Realizability and Parametricity in Pure Type Systems

Based on the paper by
Jean-Philippe Bernardy and Marc Lasson

Eskil Dam & Sophie Krijgsman

December 17, 2025

Introduction

- We start from a programming language described as a **Pure Type System (PTS)**.
- **Main idea of the paper:** build a **logic** on top of that PTS in which we can reason about its programmes.
- The original PTS P = *programming language* (types and terms).
- A new PTS P^2 is constructed as a *logic* whose formulas state properties about programmes in P.

Introduction

- We start from a programming language described as a **Pure Type System (PTS)**.
- **Main idea of the paper:** build a **logic** on top of that PTS in which we can reason about its programmes.
- The original PTS P = *programming language* (types and terms).
- A new PTS P^2 is constructed as a *logic* whose formulas state properties about programmes in P.

Research Question

How can we systematically build such a logic so that *parametricity* and *realizability* are both internalized in it, and how are these two notions related in the general setting of PTSs?

Parametricity (informal)

Idea

- Polymorphic programs must behave *uniformly* for all type instances.
- Types are interpreted as *relations* on terms; a program is parametric if it *preserves* these relations.
- Example (Haskell-style type):

```
f :: forall a. [a] -> [a]
```

Parametricity says that f cannot depend on the concrete type a, only on the structure of the list.

Realizability (informal)

Idea

- Realizability connects *formulas* with *programs* that witness their truth.
- A realizer for $A \wedge B$ can be seen as a pair of programs (p_A, p_B) realizing A and B.
- A realizer for $A \to B$ is a program that turns any realizer of A into a realizer of B.

The First Level: Pure Type Systems

Pure Type System (PTS)

A PTS is given by a specification $(\mathcal{S}, \mathcal{A}, \mathcal{R})$:

- $\mathcal{S}$: a set of *sorts* (e.g. $\star, \Box$).
- $\mathcal{A} \subseteq \mathcal{S} \times S$: *axioms.*
  $(s_1, s_2) \in A$ expresses that s_1 has sort s_2.
- $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$: *rules* for product types.
  $(s_1, s_2, s_3) \in \mathcal{R}$ determines the sort of Π-types with domain of sort s_1 and codomain of sort s_2.

Typing judgements have the form $\Gamma \vdash A : B$.
We write $\Gamma \vdash A : B : C$ as shorthand for both $\Gamma \vdash A : B$ and $\Gamma \vdash B : C$.

The First Level: Pure Type Systems

Pure Type System (PTS)

A PTS is given by a specification $(\mathcal{S}, \mathcal{A}, \mathcal{R})$:

- $\mathcal{S}$: a set of *sorts* (e.g. $\star, \square$).
- $\mathcal{A} \subseteq \mathcal{S} \times S$: *axioms.*
  $(s_1, s_2) \in A$ expresses that s_1 has sort s_2.
- $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$: *rules* for product types.
  $(s_1, s_2, s_3) \in \mathcal{R}$ determines the sort of Π-types with domain of sort s_1 and codomain of sort s_2.

Typing judgements have the form $\Gamma \vdash A : B$.
We write $\Gamma \vdash A : B : C$ as shorthand for both $\Gamma \vdash A : B$ and $\Gamma \vdash B : C$.

Example: λ_2 (System F) as a PTS

$$S_{\lambda_2} = \{\star, \square\}, \quad A_{\lambda_2} = \{(\star, \square)\}, \quad R_{\lambda_2} = \{(\star, \star, \star), (\square, \star, \star)\}.$$

Reminder: λ_2 (System F) from the Course

Polymorphic lambda calculus λ_2

- Types:

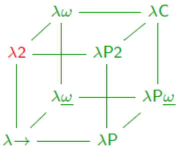
$$A, B ::= a \mid A \to B \mid \forall a.\, A$$

- Terms (Church-style):

$$M, N ::= x \mid MN \mid \lambda x : A.\, M \mid MA \mid \Lambda a.\, M$$

- Polymorphic identity:

$$id := \lambda a : *.\, \lambda x : a.\, x \quad : \quad \Pi a : *.\, a \to a.$$



λ_2 as a PTS: Reading the Rules in R

Specification for λ_2

$$S_{\lambda_2} = \{\star, \square\}, \quad A_{\lambda_2} = \{(\star, \square)\}, \quad R_{\lambda_2} = \{(\star, \star, \star), (\square, \star, \star)\}.$$

The rule $(\star, \star, \star)$: arrow types

$$\frac{\Gamma \vdash A : \star \qquad \Gamma, x : A \vdash B : \star}{\Gamma \vdash \Pi x : A.\, B : \star} \quad (\star, \star, \star) \in R$$

- Domain A has sort $\star$ (ordinary type), Codomain B has sort $\star$.
- Resulting Π-type also has sort $\star$.
- If B does not depend on x, then

$$\Pi x : A.\, B \;\equiv\; A \to B.$$

This rule *gives us arrow types.*

λ_2 as a PTS: Polymorphic Types

The rule $(\Box, \star, \star)$: polymorphism

$$\frac{\Gamma \vdash A : \Box \qquad \Gamma, \alpha : A \vdash B : \star}{\Gamma \vdash \Pi\alpha : A.\, B : \star} \quad (\Box, \star, \star) \in R$$

- Bound variable α has sort $\Box$ (a *type-level* variable).
- Body B has sort $\star$ (an ordinary type).
- The resulting Π-type again has sort $\star$.
- We are binding a *type variable*, so

$$\Pi\alpha : \Box.\, B \;\equiv\; \forall\alpha.\, B.$$

This rule *gives us polymorphic types*.

Sort-Annotated Terms

Sort Annotations

- Variables come with a *sort tag*: for each sort $s \in S$ we have a set V_s of variables of sort s.
- We write x^s to mean: $x \in V_s$ (so x is a variable of sort s).
- Typing in the context looks like: $x^s : A$ with $\Gamma \vdash A : s$.

Sort-Annotated Terms

Sort Annotations

- Variables come with a *sort tag*: for each sort $s \in S$ we have a set V_s of variables of sort s.
- We write x^s to mean: $x \in V_s$ (so x is a variable of sort s).
- Typing in the context looks like: $x^s : A$ with $\Gamma \vdash A : s$.

Product Rule for λ_2

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, x^{s_1} : A \vdash B : s_2}{\Gamma \vdash (\Pi x^{s_1} : A.\, B) : s_3} \quad \text{if } (s_1, s_2, s_3) \in R$$

Sort-Annotated Terms

Sort Annotations

- Variables come with a *sort tag*: for each sort $s \in S$ we have a set V_s of variables of sort s.
- We write x^s to mean: $x \in V_s$ (so x is a variable of sort s).
- Typing in the context looks like: $x^s : A$ with $\Gamma \vdash A : s$.

Product Rule for λ_2

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, x^{s_1} : A \vdash B : s_2}{\Gamma \vdash (\Pi x^{s_1} : A.\, B) : s_3} \quad \text{if } (s_1, s_2, s_3) \in R$$

Application Rule for λ_2

$$\frac{\Gamma \vdash F : \Pi x^s : A.\ B \qquad \Gamma \vdash a : A}{\Gamma \vdash (F\ a)^s : B[x \mapsto a]}$$

λ_2 Terms: Examples

Conventions for λ_2

- Term variables $x, y, z, \ldots$ range over $V_\star$.
- Type variables $\alpha, \beta, \gamma, \ldots$ range over $V_\Box$.

λ_2 Terms: Examples

Conventions for λ_2

- Term variables $x, y, z, \ldots$ range over $V_\star$.
- Type variables $\alpha, \beta, \gamma, \ldots$ range over $V_\Box$.

Identity (term) and Unit (type)

$$Unit \equiv \Pi\alpha : \star.\, \alpha \to \alpha$$

$$Id \equiv \lambda(\alpha : \star)(x : \alpha).\, x \qquad \text{with} \qquad \vdash Id : Unit.$$

λ_2 Terms: Examples

Conventions for λ_2

- Term variables $x, y, z, \dots$ range over $V_\star$.
- Type variables $\alpha, \beta, \gamma, \dots$ range over $V_\square$.

Identity (term) and Unit (type)

$$Unit \equiv \Pi\alpha : \star.\, \alpha \to \alpha$$

$$Id \equiv \lambda(\alpha : \star)(x : \alpha).\, x \qquad \text{with} \qquad \vdash Id : Unit.$$

Church Numerals

$$Nat \equiv \Pi\alpha : \star.\, (\alpha \to \alpha) \to (\alpha \to \alpha)$$

$$0 \equiv \lambda(\alpha : \star)(f : \alpha \to \alpha)(x : \alpha).\, x \qquad \text{with} \qquad \vdash 0 : Nat.$$

$$Succ \equiv \lambda(n : Nat)(\alpha : \star)(f : \alpha \to \alpha)(x : \alpha).\, f\,(n\,\alpha\,f\,x) \quad : \quad Nat \to Nat.$$

Constructing the logic

We want to build a logic *strong enough* to reason about our PTS.

Note

In λ_2, we want to express formulas like $\forall \alpha : \star . A$, whereas we do not need this in $\lambda \rightarrow$. This means not every PTS needs the same logic.

- The more powerful the PTS, the more powerful the logic.
- Idea: use the PTS itself to construct the logic
- Thanks to Curry-Howard, a PTS is already a logic, we just need to add some extra things. This works for *any* PTS.

Question

How exactly do we build this logic?

Idea of the construction

- We denote the logic of a PTS P by P^2.
- P^2 is a PTS, so we only need to specify sorts, axioms and rules
- We want to keep a copy of P inside P^2

Sorts

The PTS λ_2 consists of two sorts: $\star$ and $\square$. In its logic λ_2^2, we extend this with the sorts $\lceil\star\rceil$ and $\lceil\square\rceil$.

- $\lceil\star\rceil$ is the sort of all propositions
- $\lceil\square\rceil$ is the sort of propositions $\lceil\star\rceil$, as well as predicates $\tau \to \lceil\star\rceil$ and relations $\tau_1 \to \cdots \to \tau_n \to \lceil\star\rceil$

Axioms

We have the axioms $\star : \square$ and $\lceil\star\rceil : \lceil\square\rceil$

The rules of λ_2^2

Recall: product rule for PTS

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

λ_2 has rules $(\star, \star, \star)$ and $(\square, \star, \star)$.

- We want a copy of λ_2, so we need $(\star, \star, \star)$ and $(\square, \star, \star)$.
- We also add $(\lceil\star\rceil, \lceil\star\rceil, \lceil\star\rceil)$ and $(\lceil\square\rceil, \lceil\star\rceil, \lceil\star\rceil)$.

Example formula: reflexivity

$$\Pi\alpha : \star.\Pi x : \alpha.x =_\alpha x$$

This means we also need to add the rules $(\square, \lceil\star\rceil, \lceil\star\rceil)$ for quantifying over types and $(\star, \lceil\star\rceil, \lceil\star\rceil)$ for quantifying over programs.
Finally, we need the rule $(\star, \lceil\square\rceil, \lceil\square\rceil)$ to construct predicates $\tau \to \lceil\star\rceil$ etc.

Overview of λ_2^2

PTS for λ_2^2

λ_2
$\mathcal{S} = \{\star, \square\}$
$\mathcal{A} = \{(\star, \square)\}$
$\mathcal{R} = \{(\star, \star, \star), (\square, \star, \star)\}$

λ_2^2
$\mathcal{S} = \{\star, \square, \lceil\star\rceil, \lceil\square\rceil\}$
$\mathcal{A} = \{(\star, \square), (\lceil\star\rceil, \lceil\square\rceil)\}$
$\mathcal{R} = \{(\star, \star, \star), (\square, \star, \star),$
$(\lceil\star\rceil, \lceil\star\rceil, \lceil\star\rceil), (\lceil\square\rceil, \lceil\star\rceil, \lceil\star\rceil),$
$(\star, \lceil\star\rceil, \lceil\star\rceil), (\square, \lceil\star\rceil, \lceil\star\rceil),$
$(\star, \lceil\square\rceil, \lceil\square\rceil)\}$

These rules allow us to type the reflexivity formula $\Pi\alpha : \star.\Pi x : \alpha.x =_\alpha x$.

Rules of P^2 for a general PTS P

Let $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ be the PTS P.

Definition

We define the PTS P^2 as $(\mathcal{S}^2, \mathcal{A}^2, \mathcal{R}^2)$ where

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

In general, for any sort s of P, we think of $\lceil s \rceil$ as the sort of *formulas expressing properties of inhabitants of* s.

Seperation

Let P be an arbitrary PTS.

Theorem 1 (seperation)

If a *program* or *type* is typable in P^2, then it is also typable in P. More formally, for a sort $s \in S$ we have that if $\Gamma \vdash A : B : s$, then there is a subcontext $\Gamma' \subseteq \Gamma$ such that $\Gamma' \vdash_P A : B : s$

Seperation

Let P be an arbitrary PTS.

Theorem 1 (seperation)

If a *program* or *type* is typable in P^2, then it is also typable in P. More formally, for a sort $s \in S$ we have that if $\Gamma \vdash A : B : s$, then there is a subcontext $\Gamma' \subseteq \Gamma$ such that $\Gamma' \vdash_P A : B : s$

Recall our rules:

$$\begin{aligned} R^2 = R &\cup \{(\lceil s_1 \rceil, \lceil s_2 \rceil, \lceil s_3 \rceil) \mid (s_1, s_2, s_3) \in R\} \\ &\cup \{(s_1, \lceil s_3 \rceil, \lceil s_3 \rceil) \mid (s_1, s_2, s_3) \in R\} \\ &\cup \{(s_1, \lceil s_2 \rceil, \lceil s_2 \rceil) \mid (s_1, s_2) \in A\} \end{aligned}$$

The proof is by induction on the structure of the program (or type).

Lifting

We have defined $\lceil s \rceil$ for any sort s, which we call the *lifting* of a sort. Similarly, we define the lift of any expression. Nothing really changes, except the naming convention of variables, and the change of s to $\lceil s \rceil$

Example

The type

$$\mathsf{Nat} \equiv \Pi\alpha : \star.(\alpha \to \alpha) \to (\alpha \to \alpha)$$

gets lifted to

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

Any inhabited type gets lifted to a logical tautology, since the inhabitants get lifted to proofs.

Syntactical rules for lifting

Rules for lifting of expressions (top) and contexts (bottom)

- Note that $\mathring{x}$ is the *renamed* variant of the variable x, for example renaming α to X.
- Structure remains the same

$$\lceil x \rceil = \mathring{x}$$
$$\lceil s \rceil = \lceil s \rceil$$
$$\lceil \Pi x : A.B \rceil = \Pi \mathring{x} : \lceil A \rceil . \lceil B \rceil$$
$$\lceil \lambda x : A.b \rceil = \lambda \mathring{x} : \lceil A \rceil . \lceil b \rceil$$
$$\lceil AB \rceil = \lceil A \rceil \lceil B \rceil$$

$$\lceil <> \rceil = <>$$
$$\lceil \Gamma, x : A \rceil = \lceil \Gamma \rceil , \mathring{x} : \lceil A \rceil$$

Lemmas about lifting

Extending the lifting from sorts to expressions leads to nice lemmas

Lemma 1 (lifting preserves typing)

For all expressions A, B and sorts s in P we have

$$\Gamma \vdash_P A : B : s \implies \lceil \Gamma \rceil \vdash_{P^2} \lceil A \rceil : \lceil B \rceil : \lceil s \rceil$$

This is because the logic P^2 contains a lifted copy of all sorts, axioms and rules of the PTS P.

Lemmas about lifting

Extending the lifting from sorts to expressions leads to nice lemmas

Lemma 1 (lifting preserves typing)

For all expressions A, B and sorts s in P we have

$$\Gamma \vdash_P A : B : s \implies \lceil \Gamma \rceil \vdash_{P^2} \lceil A \rceil : \lceil B \rceil : \lceil s \rceil$$

This is because the logic P^2 contains a lifted copy of all sorts, axioms and rules of the PTS P.

Lemma 2 (lifting preserves β-reduction)

If $A \longrightarrow_\beta B$, then $\lceil A \rceil \longrightarrow_\beta \lceil B \rceil$

Formal proof by induction, informally true because lifting only renames variables and lifts sorts.

Projection

- Projection maps second-level terms in P^2 to first-level terms in P.
- It removes all second-level constructs (formulas, proofs) and all interactions between levels.
- First-level subterms are erased.
- Projection is defined only on second-level terms.

Renaming convention

Variables $x^{\lceil s \rceil}$ are renamed to $\dot{x}^s$. This renaming cancels the one introduced by lifting.

Projection: Examples

Examples in F^2

$$\lfloor \top \rfloor = \text{Unit} \qquad \lfloor \mathit{Obvious} \rfloor = \mathit{Id}$$

$$\lfloor \Pi(\alpha : \star)(x : \alpha).\, x =_\alpha x \rfloor = \text{Unit} \qquad \lfloor N\, t \rfloor = \mathit{Nat}$$

- $\top \equiv \Pi X : \lceil \star \rceil.\, X \to X$ is a logical tautology; erasing the logical layer yields the first-level type $\text{Unit} \equiv \Pi\alpha : \star.\, \alpha \to \alpha$.
- *Obvious* is a proof of $\top$; under projection, proofs are erased, yielding the corresponding first-level program *Id*.
- Leibniz equality $x =_\alpha x$ is purely logical; its reflexivity carries no computational content after projection.
- $N\, t$ is a predicate over *Nat*; projection removes the predicate layer and retains its domain.

Examples in detail

Leibniz

$$\left\lfloor \Pi\alpha^\star : \star.\, \Pi x^\star : \alpha.\, (x =_\alpha x) \right\rfloor = \left\lfloor \Pi x^\star : \alpha.\, (x =_\alpha x) \right| B \rfloor)$$
$$= \lfloor x =_\alpha x \rfloor$$
$$= \mathsf{Unit}$$

Examples in detail

Leibniz

$$\left\lfloor \Pi\alpha^\star : \star.\ \Pi x^\star : \alpha.\ (x =_\alpha x) \right\rfloor = \left\lfloor \Pi x^\star : \alpha.\ (x =_\alpha x) \right\rfloor B \rfloor)$$
$$= \lfloor x =_\alpha x \rfloor$$
$$= \mathsf{Unit}$$

N t

$$\lfloor N\, t \rfloor = \lfloor (N\, t)_{Nat} \rfloor$$
$$= \lfloor N \rfloor \qquad (\text{rule: } \lfloor (A\, B)_s \rfloor = \lfloor A \rfloor)$$
$$= Nat.$$

Lemma 3 and 4

Lemma 3: projection is the left inverse of lifting

For any first-level term A,

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

Lemma 3 and 4

Lemma 3: projection is the left inverse of lifting

For any first-level term A,

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

Lemma 4: projection preserves typing

If

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

then

$$\lfloor \Gamma \rfloor \vdash \lfloor A \rfloor : \lfloor B \rfloor : s.$$

Example: Lemma 4 in F^2

The formula $\top$ and its proof

In F^2, truth is defined as $\top \equiv \Pi X : \lceil * \rceil .\, X \to X$ and is proved by Obvious $\equiv \lambda(X : \lceil * \rceil)(h : X).\, h$
with typing judgement

$$\vdash \text{Obvious} : \top : \lceil * \rceil .$$

Projection (Lemma 4)

Applying projection removes the second level:

$$\lfloor \top \rfloor = \text{Unit} \qquad \lfloor \text{Obvious} \rfloor = \text{Id}$$

hence

$$\vdash \text{Id} : \text{Unit} : * .$$

Lemma 5

Lemma 5 (β-reduction)

If $A \to_\beta B$, then either

$$\lfloor A \rfloor \to_\beta \lfloor B \rfloor \quad \text{or} \quad \lfloor A \rfloor = \lfloor B \rfloor.$$

Case 1

$$A \equiv (\lambda X : \lceil * \rceil .\, t)\, \top \qquad \to_\beta \qquad B \equiv t[\top / X].$$

$$\begin{aligned} \lfloor (\lambda X : \lceil * \rceil .\, t)\, \top \rfloor &= (\lambda \alpha : * .\, \lfloor t \rfloor)\, \lfloor \top \rfloor \\ &= (\lambda \alpha : * .\, \lfloor t \rfloor)\ \text{Unit} \qquad \text{and} \qquad \lfloor t[\top / X] \rfloor = \lfloor t \rfloor. \\ &\to_\beta \lfloor t \rfloor \end{aligned}$$

$$\boxed{\lfloor A \rfloor = \lfloor (\lambda \alpha : * .\, \lfloor t \rfloor)\, \top \rfloor \to_\beta \lfloor t \rfloor = \lfloor B \rfloor.}$$

Lemma 5: β-reduction erased by projection

Case 2

Take $A = (\lambda x^* : \alpha.\, x)\; y$ and $B = y$ since they are both first-level terms, projection acts like this:

$$\lfloor (\lambda x^* : \alpha.\, x)\; y \rfloor \;=\; y$$

$$\lfloor y \rfloor \;=\; y.$$

After projection, the β-reduction is erased:

$$\lfloor A \rfloor \;=\; \lfloor (\lambda x^* : \alpha.\, x) \rfloor\; y \quad = \quad y \;=\; \lfloor B \rfloor.$$

Rules for Projection

Projection rules

$$
\begin{array}{rcl}
\lfloor x^{\lceil s \rceil} \rfloor & = & \dot{x}^s \\
\lfloor \lceil s \rceil \rfloor & = & s \\
\lfloor \Pi x^s : A.\, B \rfloor & = & \lfloor B \rfloor \\
\lfloor \Pi x^{\lceil s \rceil} : A.\, B \rfloor & = & \Pi \dot{x}^s : \lfloor A \rfloor.\, \lfloor B \rfloor \\
\lfloor \lambda x^s : A.\, B \rfloor & = & \lfloor B \rfloor \\
\lfloor \lambda x^{\lceil s \rceil} : A.\, B \rfloor & = & \lambda \dot{x}^s : \lfloor A \rfloor.\, \lfloor B \rfloor \\
\lfloor (A\, B)_s \rfloor & = & \lfloor A \rfloor \\
\lfloor (A\, B)_{\lceil s \rceil} \rfloor & = & \lfloor A \rfloor \, \lfloor B \rfloor \\
\hline
\lfloor <> \rfloor & = & <> \\
\lfloor \Gamma, x^s : A \rfloor & = & \lfloor \Gamma \rfloor \\
\lfloor \Gamma, x^{\lceil s \rceil} : A \rfloor & = & \lfloor \Gamma \rfloor, \dot{x}^s : \lfloor A \rfloor
\end{array}
$$

Strong Normalization

Theorem 2 (normalization)

If P is strongly normalizing, so is P^2

Proof

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

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

Strong Normalization

Theorem 2 (normalization)

If P is strongly normalizing, so is P^2

Proof

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

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

Contradiction

- By Lemma 4, the projection $\lfloor A \rfloor$ is a well-typed term of P.
- By Lemma 5, projection does not introduce new β-reduction.
- Hence, any non-termination in P^2 yields a non-terminating term of P.
- This contradicts strong normalization of P.

Recap

- We considered Pure Type Systems, and constructed a logic for every PTS
- In this logic, there are formulas which can refer to the programs and types from the PTS
- We proved that the levels are seperated, and considered lifting and projection to convert between the levels
- Finally, we looked at the proof that if the PTS P is strongly normalizing, then P^2 is as well.