can prove that F^2 is equivalent to this presentation:

programs:

$$t, t_1, t_2 \quad := \quad x \mid \lambda x : \tau . t \mid \Lambda \alpha . \tau \mid (t_1\, t_2) \mid (t\, \tau)$$

types:

$$\tau, \sigma \quad := \quad \alpha \mid \sigma \to \tau \mid \forall \alpha . \tau$$

formulas:

$$P, Q \quad := \quad X\, t_1 \ldots t_n \mid P \to Q \mid \forall \alpha . P \mid \forall x : \tau . P \mid \forall X : \tau_1 \to \ldots \to \tau_n \to \texttt{Prop} . P$$

- \+ a proof system

Second-order logic F^2 – A stratified presentation

- We can prove that F^2 is equivalent to this presentation:

programs:

$$t, t_1, t_2 \quad := \quad x \mid \lambda x : \tau.t \mid \Lambda\alpha.\tau \mid (t_1\, t_2) \mid (t\, \tau)$$

types:

$$\tau, \sigma \quad := \quad \alpha \mid \sigma \to \tau \mid \forall\alpha.\tau$$

formulas:

$$P, Q \quad := \quad X\, t_1 \ldots t_n \mid P \to Q \mid \forall\alpha.P \mid \forall x : \tau.P \mid \forall X : \tau_1 \to \ldots \to \tau_n \to \mathtt{Prop}.P$$

- + a proof system
- In the PTS presentation, proofs are represented by terms

Second-order logic: F^2 – Examples

Here are some examples in F^2.

- Truth: $\top \equiv \forall X : \lceil\star\rceil . X \to X$
  and is proved by $\lambda X : \lceil\star\rceil (h : X) . h$

Second-order logic: F^2 – Examples

Here are some examples in F^2.

- Truth: $\top \equiv \forall X : \lceil \star \rceil . X \to X$
  and is proved by $\lambda X : \lceil \star \rceil (h : X) . h$
- Leibniz equality: $x =_\tau y \equiv \forall X : \tau \to \lceil \star \rceil . X\, x \to X\, y$

Second-order logic: F^2 – Examples

Here are some examples in F^2.

- Truth: $\top \equiv \forall X : \lceil\star\rceil . X \to X$
  and is proved by $\lambda X : \lceil\star\rceil (h : X).h$
- Leibniz equality: $x =_\tau y \equiv \forall X : \tau \to \lceil\star\rceil . X\,x \to X\,y$
- $\forall(\alpha : \star)(x : \alpha).x =_\alpha x$ is proved by
  $\lambda(\alpha : \star)(x : \alpha)(X : \alpha \to \lceil\star\rceil)(h : X\,x).h$

Second-order logic: F^2 – Examples

Here are some examples in F^2.

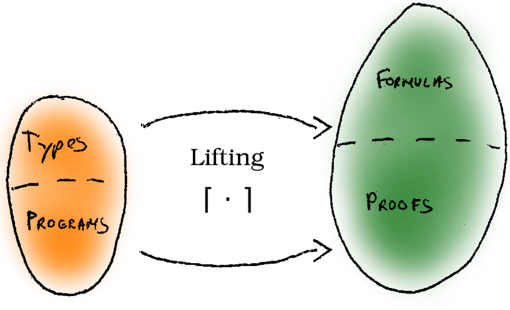
- Truth: $\top \equiv \forall X : \lceil \star \rceil . X \to X$
  and is proved by $\lambda X : \lceil \star \rceil (h : X).h$
- Leibniz equality: $x =_\tau y \equiv \forall X : \tau \to \lceil \star \rceil . X\, x \to X\, y$
- $\forall(\alpha : \star)(x : \alpha).x =_\alpha x$ is proved by
  $\lambda(\alpha : \star)(x : \alpha)(X : \alpha \to \lceil \star \rceil)(h : X\, x).h$
- The induction principle over Nat:

$$N \equiv \lambda x : \mathrm{Nat}\,.\forall X : \mathrm{Nat} \to \lceil \star \rceil.(\forall y : \mathrm{Nat}\,.X\, y \to X\,(\mathrm{Succ}\ y)) \to X\, 0 \to X\, x$$

Lifting and projection

Lifting

$\lceil\cdot\rceil$ embeds the first level toward its copy.

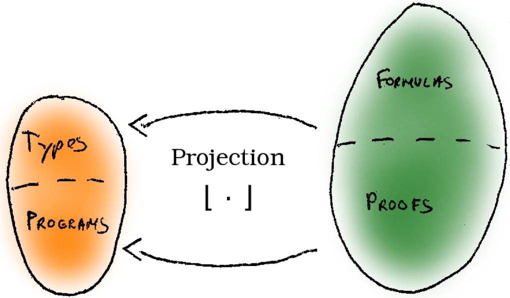


$$\lceil\forall\alpha:\star.\alpha\to\alpha\rceil \equiv \forall X:\lceil\star\rceil.X\to X$$
$$\lceil\mathsf{Nat}\rceil \equiv \forall X:\lceil\star\rceil.(X\to X)\to X\to X$$

Lifting and projection

Projection

$\lfloor \cdot \rfloor$ collapses the second level toward the first level.



$$\lfloor t_1 =_\tau t_2 \rfloor \equiv \lfloor \forall X : \tau \to \lceil \star \rceil . X\, t_1 \to X\, t_2 \rfloor \equiv \forall \alpha : \star . \alpha \to \alpha$$

$$\begin{aligned} \lfloor N\, t \rfloor &\equiv \lfloor \forall X : \mathsf{Nat} \to \lceil \star \rceil . (\forall y : \mathsf{Nat} . X\, y \to X\, (\mathsf{Succ}\, y)) \to X\, 0 \to X\, t \rfloor \\ &\equiv \forall \alpha . (\alpha \to \alpha) \to \alpha \to \alpha \equiv \mathsf{Nat} \end{aligned}$$

Lifting and projection – Lemmas

Lifting and projection – Lemmas

Lifting preserves typing

$$\Gamma \vdash A : B : s \Rightarrow \lceil \Gamma \rceil \vdash \lceil A \rceil : \lceil B \rceil : \lceil s \rceil$$

Lifting and projection – Lemmas

Lifting preserves typing

$$\Gamma \vdash A : B : s \Rightarrow \lceil \Gamma \rceil \vdash \lceil A \rceil : \lceil B \rceil : \lceil s \rceil$$

Lifting preserves β-reduction

$$A \longrightarrow_\beta B \Rightarrow \lceil A \rceil \longrightarrow_\beta \lceil B \rceil$$

Lifting and projection – Lemmas

Lifting preserves typing

$$\Gamma \vdash A : B : s \Rightarrow \lceil \Gamma \rceil \vdash \lceil A \rceil : \lceil B \rceil : \lceil s \rceil$$

Lifting preserves β-reduction

$$A \longrightarrow_\beta B \Rightarrow \lceil A \rceil \longrightarrow_\beta \lceil B \rceil$$

Projection preserves typing

$$\Gamma \vdash A : B : \lceil s \rceil \Rightarrow \lfloor \Gamma \rfloor \vdash \lfloor A \rfloor : \lfloor B \rfloor : s$$

Lifting and projection – Lemmas

Lifting preserves typing

$$\Gamma \vdash A : B : s \Rightarrow \lceil \Gamma \rceil \vdash \lceil A \rceil : \lceil B \rceil : \lceil s \rceil$$

Lifting preserves β-reduction

$$A \longrightarrow_\beta B \Rightarrow \lceil A \rceil \longrightarrow_\beta \lceil B \rceil$$

Projection preserves typing

$$\Gamma \vdash A : B : \lceil s \rceil \Rightarrow \lfloor \Gamma \rfloor \vdash \lfloor A \rfloor : \lfloor B \rfloor : s$$

Projection preserves or removes β-reduction

If $A \longrightarrow_\beta B$, then either $\lfloor A \rfloor \longrightarrow_\beta \lfloor B \rfloor$ or $\lfloor A \rfloor = \lfloor B \rfloor$.

Lifting and projection – Lemmas

Lifting preserves typing

$$\Gamma \vdash A : B : s \Rightarrow \lceil \Gamma \rceil \vdash \lceil A \rceil : \lceil B \rceil : \lceil s \rceil$$

Lifting preserves β-reduction

$$A \longrightarrow_\beta B \Rightarrow \lceil A \rceil \longrightarrow_\beta \lceil B \rceil$$

Projection preserves typing

$$\Gamma \vdash A : B : \lceil s \rceil \Rightarrow \lfloor \Gamma \rfloor \vdash \lfloor A \rfloor : \lfloor B \rfloor : s$$

Projection preserves or removes β-reduction

If $A \longrightarrow_\beta B$, then either $\lfloor A \rfloor \longrightarrow_\beta \lfloor B \rfloor$ or $\lfloor A \rfloor = \lfloor B \rfloor$.

Projection is the left inverse of lifting

$$\lfloor \lceil A \rceil \rfloor = A$$

Strong normalization

Theorem (Normalization)

If P is strongly normalizing, so is P^2.

Strong normalization

Theorem (Normalization)

If P is strongly normalizing, so is P^2.

Proof sketch.

If a term A is typable in P^2 and not normalizable, then :

- one of the first-level subterms of A is not normalizable, or
- the first-level term $\lfloor A \rfloor$ is not normalizable.

□

Parametricity and realizability in PTS's

In the following sections,

- We are going to define a parametricity relation :
  $(A, B) \in [\![C]\!]$ (we no longer use the notation $A \sim_C B$)
- and a realizability relation : $A \Vdash B$.

Parametricity and realizability in PTS's

In the following sections,

- We are going to define a parametricity relation :
  $(A, B) \in [\![C]\!]$ (we no longer use the notation $A \sim_C B$)
- and a realizability relation : $A \Vdash B$.

$$\overbrace{(\ \underset{\substack{\uparrow \\ \textit{Program}}}{\cdot}\ ,\ \underset{\substack{\uparrow \\ \textit{Program}}}{\cdot}\ ) \in [\![\ \underset{\substack{\uparrow \\ \textit{Type}}}{\cdot}\ ]\!]}^{\textit{Formula}} \qquad\qquad \overbrace{\underset{\substack{\uparrow \\ \textit{Program}}}{\cdot}\ \Vdash\ \underset{\substack{\uparrow \\ \textit{Formula}}}{\cdot}}^{\textit{Formula}}$$

Parametricity and realizability in PTS's

In the following sections,

- We are going to define a parametricity relation :
  $(A, B) \in [\![C]\!]$ (we no longer use the notation $A \sim_C B$)
- and a realizability relation : $A \Vdash B$.

$$\overbrace{(\;\cdot\;,\;\cdot\;) \in [\![\;\cdot\;]\!]}^{\textit{Formula}} \qquad \overbrace{\cdot \Vdash \cdot}^{\textit{Formula}}$$

Program / Program / Type
Type / Type / Sort

Program / Formula
Type / Lifted Sort

Parametricity in PTS's

- We define at the same time :
  - a ternary notation $(\cdot, \cdot) \in [\![\cdot]\!]$
  - a unary notation $[\![\cdot]\!]$

Parametricity in PTS's

- We define at the same time :
  - a ternary notation $(\cdot, \cdot) \in [\![\cdot]\!]$
  - a unary notation $[\![\cdot]\!]$
- We want to satisfy the abstraction theorem:

Parametricity in PTS's

- We define at the same time :
  - a ternary notation $(\cdot,\cdot) \in [\![\cdot]\!]$
  - a unary notation $[\![\cdot]\!]$
- We want to satisfy the abstraction theorem:

Theorem (abstraction)

If $\Gamma \vdash A : B : s$, *then*

$$[\![\Gamma]\!] \;\vdash\; [\![A]\!] \;:\; (A, A) \in [\![B]\!] \;:\; \lceil s \rceil$$

Parametricity in PTS's – Products, sorts and variables

$$
\begin{aligned}
&(A_1, A_2) \in [\![\forall x : B.C]\!] \equiv \\
&\qquad \forall (x_1 : B)(x_2 : B).(x_1, x_2) \in [\![B]\!] \to (A_1\, x_1, A_2\, x_2) \in [\![C]\!]
\end{aligned}
$$

Parametricity in PTS's – Products, sorts and variables

$$(A_1, A_2) \in [\![\forall x : B.C]\!] \equiv$$
$$\forall (x_1 : B)(x_2 : B)(x_R : (x_1, x_2) \in [\![B]\!]).(A_1\, x_1, A_2\, x_2) \in [\![C]\!]$$

Parametricity in PTS's – Products, sorts and variables

$$(A_1, A_2) \in [\![\forall x : B.C]\!] \equiv$$
$$\forall (x_1 : B)(x_2 : B)(x_R : (x_1, x_2) \in [\![B]\!]).(A_1\, x_1, A_2\, x_2) \in [\![C]\!]$$

$$(A_1, A_2) \in [\![x]\!] \equiv (x_R\, A_1\, A_2)$$

Parametricity in PTS's – Products, sorts and variables

$$(A_1, A_2) \in [\![\forall x : B.C]\!] \equiv \\ \forall (x_1 : B)(x_2 : B)(x_R : (x_1, x_2) \in [\![B]\!]).(A_1\, x_1, A_2\, x_2) \in [\![C]\!]$$

$$(A_1, A_2) \in [\![x]\!] \equiv (x_R\, A_1\, A_2)$$

$$(A_1, A_2) \in [\![s]\!] \equiv A_1 \to A_2 \to \lceil s \rceil$$

Parametricity in PTS's – Example

$$(t_1, t_2) \in [\![\forall\alpha : \star.\alpha \to \alpha]\!] \quad \equiv \quad \forall(\alpha_1 : \star)(\alpha_2 : \star)(\alpha_R : \ (\alpha_1, \alpha_2) \in [\![\star]\!]\ ). \\ (t_1\,\alpha_1, t_2\,\alpha_2) \in [\![\alpha \to \alpha]\!]$$

Parametricity in PTS's – Example

$$
\begin{aligned}
(t_1, t_2) \in [\![\forall \alpha : \star . \alpha \to \alpha]\!] \quad &\equiv \quad \forall (\alpha_1 : \star)(\alpha_2 : \star)(\alpha_R : \ (\alpha_1, \alpha_2) \in [\![\star]\!] \ ). \\
&\qquad (t_1\, \alpha_1, t_2\, \alpha_2) \in [\![\alpha \to \alpha]\!] \\
(\alpha_1, \alpha_2) \in [\![\star]\!] \quad &\equiv \quad \alpha_1 \to \alpha_2 \to \lceil \star \rceil
\end{aligned}
$$

Parametricity in PTS's – Example

$$
\begin{aligned}
(t_1, t_2) \in [\![\forall\alpha : \star.\alpha \to \alpha]\!] &\equiv \forall(\alpha_1 : \star)(\alpha_2 : \star)(\alpha_R : \alpha_1 \to \alpha_2 \to \lceil\star\rceil). \\
&\qquad (t_1\,\alpha_1, t_2\,\alpha_2) \in [\![\alpha \to \alpha]\!] \\
(\alpha_1, \alpha_2) \in [\![\star]\!] &\equiv \alpha_1 \to \alpha_2 \to \lceil\star\rceil
\end{aligned}
$$

Parametricity in PTS's – Example

$$
\begin{aligned}
(t_1, t_2) \in [\![\forall \alpha : \star . \alpha \to \alpha]\!] &\equiv \forall (\alpha_1 : \star)(\alpha_2 : \star)(\alpha_R : \alpha_1 \to \alpha_2 \to \lceil \star \rceil). \\
& \qquad (t_1\, \alpha_1, t_2\, \alpha_2) \in [\![\alpha \to \alpha]\!] \\
(\alpha_1, \alpha_2) \in [\![\star]\!] &\equiv \alpha_1 \to \alpha_2 \to \lceil \star \rceil \\
(t_1\, \alpha_1, t_2\, \alpha_2) \in [\![\alpha \to \alpha]\!] &\equiv
\end{aligned}
$$

Parametricity in PTS's – Example

$$
\begin{aligned}
(t_1, t_2) \in [\![\forall \alpha : \star . \alpha \to \alpha]\!] &\equiv \forall (\alpha_1 : \star)(\alpha_2 : \star)(\alpha_R : \alpha_1 \to \alpha_2 \to \lceil \star \rceil). \\
&\qquad (t_1\, \alpha_1, t_2\, \alpha_2) \in [\![\alpha \to \alpha]\!] \\
(\alpha_1, \alpha_2) \in [\![\star]\!] &\equiv \alpha_1 \to \alpha_2 \to \lceil \star \rceil \\
(t_1\, \alpha_1, t_2\, \alpha_2) \in [\![\alpha \to \alpha]\!] &\equiv \forall (x_1 : \alpha)(x_2 : \alpha). \\
&\qquad (x_1, x_2) \in [\![\alpha]\!] \to (t_1\, \alpha_1\, x_1, t_2\, \alpha_2\, x_2) \in [\![\alpha]\!]
\end{aligned}
$$

Parametricity in PTS's – Example

$$
\begin{array}{rcl}
(t_1, t_2) \in [\![\forall\alpha : \star.\alpha \to \alpha]\!] & \equiv & \forall(\alpha_1 : \star)(\alpha_2 : \star)(\alpha_R : \alpha_1 \to \alpha_2 \to \lceil\star\rceil). \\
 & & \quad (t_1\,\alpha_1, t_2\,\alpha_2) \in [\![\alpha \to \alpha]\!] \\
(\alpha_1, \alpha_2) \in [\![\star]\!] & \equiv & \alpha_1 \to \alpha_2 \to \lceil\star\rceil \\
(t_1\,\alpha_1, t_2\,\alpha_2) \in [\![\alpha \to \alpha]\!] & \equiv & \forall(x_1 : \alpha)(x_2 : \alpha). \\
 & & \quad (x_1, x_2) \in [\![\alpha]\!] \to (t_1\,\alpha_1\,x_1, t_2\,\alpha_2\,x_2) \in [\![\alpha]\!] \\
(A, B) \in [\![\alpha]\!] & \equiv & \alpha_R\,A\,B
\end{array}
$$

Parametricity in PTS's – Example

$$
\begin{aligned}
(t_1, t_2) \in [\![\forall \alpha : \star . \alpha \to \alpha]\!] &\equiv \forall (\alpha_1 : \star)(\alpha_2 : \star)(\alpha_R : \alpha_1 \to \alpha_2 \to \lceil \star \rceil). \\
&\qquad (t_1\, \alpha_1, t_2\, \alpha_2) \in [\![\alpha \to \alpha]\!] \\
(\alpha_1, \alpha_2) \in [\![\star]\!] &\equiv \alpha_1 \to \alpha_2 \to \lceil \star \rceil \\
(t_1\, \alpha_1, t_2\, \alpha_2) \in [\![\alpha \to \alpha]\!] &\equiv \forall (x_1 : \alpha)(x_2 : \alpha). \\
&\qquad \alpha_R\, x_1\, x_2 \to \alpha_R\, (t_1\, \alpha_1\, x_1)\, (t_2\, \alpha_2\, x_2) \\
(A, B) \in [\![\alpha]\!] &\equiv \alpha_R\, A\, B
\end{aligned}
$$

Parametricity in PTS's – Example

$$
\begin{aligned}
(t_1, t_2) \in [\![\forall \alpha : \star . \alpha \to \alpha]\!] &\equiv \forall (\alpha_1 : \star)(\alpha_2 : \star)(\alpha_R : \alpha_1 \to \alpha_2 \to \lceil \star \rceil). \\
&\qquad (t_1\, \alpha_1, t_2\, \alpha_2) \in [\![\alpha \to \alpha]\!] \\
(\alpha_1, \alpha_2) \in [\![\star]\!] &\equiv \alpha_1 \to \alpha_2 \to \lceil \star \rceil \\
(t_1\, \alpha_1, t_2\, \alpha_2) \in [\![\alpha \to \alpha]\!] &\equiv \forall (x_1 : \alpha)(x_2 : \alpha). \\
&\qquad \alpha_R\, x_1\, x_2 \to \alpha_R\, (t_1\, \alpha_1\, x_1)\, (t_2\, \alpha_2\, x_2) \\
(A, B) \in [\![\alpha]\!] &\equiv \alpha_R\, A\, B
\end{aligned}
$$

Finally,

$$
\begin{aligned}
&(t_1, t_2) \in [\![\forall \alpha : \star . \alpha \to \alpha]\!] \equiv \\
&\quad \forall (\alpha_1 : \star)(\alpha_2 : \star)(\alpha_R : \alpha_1 \to \alpha_2 \to \lceil \star \rceil). \\
&\qquad \forall (x_1 : \alpha_1)(x_2 : \alpha_2). \alpha_R\, x_1\, x_2 \to \alpha_R (t_1\, \alpha_1\, x_1)\, (t_2\, \alpha_2\, x_2)
\end{aligned}
$$

Parametricity in PTS's

- Here is the transformation for the product:

$$(A_1, A_2) \in [\![\forall x : B.C]\!] \equiv$$
$$\forall (x_1 : B)(x_2 : B)(x_R : (x_1, x_2) \in [\![B]\!]).(A_1\, x_1, A_2\, x_2) \in [\![C]\!]$$

Parametricity in PTS's

- Here is the transformation for the product:

$$(A_1, A_2) \in [\![\forall x : B.C]\!] \equiv \forall (x_1 : B)(x_2 : B)(x_R : (x_1, x_2) \in [\![B]\!]).(A_1\, x_1, A_2\, x_2) \in [\![C]\!]$$

- If we have $\vdash (\lambda x : B.A) : (\forall x : B.C)$, since we want to satisfy the abstraction theorem, we must take

$$[\![\lambda x : B.A]\!] \equiv \lambda (x_1 : B)(x_2 : B)(x_R : (x_1, x_2) \in [\![B]\!]).[\![A]\!]$$

Parametricity in PTS's

- Here is the transformation for the product:

$$(A_1, A_2) \in [\![\forall x : B.C]\!] \equiv \\ \forall(x_1 : B)(x_2 : B)(x_R : (x_1, x_2) \in [\![B]\!]).(A_1\, x_1, A_2\, x_2) \in [\![C]\!]$$

- If we have $\vdash (\lambda x : B.A) : (\forall x : B.C)$, since we want to satisfy the abstraction theorem, we must take

$$[\![\lambda x : B.A]\!] \equiv \lambda(x_1 : B)(x_2 : B)(x_R : (x_1, x_2) \in [\![B]\!]).[\![A]\!]$$

- Symmetrically, we need to take $[\![(A\, B)]\!] \equiv ([\![A]\!]\; B\; B\; [\![B]\!])$.

Parametricity in PTS's – The whole definition

Definition (parametricity)

$(C_1, C_2) \in [\![s]\!]$
$(C_1, C_2) \in [\![\forall x : A.\, B]\!]$

$(C_1, C_2) \in [\![T]\!]$
$[\![x]\!]$
$[\![\lambda x : A.B]\!]$
$[\![A\, B]\!]$
$[\![T]\!]$

Parametricity in PTS's – The whole definition

Definition (parametricity)

$$
\begin{array}{lcl}
(C_1, C_2) \in [\![s]\!] & = & C_1 \to C_2 \to \lceil s \rceil \\
(C_1, C_2) \in [\![\forall x : A.\, B]\!] & = & \forall (x_1 : A)(x_2 : A)(x_R : (x_1, x_2) \in [\![A]\!]). \\
 & & \quad (C_1\, x_1, C_2\, x_2) \in [\![B]\!] \\
(C_1, C_2) \in [\![T]\!] & = & ([\![T]\!]\; C_1\; C_2)\ \underline{\text{otherwise}} \\
[\![x]\!] & & \\
[\![\lambda x : A.B]\!] & & \\
[\![A\, B]\!] & & \\
[\![T]\!] & &
\end{array}
$$

Theorem (abstraction)

If $\Gamma \vdash A : B : s$, *then* $[\![\Gamma]\!] \vdash [\![A]\!] : (A, A) \in [\![B]\!] : \lceil s \rceil$

Parametricity in PTS's – The whole definition

Definition (parametricity)

$$
\begin{array}{lcl}
(C_1, C_2) \in [\![s]\!] & = & C_1 \to C_2 \to \lceil s \rceil \\
(C_1, C_2) \in [\![\forall x : A.\, B]\!] & = & \forall (x_1 : A)(x_2 : A)(x_R : (x_1, x_2) \in [\![A]\!]). \\
 & & \quad (C_1\, x_1, C_2\, x_2) \in [\![B]\!] \\
(C_1, C_2) \in [\![T]\!] & = & ([\![T]\!]\, C_1\, C_2)\ \underline{\text{otherwise}} \\
[\![x]\!] & = & x_R \\
[\![\lambda x : A.B]\!] & = & \lambda (x_1 : A)(x_2 : A)(x_R : (x_1, x_2) \in [\![A]\!]).[\![B]\!] \\
[\![A\, B]\!] & = & [\![A]\!]\, B\, B\, [\![B]\!] \\
[\![T]\!] & = & \lambda (x_1\, x_2 : T).(x_1, x_2) \in [\![T]\!]\ \underline{\text{otherwise}}
\end{array}
$$

Theorem (abstraction)

If $\Gamma \vdash A : B : s$, *then* $[\![\Gamma]\!] \vdash [\![A]\!] : (A, A) \in [\![B]\!] : \lceil s \rceil$

Parametricity in PTS's – The *n*-ary version

Definition (parametricity)

$$
\begin{array}{lcl}
\overline{C} \in [\![s]\!]_n & = & \overline{C} \to \lceil s \rceil \\
\overline{C} \in [\![\forall x : A.\, B]\!]_n & = & \forall \overline{x : A}.\, \forall x_R : \overline{x} \in [\![A]\!]_n.\, \overline{z\, x} \in [\![B]\!]_n \\
\overline{C} \in [\![T]\!]_n & = & [\![T]\!]_n\, \overline{C} \ \underline{\text{otherwise}} \\
[\![x]\!]_n & = & x_R \\
[\![\lambda x : A.\, B]\!]_n & = & \lambda \overline{x : A}.\, \lambda x_R : \overline{x} \in [\![A]\!]_n.\, [\![B]\!]_n \\
[\![A\, B]\!]_n & = & [\![A]\!]_n\, \overline{B}\, [\![B]\!]_n \\
[\![T]\!]_n & = & \lambda \overline{z : T}.\, \overline{C} \in [\![T]\!]_n \ \underline{\text{otherwise}}
\end{array}
$$

Theorem (abstraction)

If $\Gamma \vdash A : B : s$, *then* $[\![\Gamma]\!]_n \vdash [\![A]\!]_n : \overline{A} \in [\![B]\!]_n : \lceil s \rceil$

Realizability in second-order logic

In traditional presentation of realizability:

- $t \Vdash P \to Q \equiv \forall x, x \Vdash P \to (t\,x) \Vdash Q$

Realizability in second-order logic

In traditional presentation of realizability:

- $t \Vdash P \to Q \equiv \forall x, x \Vdash P \to (t\, x) \Vdash Q$
- $t \Vdash \forall x.P \equiv \forall x, t \Vdash P$

Realizability in second-order logic

In traditional presentation of realizability:

- $t \Vdash P \rightarrow Q \equiv \forall x, x \Vdash P \rightarrow (t\,x) \Vdash Q$
- $t \Vdash \forall x.P \equiv \forall x, t \Vdash P$

There are two kinds of quantification:

- First-level quantification
- Second-level quantification

Realizability in second-order logic

In traditional presentation of realizability:

- $t \Vdash P \to Q \equiv \forall x, x \Vdash P \to (t\, x) \Vdash Q$
- $t \Vdash \forall x. P \equiv \forall x, t \Vdash P$

There are two kinds of quantification:

- First-level quantification
- Second-level quantification

Realizability in second-order logic

In traditional presentation of realizability:

- $t \Vdash P \rightarrow Q \equiv \forall x, x \Vdash P \rightarrow (t\, x) \Vdash Q$
- $t \Vdash \forall x.P \equiv \forall x, t \Vdash P$

There are two kinds of quantification:

- First-level quantification : uniform,
- Second-level quantification

Realizability in second-order logic

In traditional presentation of realizability:

- $t \Vdash P \to Q \equiv \forall x, x \Vdash P \to (t\,x) \Vdash Q$
- $t \Vdash \forall x.P \equiv \forall x, t \Vdash P$

There are two kinds of quantification:

- First-level quantification : uniform,
- Second-level quantification

Realizability in second-order logic

In traditional presentation of realizability:

- $t \Vdash P \to Q \equiv \forall x, x \Vdash P \to (t\,x) \Vdash Q$
- $t \Vdash \forall x.P \equiv \forall x, t \Vdash P$

There are two kinds of quantification:

- First-level quantification : uniform,
- Second-level quantification : things happen.

Pure Type Systems – A technical detail: sort annotations

- We annotate variables with the sort of their type

Pure Type Systems – A technical detail: sort annotations

- We annotate variables with the sort of their type
- Here is the product rule :

$$\textsc{Product}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x \quad : A \ \vdash B : s_2}{\Gamma \vdash (\forall x \quad : A.B)\ : s_3}\ (s_1, s_2, s_3) \in \mathcal{R}$$

Pure Type Systems – A technical detail: sort annotations

- We annotate variables with the sort of their type
- Here is the product rule :

$$\textsc{Product}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x^{s_1} : A \vdash B : s_2}{\Gamma \vdash (\forall x^{s_1} : A.B) : s_3}\ (s_1, s_2, s_3) \in \mathcal{R}$$

- We can distinguish the two kinds of quantification:
  - First-level quantification of the form $\forall x^{s} : A.B$,
  - Second-level quantification of the form $\forall x^{\lceil s \rceil} : A.B$.

Realizability in PTS's

- We define at the same time :
  - a binary notation $\cdot \Vdash \cdot$
  - a unary notation $\langle \cdot \rangle$
- We want to satisfy the adequacy theorem:

Theorem (adequacy)

If $\Gamma \vdash A : B : \lceil s \rceil$, *then*

$$\langle \Gamma \rangle \ \vdash \ \langle A \rangle \ : \ \lfloor A \rfloor \Vdash B \ : \ \lceil s \rceil$$

Realizability in PTS's – The products

- First level quantification :

$$C \Vdash \forall x^s : A.B = \forall x^s : A.C \Vdash B$$

Realizability in PTS's – The products

- First level quantification :

$$C \Vdash \forall x^s : A.B = \forall x^s : A.C \Vdash B$$

- Second level quantification :

$$C \Vdash \forall x^{\lceil s \rceil} : A.B = \forall (\lfloor x \rfloor^s : \lfloor A \rfloor)(x^{\lceil s \rceil} : \lfloor x \rfloor \Vdash A).(C\ \lfloor x \rfloor) \Vdash B$$

Realizability in PTS's – The products

- First level quantification :

$$C \Vdash \forall x^s : A.B = \forall x^s : A.C \Vdash B$$

- Second level quantification :

$$C \Vdash \forall x^{\lceil s \rceil} : A.B = \forall (\lfloor x \rfloor^s : \lfloor A \rfloor)(x^{\lceil s \rceil} : \lfloor x \rfloor \Vdash A).(C\ \lfloor x \rfloor) \Vdash B$$

- Sorts :

$$C \Vdash \lceil s \rceil = C \to \lceil s \rceil$$

Realizability in PTS's – The products

- First level quantification :

$$C \Vdash \forall x^s : A.B = \forall x^s : A.C \Vdash B$$

- Second level quantification :

$$C \Vdash \forall x^{\lceil s \rceil} : A.B = \forall (\lfloor x \rfloor^s : \lfloor A \rfloor)(x^{\lceil s \rceil} : \lfloor x \rfloor \Vdash A).(C\ \lfloor x \rfloor) \Vdash B$$

- Sorts :

$$C \Vdash \lceil s \rceil = C \to \lceil s \rceil$$

In F^2,

Realizability in PTS's – The products

- First level quantification :

$$C \Vdash \forall x^s : A.B = \forall x^s : A.C \Vdash B$$

- Second level quantification :

$$C \Vdash \forall x^{\lceil s \rceil} : A.B = \forall (\lfloor x \rfloor^s : \lfloor A \rfloor)(x^{\lceil s \rceil} : \lfloor x \rfloor \Vdash A).(C\ \lfloor x \rfloor) \Vdash B$$

- Sorts :

$$C \Vdash \lceil s \rceil = C \to \lceil s \rceil$$

In F^2,

$$t \Vdash \forall x : \tau.P \equiv \forall x : \tau.t \Vdash Q$$

Realizability in PTS's – The products

- First level quantification :

$$C \Vdash \forall x^s : A.B = \forall x^s : A.C \Vdash B$$

- Second level quantification :

$$C \Vdash \forall x^{\lceil s \rceil} : A.B = \forall (\lfloor x \rfloor^s : \lfloor A \rfloor)(x^{\lceil s \rceil} : \lfloor x \rfloor \Vdash A).(C \lfloor x \rfloor) \Vdash B$$

- Sorts :

$$C \Vdash \lceil s \rceil = C \to \lceil s \rceil$$

In F^2,

$$t \Vdash \forall x : \tau.P \equiv \forall x : \tau.t \Vdash Q$$

Realizability in PTS's – The products

- First level quantification :

$$C \Vdash \forall x^s : A.B = \forall x^s : A.C \Vdash B$$

- Second level quantification :

$$C \Vdash \forall x^{\lceil s \rceil} : A.B = \forall (\lfloor x \rfloor^s : \lfloor A \rfloor)(x^{\lceil s \rceil} : \lfloor x \rfloor \Vdash A).(C\, \lfloor x \rfloor) \Vdash B$$

- Sorts :

$$C \Vdash \lceil s \rceil = C \to \lceil s \rceil$$

In F^2,

$$t \Vdash \forall x : \tau.P \equiv \forall x : \tau.t \Vdash Q$$

$$t \Vdash P \to Q \equiv \forall x : \lfloor P \rfloor.x \Vdash P \to (t\, x) \Vdash Q$$

Realizability in PTS's – The products

- First level quantification :

$$C \Vdash \forall x^s : A.B = \forall x^s : A.C \Vdash B$$

- Second level quantification :

$$C \Vdash \forall x^{\lceil s \rceil} : A.B = \forall (\lfloor x \rfloor^s : \lfloor A \rfloor)(x^{\lceil s \rceil} : \lfloor x \rfloor \Vdash A).(C\, \lfloor x \rfloor) \Vdash B$$

- Sorts :

$$C \Vdash \lceil s \rceil = C \to \lceil s \rceil$$

In F^2,

$$t \Vdash \forall x : \tau.P \equiv \forall x : \tau.t \Vdash Q$$

$$t \Vdash P \to Q \equiv \forall x : \lfloor P \rfloor.x \Vdash P \to (t\, x) \Vdash Q$$

Realizability in PTS's – The products

- First level quantification :

$$C \Vdash \forall x^s : A.B = \forall x^s : A.C \Vdash B$$

- Second level quantification :

$$C \Vdash \forall x^{\lceil s \rceil} : A.B = \forall (\lfloor x \rfloor^s : \lfloor A \rfloor)(x^{\lceil s \rceil} : \lfloor x \rfloor \Vdash A).(C\,\lfloor x \rfloor) \Vdash B$$

- Sorts :

$$C \Vdash \lceil s \rceil = C \to \lceil s \rceil$$

In F^2,

$$t \Vdash \forall x : \tau.P \equiv \forall x : \tau.t \Vdash Q$$

$$t \Vdash P \to Q \equiv \forall x : \lfloor P \rfloor.x \Vdash P \to (t\,x) \Vdash Q$$

$$t \Vdash \forall X : \tau_1 \to \cdots \to \tau_n \to \lceil \star \rceil.P \equiv \forall \alpha : \star.\forall X : \tau_1 \to \cdots \to \tau_n \to \alpha \to \lceil \star \rceil.(t\,\alpha) \Vdash P$$

Realizability in PTS's – The whole definition

Definition (realizability)

$$
\begin{array}{lcl}
C \Vdash \lceil s \rceil & = & C \to \lceil s \rceil \\
C \Vdash \forall x^s : A.B & = & \forall x^s : A. C \Vdash B \\
C \Vdash \forall x^{\lceil s \rceil} : A.B & = & \forall (\lfloor x \rfloor^s : \lfloor A \rfloor)(x^{\lceil s \rceil} : \lfloor x \rfloor \Vdash A).(C\,\lfloor x \rfloor) \Vdash B \\
C \Vdash F & = & \langle F \rangle\, C \ \underline{\text{otherwise}}
\end{array}
$$

Theorem (adequacy)

If $\Gamma \vdash A : B : \lceil s \rceil$, *then*

$$\langle \Gamma \rangle \vdash \langle A \rangle : \lfloor A \rfloor \Vdash B : \lceil s \rceil$$

Realizability in PTS's – The whole definition

Definition (realizability)

$$
\begin{array}{lcl}
C \Vdash \lceil s \rceil & = & C \to \lceil s \rceil \\
C \Vdash \forall x^s : A.B & = & \forall x^s : A.C \Vdash B \\
C \Vdash \forall x^{\lceil s \rceil} : A.B & = & \forall (\lfloor x \rfloor^s : \lfloor A \rfloor)(x^{\lceil s \rceil} : \lfloor x \rfloor \Vdash A).(C\, \lfloor x \rfloor) \Vdash B \\
C \Vdash F & = & \langle F \rangle\, C \ \underline{\text{otherwise}} \\
\hline
\langle x^{\lceil s \rceil} \rangle & = & x^{\lceil s \rceil} \\
\langle \lambda x^s : A.B \rangle & = & \lambda x^s : A.\langle B \rangle \\
\langle \lambda x^{\lceil s \rceil} : A.B \rangle & = & \lambda (\lfloor x \rfloor^s : \lfloor A \rfloor)(x^{\lceil s \rceil} : \lfloor x \rfloor \Vdash A).\langle B \rangle \\
\langle (A\, B)_s \rangle & = & (\langle A \rangle\, B)_s \\
\langle (A\, B)_{\lceil s \rceil} \rangle & = & ((\langle A \rangle\, \lfloor B \rfloor)_s\, \langle B \rangle)_{\lceil s \rceil} \\
\langle T \rangle & = & \lambda z^s : \lfloor T \rfloor .\, z \Vdash T \ \underline{\text{otherwise}}
\end{array}
$$

Theorem (adequacy)

If $\Gamma \vdash A : B : \lceil s \rceil$, *then*

$$\langle \Gamma \rangle \vdash \langle A \rangle : \lfloor A \rfloor \Vdash B : \lceil s \rceil$$

From realizability to parametricity

Theorem (realizability increases arity of parametricity)

$$\left(B, \overline{C}\right) \in [\![A]\!]_{n+1} = B \Vdash \left(\overline{C} \in [\![A]\!]_n\right)$$

and

$$[\![A]\!]_{n+1} = \langle [\![A]\!]_n \rangle$$

Lemma (0-parametricity is lifting)

$$[\![A]\!]_0 \equiv \lceil A \rceil$$

We can define parametricity with lifting+realizability:

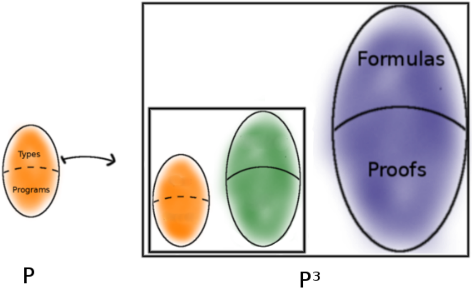
Corollary (From realizability to parametricity)

$$\overline{z} \in [\![A]\!]_n = z_1 \Vdash z_2 \Vdash \cdots \Vdash z_n \Vdash \lceil A \rceil$$

and

$$[\![A]\!]_n = \langle \cdots \langle \lceil A \rceil \rangle \cdots \rangle$$

A third level – From parametricity to realizability



Theorem (From parametricity to realizability)

If A is a second-level term, then

$$z \Vdash A = \lfloor \lceil z \rceil \in [\![A]\!]_1 \rfloor \qquad \textit{and} \qquad \langle A \rangle = \lfloor [\![A]\!]_1 \rfloor$$

Representation theorems

If $P = F$ or $P = F_\omega$ or $P =$ calculus of construction,

Representation theorems

If $P = F$ or $P = F_\omega$ or $P =$ calculus of construction,

- We can follow Krivine's methodology

Representation theorems

If $P = F$ or $P = F_\omega$ or $P =$ calculus of construction,

- We can follow Krivine's methodology
- Using second-order encoding:
  - We can encode Leibniz equality $\cdot =_\tau \cdot$
  - We use the induction principle $N\,x$ to encode integer in proofs

Representation theorems

If $P = F$ or $P = F_\omega$ or $P =$ calculus of construction,

- We can follow Krivine's methodology
- Using second-order encoding:
  - We can encode Leibniz equality $\cdot =_\tau \cdot$
  - We use the induction principle $N\,x$ to encode integer in proofs
- N is a datatype :

$$\forall r\, x, r \Vdash N\,x \Leftrightarrow (N\,x \wedge r =_{\mathrm{Nat}} x)$$

Representation theorems

If $P = F$ or $P = F_\omega$ or $P =$ calculus of construction,

- We can follow Krivine's methodology
- Using second-order encoding:
  - We can encode Leibniz equality $\cdot =_\tau \cdot$
  - We use the induction principle $N\,x$ to encode integer in proofs
- N is a datatype :

$$\forall r\, x, r \Vdash N\,x \Leftrightarrow (N\,x \wedge r =_{\mathrm{Nat}} x)$$

- From any proof π of

$$\forall x_1 ... x_n : \mathrm{Nat}, N\,x_1 \to \cdots \to N\,x_n \to N\,(f\,x_1\,...\,x_n)$$

Representation theorems

If $P = F$ or $P = F_\omega$ or $P =$ calculus of construction,

- We can follow Krivine's methodology
- Using second-order encoding:
  - We can encode Leibniz equality $\cdot =_\tau \cdot$
  - We use the induction principle $N\,x$ to encode integer in proofs
- N is a datatype :

$$\forall r\,x, r \Vdash N\,x \Leftrightarrow (N\,x \wedge r =_{\text{Nat}} x)$$

- From any proof π of

$$\forall x_1 ... x_n : \text{Nat}, N\,x_1 \to \cdots \to N\,x_n \to N\,(f\,x_1 \ldots x_n)$$

- ... we obtain a program $\lfloor \pi \rfloor$ such that $\lfloor \pi \rfloor =_{\text{Nat}} f$

Representation theorems

If $P = F$ or $P = F_\omega$ or $P =$ calculus of construction,

- We can follow Krivine's methodology
- Using second-order encoding:
  - We can encode Leibniz equality $\cdot =_\tau \cdot$
  - We use the induction principle $N\,x$ to encode integer in proofs
- N is a datatype :

$$\forall r\,x, r \Vdash N\,x \Leftrightarrow (N\,x \wedge r =_{\mathrm{Nat}} x)$$

- From any proof π of

$$\forall x_1 ... x_n : \mathrm{Nat}, N\,x_1 \to \cdots \to N\,x_n \to N\,(f\,x_1 \ldots x_n)$$

- ... we obtain a program $\lfloor \pi \rfloor$ such that $\lfloor \pi \rfloor =_{\mathrm{Nat}} f$
- Conversely : if $\vdash_P p : \mathrm{Nat} \to \mathrm{Nat}$ we can find π_p such that $\vdash_{P^2} \pi_p : \forall x : \mathrm{Nat}, N\,x \to N\,(p\,x)$.

Representation theorems

If $P = F$ or $P = F_\omega$ or $P =$ calculus of construction,

- We can follow Krivine's methodology
- Using second-order encoding:
  - We can encode Leibniz equality $\cdot =_\tau \cdot$
  - We use the induction principle $N\,x$ to encode integer in proofs
- N is a datatype :

$$\forall r\,x, r \Vdash N\,x \Leftrightarrow (N\,x \wedge r =_{\mathrm{Nat}} x)$$

- From any proof π of

$$\forall x_1 ... x_n : \mathrm{Nat}, N\,x_1 \to \cdots \to N\,x_n \to N\,(f\,x_1 \ldots x_n)$$

- ... we obtain a program $\lfloor \pi \rfloor$ such that $\lfloor \pi \rfloor =_{\mathrm{Nat}} f$
- Conversely : if $\vdash_P p : \mathrm{Nat} \to \mathrm{Nat}$ we can find π_p such that $\vdash_{P^2} \pi_p : \forall x : \mathrm{Nat}, N\,x \to N\,(p\,x)$.

Theorem

Arithmetic functions representable in P are those provably total in P^2.

Inductive types

- Encoding of conjunction:

$$\textbf{data}\ \_ \wedge \_ : \lceil s \rceil \to \lceil s \rceil \to \lceil s \rceil\ \textbf{where}$$
$$\quad \mathit{conj} : \Pi\, P\, Q : \lceil s \rceil . P \to Q \to P \wedge Q$$

Inductive types

- Encoding of conjunction:

$$\begin{array}{l}\textbf{data}\ \_ \wedge \_ : \lceil s \rceil \to \lceil s \rceil \to \lceil s \rceil\ \textbf{where} \\ \quad conj : \Pi\, P\, Q : \lceil s \rceil . P \to Q \to P \wedge Q\end{array}$$

- Projection $\lfloor \wedge \rfloor = \times$:

$$\begin{array}{l}\textbf{data}\ \_ \times \_ : s \to s \to s\ \textbf{where} \\ \quad (\_, \_) : \Pi\, \alpha\, \beta : s.\alpha \to \beta \to \alpha \times \beta\end{array}$$

$$\begin{aligned}
\textbf{data } \langle\wedge\rangle : \; & \Pi(\alpha : s).(\alpha \to \lceil s \rceil) \to \\
& \Pi(\beta : s).(\beta \to \lceil s \rceil) \to \\
& \alpha \times \beta \to s \textbf{ where} \\
\langle conj \rangle : \; & \Pi(\alpha : s)(P : \alpha \to \lceil s \rceil) \\
& \quad (\beta : s)(Q : \beta \to \lceil s \rceil)(x : \alpha)(y : \beta). \\
& \quad P\,x \to Q\,y \to \langle\wedge\rangle\,\alpha\,P\,\beta\,Q\,(x, y)
\end{aligned}$$

By definition, $t \Vdash P \wedge Q$ means $\langle\wedge\rangle\,\lfloor P \rfloor\,\langle P \rangle\,\lfloor Q \rfloor\,\langle Q \rangle\,t$. We have

$$t \Vdash P \wedge Q \Leftrightarrow (\pi_1\,t) \Vdash P \wedge (\pi_2\,t) \Vdash Q$$

where π_1 and π_2 are projections upon cartesian product.

Conclusion

- We gave a systematic way to formalize the meta-theory to study a programming language
- An account of parametricity and realizability in PTSs
- We exposed links between the two
- Extension: works with inductive types