Pure Type Systems (PTS) – syntax

In a PTS type expressions and term expressions are lumped together into a single syntactic category of *pseudo-terms*:

will also use $A, B, \ldots$ $M, N, \ldots$ to stand for pseudo-terms

$$\boxed{\begin{array}{llll} t & ::= & x & \text{variable} \\ & | & s & \text{sort} \\ & | & \Pi x : t\,(t) & \text{dependent function type} \\ & | & \lambda x : t\,(t) & \text{function abstraction} \\ & | & t\,t & \text{function application} \end{array}}$$

where x ranges over a countably infinite set $\mathbf{Var}$ of variables and s ranges over a disjoint set $\mathbf{Sort}$ of *sort symbols* – constants that denote various universes (= types whose elements denote types of various sorts) [*kind* is a commonly used synonym for *sort*]. $\lambda x : t\,(t')$ and $\Pi x : t\,(t')$ both bind free occurrences of x in the pseudo-term t'.

$\boxed{t[t'/x]}$ denotes result of capture-avoiding substitution of t' for all free occurrences of x in t.

$\boxed{t \to t} \triangleq \Pi x : t\,(t')$ where $x \notin fv(t')$.

Pure Type Systems – specifications

The typing rules for a particular PTS are parameterised by a *specification* $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ where:

- $\mathcal{S} \subseteq \mathbf{Sort}$

  Elements $s \in \mathcal{S}$ denote the different universes of types in this PTS.
- $\mathcal{A} \subseteq \mathbf{Sort} \times \mathbf{Sort}$

  Elements $(s_1, s_2) \in \mathcal{A}$ are called *axioms*. They determine the typing relation between universes in this PTS.
- $\mathcal{R} \subseteq \mathbf{Sort} \times \mathbf{Sort} \times \mathbf{Sort}$

  Elements $(s_1, s_2, s_3) \in \mathcal{R}$ are called rules. They determine which kinds of dependent function can be formed and in which universes they live.

Pure Type Systems – specifications

The typing rules for a particular PTS are parameterised by a *specification* $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ where:

- $\mathcal{S} \subseteq \mathbf{Sort}$

  Elements $s \in \mathcal{S}$ denote the different universes of types in this PTS.
- $\mathcal{A} \subseteq \mathbf{Sort} \times \mathbf{Sort}$

  Elements $(s_1, s_2) \in \mathcal{A}$ are called *axioms*. They determine the typing relation between universes in this PTS.
- $\mathcal{R} \subseteq \mathbf{Sort} \times \mathbf{Sort} \times \mathbf{Sort}$

  Elements $(s_1, s_2, s_3) \in \mathcal{R}$ are called rules. They determine which kinds of dependent function can be formed and in which universes they live.

The PTS with specification $\mathbf{S}$ will be denoted $\boxed{\lambda\mathbf{S}}$.

Pure Type Systems – typing judgements

take the form

$$\boxed{\Gamma \vdash t : t'}$$

where t, t' are pseudo-terms and Γ is a *context*, a form of typing environment given by the grammar

$$\Gamma ::= \diamond \mid \Gamma, x : t$$

(Thus contexts are finite ordered lists of (variable,pseudo-term)-pairs, with the empty list denoted $\diamond$, the head of the list on the right and list-cons denoted by $\_,\_$. Unlike previous type systems in this course, *the order in which typing declarations* $x : t$ *occur in a context is important*.)
A typing judgement is *derivable* if it is in the set inductively generated by the rules on the next slide, which are parameterised by a given specification $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$.

Pure Type Systems – typing rules

for a given specification

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

$$\textbf{(axiom)}\ \frac{}{\diamond \vdash s_1 : s_2} \text{ if } (s_1, s_2) \in \mathcal{A}$$

Pure Type Systems – typing rules

$$\textbf{(axiom)}\ \frac{}{\diamond \vdash s_1 : s_2} \quad \text{if } (s_1, s_2) \in \mathcal{A}$$

$$\textbf{(start)}\ \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \quad \text{if } x \notin dom(\Gamma)$$

Properties of Pure Type Systems in general

- **Correctness of types.** If $\Gamma \vdash M : A$, then either $A \in \mathcal{S}$, or $\Gamma \vdash A : s$ for some $s \in \mathcal{S}$.

pseudo terms that appear as types, i.e. to the right of _ : _ in a derivable typing judgement, are either sorts, or have a sort

"everything is well-sorted"

Pure Type Systems – typing rules

$$\textbf{(axiom)}\ \frac{}{\diamond \vdash s_1 : s_2} \quad \text{if } (s_1, s_2) \in \mathcal{A}$$

$$\textbf{(start)}\ \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \quad \text{if } x \notin dom(\Gamma)$$

$$\textbf{(weaken)}\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s}{\Gamma, x : B \vdash M : A} \quad \text{if } x \notin dom(\Gamma)$$

Pure Type Systems – typing rules

$$(\textbf{axiom})\ \frac{}{\diamond \vdash s_1 : s_2} \quad \text{if } (s_1, s_2) \in \mathcal{A}$$

$$(\textbf{start})\ \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \quad \text{if } x \notin dom(\Gamma)$$

$$(\textbf{weaken})\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s}{\Gamma, x : B \vdash M : A} \quad \text{if } x \notin dom(\Gamma)$$

$$(\textbf{conv})\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s}{\Gamma \vdash M : B} \quad \text{if } A =_\beta B$$

β- conversion

Pure Type Systems – beta-conversion

- *beta-reduction* of pseudo-terms: $\boxed{t \longrightarrow t'}$ means t' can be obtained from t (up to alpha-conversion, of course) by replacing a subexpression which is a *redex* by its corresponding *reduct*. There is only one form of redex-reduct pair:

$$(\lambda x : t\,(t_1))\,t_2 \longrightarrow t_1[t_2/x]$$

- As usual, $\longrightarrow^*$ denotes the reflexive-transitive closure of $\longrightarrow$.
- *beta-conversion* of pseudo-terms: $=_\beta$ is the reflexive-symmetric-transitive closure of $\longrightarrow$ (i.e. the smallest equivalence relation containing $\longrightarrow$).

Pure Type Systems – typing rules

$$\textbf{(axiom)}\ \frac{}{\diamond \vdash s_1 : s_2} \quad \text{if } (s_1, s_2) \in \mathcal{A}$$

$$\textbf{(start)}\ \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \quad \text{if } x \notin dom(\Gamma)$$

$$\textbf{(weaken)}\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s}{\Gamma, x : B \vdash M : A} \quad \text{if } x \notin dom(\Gamma)$$

$$\textbf{(conv)}\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s}{\Gamma \vdash M : B} \quad \text{if } A =_\beta B$$

needed to ensure "correctness of types" property

Pure Type Systems – typing rules

$$(\textbf{axiom})\ \frac{}{\diamond \vdash s_1 : s_2} \quad \text{if } (s_1, s_2) \in \mathcal{A}$$

$$(\textbf{start})\ \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \quad \text{if } x \notin dom(\Gamma)$$

$$(\textbf{weaken})\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s}{\Gamma, x : B \vdash M : A} \quad \text{if } x \notin dom(\Gamma)$$

$$(\textbf{conv})\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s}{\Gamma \vdash M : B} \quad \text{if } A =_\beta B$$

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

Pure Type Systems – typing rules

$$\textbf{(axiom)}\ \frac{}{\diamond \vdash s_1 : s_2} \quad \text{if } (s_1, s_2) \in \mathcal{A}$$

$$\textbf{(start)}\ \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \quad \text{if } x \notin dom(\Gamma)$$

$$\textbf{(weaken)}\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s}{\Gamma, x : B \vdash M : A} \quad \text{if } x \notin dom(\Gamma)$$

$$\textbf{(conv)}\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s}{\Gamma \vdash M : B} \quad \text{if } A =_\beta B$$

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

$$\textbf{(abs)}\ \frac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash \Pi x : A\ (B) : s}{\Gamma \vdash \lambda x : A\ (M) : \Pi x : A\ (B)}$$

needed to ensure "correctness of types" property

Pure Type Systems – typing rules

$$\textbf{(axiom)}\ \frac{}{\diamond \vdash s_1 : s_2}\ \text{if } (s_1, s_2) \in \mathcal{A}$$

$$\textbf{(start)}\ \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}\ \text{if } x \notin dom(\Gamma)$$

$$\textbf{(weaken)}\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s}{\Gamma, x : B \vdash M : A}\ \text{if } x \notin dom(\Gamma)$$

$$\textbf{(conv)}\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s}{\Gamma \vdash M : B}\ \text{if } A =_\beta B$$

$$\textbf{(prod)}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A\,(B) : s_3}\ \text{if } (s_1, s_2, s_3) \in \mathcal{R}$$

$$\textbf{(abs)}\ \frac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash \Pi x : A\,(B) : s}{\Gamma \vdash \lambda x : A\,(M) : \Pi x : A\,(B)}$$

$$\textbf{(app)}\ \frac{\Gamma \vdash M : \Pi x : A\,(B) \qquad \Gamma \vdash N : A}{\Gamma \vdash M\,N : B[N/x]}$$

Example PTS typing derivations

$$
\textbf{(prod)}\ \dfrac{\textbf{(axiom)}\ \dfrac{}{\diamond \vdash * : \Box} \qquad \textbf{(weaken)}\ \dfrac{\textbf{(axiom)}\ \dfrac{}{\diamond \vdash * : \Box} \qquad \textbf{(axiom)}\ \dfrac{}{\diamond \vdash * : \Box}}{\diamond, x : * \vdash * : \Box}}{\diamond \vdash * \to * : \Box}
$$

$$
\textbf{(abs)}\ \dfrac{\textbf{(start)}\ \dfrac{\textbf{(axiom)}\ \dfrac{}{\diamond \vdash * : \Box}}{\diamond, x : * \vdash x : *} \qquad \dfrac{\vdots}{\diamond \vdash * \to * : \Box}}{\diamond \vdash \lambda x : * \,(x) : * \to *}
$$

Here we assume that the PTS specification $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ has $* \in \mathcal{S}$, $\Box \in \mathcal{S}$, $(*, \Box) \in \mathcal{A}$ and $(\Box, \Box, \Box) \in \mathcal{R}$.
(Recall that $* \to * \triangleq \Pi x : * \,(*)$.)

Agenda

- general properties of PTSs
  (no proofs)
- examples of PTSs

Properties of Pure Type Systems in general

- **Correctness of types.** If $\Gamma \vdash M : A$, then either $A \in \mathcal{S}$, or $\Gamma \vdash A : s$ for some $s \in \mathcal{S}$.
- **Church-Rosser Property** (aka *confluence*). $t =_\beta t'$ iff $\exists u\,(t \to^* u \;\wedge\; t' \to^* u)$
- **Subject Reduction.** If $\Gamma \vdash M : A$ and $M \to M'$, then $\Gamma \vdash M' : A$.
- **Uniqueness of Types.** A PTS specification $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ is said to be *functional* if both $\mathcal{A}$ and $\mathcal{R}_s \triangleq \{(s_2, s_3) \mid (s, s_2, s_3) \in \mathcal{R}\}$ for each $s \in \mathcal{S}$, are single-valued binary relations.
  In this case $\lambda\mathbf{S}$ satisfies: if $\Gamma \vdash M : A$ and $\Gamma \vdash M : B$, then $A =_\beta B$.

Type-checking for a PTS, $\boldsymbol{\lambda}\mathbf{S}$

Recall the *type-checking* and *typeability* problems for a type system.

given Γ, t, t', decide whether or not $\Gamma \vdash t : t'$ holds

given Γ & t, decide whether or not there is some t' with $\Gamma \vdash t : t'$

Pure Type Systems – typing rules

$$\textbf{(axiom)}\ \frac{}{\diamond \vdash s_1 : s_2} \quad \text{if } (s_1, s_2) \in \mathcal{A}$$

$$\textbf{(start)}\ \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \quad \text{if } x \notin dom(\Gamma)$$

$$\textbf{(weaken)}\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s}{\Gamma, x : B \vdash M : A} \quad \text{if } x \notin dom(\Gamma)$$

$$\textbf{(conv)}\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s}{\Gamma \vdash M : B} \quad \text{if } A =_\beta B$$

this rule complicates type-checking & type-inference for PTSs

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

$$\textbf{(abs)}\ \frac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash \Pi x : A\,(B) : s}{\Gamma \vdash \lambda x : A\,(M) : \Pi x : A\,(B)}$$

$$\textbf{(app)}\ \frac{\Gamma \vdash M : \Pi x : A\,(B) \qquad \Gamma \vdash N : A}{\Gamma \vdash M\,N : B[N/x]}$$

(A, B, M, N range over pseudoterms, s, s_1, s_2, s_3 over sort symbols)

Type-checking for a PTS, $\boldsymbol{\lambda}\mathbf{S}$

Definition. A pseudo-term t is *legal* for a PTS specification $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ if either $t \in \mathcal{S}$ or $\Gamma \vdash t : t'$ is derivable in $\boldsymbol{\lambda}\mathbf{S}$ for some Γ and t'.

Recall the *type-checking* and *typeability* problems for a type system.

Fact(van Bentham Jutting): these problems for $\boldsymbol{\lambda}\mathbf{S}$ are decidable if $\mathbf{S}$ is finite and $\boldsymbol{\lambda}\mathbf{S}$ is *normalizing*, meaning that for every legal pseudo-term there is some finite chain of beta-reductions leading to a beta-normal form.

Type-checking for a PTS, $\boldsymbol{\lambda}\mathbf{S}$

Definition. A pseudo-term t is *legal* for a PTS specification $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ if either $t \in \mathcal{S}$ or $\Gamma \vdash t : t'$ is derivable in $\boldsymbol{\lambda}\mathbf{S}$ for some Γ and t'.

Recall the *type-checking* and *typeability* problems for a type system.

Fact(van Bentham Jutting): these problems for $\boldsymbol{\lambda}\mathbf{S}$ are decidable if $\mathbf{S}$ is finite and $\boldsymbol{\lambda}\mathbf{S}$ is *normalizing*, meaning that for every legal pseudo-term there is some finite chain of beta-reductions leading to a beta-normal form.

Fact (Meyer): the problems are undecidable for the PTS $\boldsymbol{\lambda}*$ with specification $\mathcal{S} = \{*\}$, $\mathcal{A} = \{(*, *)\}$ and $\mathcal{R} = \{(*, *, *)\}$.

Agenda

- general properties of PTSs
  (no proofs)
- examples of PTSs

PLC versus the Pure Type System $\boldsymbol{\lambda 2}$

PTS signature:

$$\mathbf{2} \triangleq (\mathcal{S}_2, \mathcal{A}_2, \mathcal{R}_2) \text{ where } \begin{cases} \mathcal{S}_2 & \triangleq & \{*, \Box\} \\ \mathcal{A}_2 & \triangleq & \{(*, \Box)\} \\ \mathcal{R}_2 & \triangleq & \{(*, *, *), (\Box, *, *)\} \end{cases}$$

Claim: $*$ acts like a universe of PLC types in $\lambda 2$

PLC versus the Pure Type System $\boldsymbol{\lambda 2}$

PTS signature:

$$\mathbf{2} \triangleq (\mathcal{S}_2, \mathcal{A}_2, \mathcal{R}_2) \text{ where } \begin{cases} \mathcal{S}_2 & \triangleq & \{*, \square\} \\ \mathcal{A}_2 & \triangleq & \{(*, \square)\} \\ \mathcal{R}_2 & \triangleq & \{(*, *, *), (\square, *, *)\} \end{cases}$$

Claim: $*$ acts like a universe of PLC types in $\lambda 2$

$\forall$-types : (prod) $\dfrac{\Gamma \vdash * : \square \quad \Gamma, \alpha : * \vdash A : *}{\Gamma \vdash \Pi \alpha : * (A) : *}$ $(\square, *, *) \in \mathcal{R}_2$

PLC versus the Pure Type System $\boldsymbol{\lambda 2}$

PTS signature:

$$\mathbf{2} \triangleq (\mathcal{S}_2, \mathcal{A}_2, \mathcal{R}_2) \text{ where } \begin{cases} \mathcal{S}_2 & \triangleq & \{*, \Box\} \\ \mathcal{A}_2 & \triangleq & \{(*, \Box)\} \\ \mathcal{R}_2 & \triangleq & \{(*, *, *), (\Box, *, *)\} \end{cases}$$

Claim: $*$ acts like a universe of PLC types in $\lambda 2$

$\rightarrow$ types: (prod) $\dfrac{\Gamma \vdash A : * \quad \Gamma, x : A \vdash B : *}{\Gamma \vdash \Pi x : A\,(B) : *}$ $(*, *, *) \in R_2$

PLC versus the Pure Type System $\lambda 2$

PTS signature:

$$\mathbf{2} \triangleq (\mathcal{S}_2, \mathcal{A}_2, \mathcal{R}_2) \text{ where } \begin{cases} \mathcal{S}_2 & \triangleq & \{*, \square\} \\ \mathcal{A}_2 & \triangleq & \{(*, \square)\} \\ \mathcal{R}_2 & \triangleq & \{(*, *, *), (\square, *, *)\} \end{cases}$$

Claim: $*$ acts like a universe of PLC types in $\lambda 2$

→ types: (prod) $\dfrac{\Gamma \vdash A : * \qquad \Gamma, x : A \vdash B : *}{\Gamma \vdash \underbrace{\Pi x : A\,(B)}\, : *}$ $(*, *, *) \in \mathcal{R}_2$

in fact can't have $x \in fv(B)$, so this is a simple function type $A \to B$

PLC versus the Pure Type System $\boldsymbol{\lambda 2}$

PTS signature:

$$\mathbf{2} \triangleq (\mathcal{S}_2, \mathcal{A}_2, \mathcal{R}_2) \text{ where } \begin{cases} \mathcal{S}_2 & \triangleq & \{*, \Box\} \\ \mathcal{A}_2 & \triangleq & \{(*, \Box)\} \\ \mathcal{R}_2 & \triangleq & \{(*, *, *), (\Box, *, *)\} \end{cases}$$

Translation of PLC types and terms to $\boldsymbol{\lambda 2}$ pseudo-terms:

$$\begin{aligned}
[\![\alpha]\!] &= \alpha \\
[\![\tau \to \tau']\!] &= \Pi x : [\![\tau]\!]\,([\![\tau']\!]) \quad \leftarrow \text{(any } x \text{ not free in } \tau'\text{)} \\
[\![\forall \alpha\,(\tau)]\!] &= \Pi \alpha : *\,([\![\tau']\!]) \\
[\![x]\!] &= x \\
[\![\lambda x : \tau\,(M)]\!] &= \lambda x : [\![\tau]\!]\,([\![M]\!]) \\
[\![M\,M']\!] &= [\![M]\!]\,[\![M']\!] \\
[\![\Lambda \alpha\,(M)]\!] &= \lambda \alpha : *\,([\![M]\!]) \\
[\![M\,\tau]\!] &= [\![M]\!]\,[\![\tau]\!]
\end{aligned}$$

Properties of the translation from PLC to $\lambda 2$

- If $\{\,\} \vdash M : \tau$ is derivable in PLC, then $\diamond \vdash [\![\tau]\!] : *$ and $\diamond \vdash [\![M]\!] : [\![\tau]\!]$ are derivable in $\lambda 2$
- In $\lambda 2$, if $\diamond \vdash t : \Box$, then $t = *$; if $\diamond \vdash t : *$, then $t = [\![\tau]\!]$ for some closed PLC type τ; and if $\diamond \vdash t : t'$ then $t = [\![M]\!]$ and $t' = [\![\tau]\!]$ for PLC expressions satisfying $\{\,\} \vdash M : \tau$.
- Under the translation, the reduction behaviour of PLC terms is preserved and reflected by beta-reduction in $\lambda 2$. (Note in particular that PLC types are translated to pseudo-terms in beta-normal form.)