Pure Type Systems

- Pure Type Systems (PTS) provide a general description for a large class of typed λ-calculi.
- PTS make it possible to derive lot of meta theoretic properties in a generic way.
- In PTS we only have one type constructor (Π) and one computation rule (β). (Therefore the name "pure").
- PTS were originally introduced (albeit in a different from) by S. Berardi and J. Terlouw as a generalization of Barendregt's λ-cube, which itself provides a fine-grained analysis of the Calculus of Constructions.

Pure Type Systems

PTS are formal systems for deriving judgments of the form

$$\Gamma \vdash M : A$$

context — list of variable declarations

pseudo-terms

M is of type A relative to a typing of the free variables of M and A (which are declared in Γ)

Syntax

PTS have a single category of expressions, which are called **pseudo-terms**.

The definitions of pseudo-terms is parameterized by a set $\mathcal{V}$ of **variables** and a set $\mathcal{S}$ of **sorts** (constants that denote the universes of the type system).

Definition

The set $\mathcal{T}$ of **pseudo-terms** are defined by the abstract syntax

$$\mathcal{T} \; ::= \; \mathcal{S} \mid \mathcal{V} \mid \mathcal{T}\,\mathcal{T} \mid \lambda\mathcal{V}{:}\mathcal{T}.\mathcal{T} \mid \Pi\mathcal{V}{:}\mathcal{T}.\mathcal{T}$$

Both Π and λ bind variables.
We have the usual notation for free variables and bound variables.

Definitions

Pseudo-terms inherit much of the standard definitions and notations of λ-calculi.

- FV(M) denotes the set of free variables of the pseudo-term M .
- We write $A \to B$ instead of $\Pi\, x : A.\, B$ whenever $x \notin$ FV(B).
- $M\,[x := N\,]$ denotes the substitution of N for all the free occurrences of x in M .
- We identify pseudo-terms that are equal up to a renaming of bound variables (**α-conversion**).
- We assume the standard variable convention, so all bound variables are chosen to be different from free variables.

Definitions

- **β-reduction** is defined as the compatible closure of the rule

$$(\lambda x{:}A.M)\,N \quad \to_\beta \quad M[x := N]$$

$\twoheadrightarrow_\beta$ is the reflexive-transitive closure of $\to_\beta$

$=_\beta$ is the reflexive-symmetric-transitive closure of $\to_\beta$

- Application associates to the left, abstraction to the right and application binds more tightly than abstraction.

- We let x, y, z, ... range over $\mathcal{V}$ and s, s', ... range over S

Salient Features of PTS

- PTS describe λ-calculi à la Church (λ-abstractions carry the domain of bound variables).

- PTS are minimal (just Π type construction and β reduction rule), which imposes strict limitations on their applicability.

- PTS model dependent types. Type constructor Π captures in the type theory the set-theoretic notion of generic or dependent function space.

Dependent types

In the type theory one can define for every set A and A-indexed family of sets $(B_a)_{x\in A}$ a new set $\Pi_{x\in A}B_x$ called dependent function space.

Elements of $\Pi_{x\in A}B_x$ are functions with domain A and such that $f(a) \in B_a$ for every $a \in A$.

Π-construction of PTS works in the same way:

$\Pi\, x{:}A.\,B(x)$ is the type of terms F such that, for every $a : A,\ F\,a : B(a)$

Specifications

The typing system of PTS is parameterized by a triple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ where

$\mathcal{S}$ is the set of universes of the type system;
$\mathcal{A}$ determine the typing relation between universes;
$\mathcal{R}$ determine which dependent function types may be found and where they live.

Definition

A PTS-**specification** is a triple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ where

- $\mathcal{S}$ is a set of **sorts**
- $\mathcal{A} \subseteq S \times S$ is a set of **axioms**
- $\mathcal{R} \subseteq S \times S \times S$ is a set of **rules**

We use $(s1,s2)$ to denote rules of the form $(s1,s2,s2)$.

Every specification $\mathbf{S}$ induces a PTS $\lambda\mathbf{S}$.

Contexts and Judgments

- The set $\mathcal{G}$ of **contexts** is given by the abstract syntax $\mathcal{G} ::= \langle\rangle \mid \mathcal{G}, \mathcal{V} : \mathcal{T}$
  - We let $\subseteq$ denote context inclusion
  - The domain of a context is defined by the clause

    $$\mathsf{dom}(x_1 : A_1, ..., x_n : A_n) = \{x_1, ..., x_n\}$$

  - We let Γ, Δ range over $\mathcal{G}$
- A **judgment** is a triple of the form $\Gamma \vdash A : B$ where $A, B \in \mathcal{T}$ and $\Gamma \in \mathcal{G}$.
- A judgment is **derivable** if it can be inferred from the typing rules of the next slide.
  - If $\Gamma \vdash A : B$ then Γ, A and B are legal.
  - If $\Gamma \vdash A : s$ for $s \in S$, we say that A is a type.

Typing rules for PTS

(axiom)	$\langle\rangle \vdash s_1 : s_2$	if $(s_1, s_2) \in \mathcal{A}$
(start)	$\dfrac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A}$	if $x \notin \mathsf{dom}(\Gamma)$
(weakening)	$\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x{:}C \vdash A : B}$	if $x \notin \mathsf{dom}(\Gamma)$
(product)	$\dfrac{\Gamma \vdash A : s_1 \quad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash (\Pi x{:}A.\, B) : s_3}$	if $(s_1, s_2, s_3) \in \mathcal{R}$
(application)	$\dfrac{\Gamma \vdash F : (\Pi x{:}A.\, B) \quad \Gamma \vdash a : A}{\Gamma \vdash F\, a : B[x := a]}$	
(abstraction)	$\dfrac{\Gamma, x{:}A \vdash b : B \quad \Gamma \vdash (\Pi x{:}A.\, B) : s}{\Gamma \vdash \lambda x{:}A.b : (\Pi x{:}A.\, B)}$	
(conversion)	$\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s}{\Gamma \vdash A : B'}$	if $B =_\beta B'$

Typing rules for PTS

$$(\text{axiom}) \quad \langle\rangle \vdash s_1 : s_2 \qquad \text{if } (s_1, s_2) \in \mathcal{A}$$

It embeds the relation $\mathcal{A}$ into the type system.

Typing rules for PTS

(start)	$\dfrac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A}$	if $x \notin \mathsf{dom}(\Gamma)$
(weakening)	$\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x{:}C \vdash A : B}$	if $x \notin \mathsf{dom}(\Gamma)$

It allows the introduction of variables in a context.

Typing rules for PTS

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

It allows for dependent function types to be formed, provided they match the rule in $\mathcal{R}$.

Typing rules for PTS

$$(\text{application}) \quad \frac{\Gamma \vdash F : (\Pi\, x\!:\!A.\, B) \quad \Gamma \vdash a : A}{\Gamma \vdash F\, a : B[x := a]}$$

It allows to form applications.

Note substitution $[x := a]$ in the type of the application, in order to accommodate type dependencies.

Typing rules for PTS

$$(\text{abstraction}) \quad \frac{\Gamma, x\!:\!A \vdash b : B \quad \Gamma \vdash (\Pi\, x\!:\!A.\, B) : s}{\Gamma \vdash \lambda\, x\!:\!A.b : (\Pi\, x\!:\!A.\, B)}$$

It allows to build λ-abstractions.

Note that the side condition requires that the dependent function type is well formed.

Typing rules for PTS

$$(\text{conversion}) \quad \frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s}{\Gamma \vdash A : B'} \quad \text{if } B =_\beta B'$$

It ensures that convertible types (i.e. types that are β-equal) have the same inhabitants.

This rule is crucial for higher-order type theories, because types are λ-terms and can be reduced, and for dependent type theories because they may occur in types.

Examples of PTS

Non-dependent type systems (i.e. an expression $M : A$ with $A : *$ cannot appear as a subexpression of $B : *$)

λ→, the simply typed λ-calculus.

$\lambda\!\to$	$\mathcal{S}$	$=$ $*,\ \Box$
	$\mathcal{A}$	$=$ $(* : \Box)$
	$\mathcal{R}$	$=$ $(*, *)$

λ2 is the PTS counterpart of Girard's System F.

$\lambda 2$	$\mathcal{S}$	$=$ $*,\ \Box$
	$\mathcal{A}$	$=$ $(* : \Box)$
	$\mathcal{R}$	$=$ $(*, *),\ (\Box, *)$

λω is the PTS counterpart of Girard's System Fω.

$\lambda\omega$	$\mathcal{S}$	$=$ $*,\ \Box$
	$\mathcal{A}$	$=$ $(* : \Box)$
	$\mathcal{R}$	$=$ $(*, *),\ (\Box, *),\ (\Box, \Box)$

In logical terms, these non-dependent systems correspond to propositional logics.

More examples of non-dependent PTS

λU⁻, Girard's System U⁻

λU^-	$\mathcal{S}$	$=$ $*,\ \Box,\ \triangle$
	$\mathcal{A}$	$=$ $(* : \Box),\ (\Box : \triangle)$
	$\mathcal{R}$	$=$ $(*, *),\ (\Box, *),\ (\Box, \Box),\ (\triangle, \Box)$

λU , System U

λU	$\mathcal{S}$	$=$ $*,\ \Box,\ \triangle$
	$\mathcal{A}$	$=$ $(* : \Box),\ (\Box : \triangle)$
	$\mathcal{R}$	$=$ $(*, *),\ (\Box, *),\ (\Box, \Box),\ (\triangle, *),\ (\triangle, \Box)$

The System **λ***

$\lambda *$	$\mathcal{S}$	$=$ $*$
	$\mathcal{A}$	$=$ $(* : *)$
	$\mathcal{R}$	$=$ $(*, *)$

λU⁻, λU and λ* are **inconsistent** in the sense that there exists a pseudo-term M such that the judgment $A : * \vdash M : A$ is derivable.

Examples of dependent PTS

It is possible to type expressions $B : *$ which contain as subexpression $M : A : *$.

λP is the PTS counterpart of the Logical Frameworks due to Harper et al.

λP	$\mathcal{S}$	$=$ $*,\ \Box$
	$\mathcal{A}$	$=$ $(* : \Box)$
	$\mathcal{R}$	$=$ $(*, *),\ (*, \Box)$

λP2 is the PTS counterpart of Longo and Moggi's system also named λP2.

$\lambda P2$	$\mathcal{S}$	$=$ $*,\ \Box$
	$\mathcal{A}$	$=$ $(* : \Box)$
	$\mathcal{R}$	$=$ $(*, *),\ (\Box, *),\ (*, \Box)$

λC (also known as **λPω**) is the PTS counterpart of Coquand and Huet's Calculus of Constructions.

λC	$\mathcal{S}$	$=$ $*,\ \Box$
	$\mathcal{A}$	$=$ $(* : \Box)$
	$\mathcal{R}$	$=$ $(*, *),\ (\Box, *),\ (*, \Box),\ (\Box, \Box)$

In logical terms, these dependent systems correspond to predicate logics.

Another example of dependent PTS

λCω is an extension of the Calculus os Constructions.

λC^ω	$\mathcal{S}$	$=$ $*,\ \Box_i \quad ,\ i \in \mathrm{N}$
	$\mathcal{A}$	$=$ $(* : \Box_0),\ (\Box_i : \Box_{i+1}) \quad ,\ i \in \mathrm{N}$
	$\mathcal{R}$	$=$ $(*, *),\ (\Box_i, *),\ (*, \Box_i),\ (\Box_i, \Box_j, \Box_{\max(i,j)}) \quad ,\ i, j \in \mathrm{N}$