# THEORETICAL PEARLS

## *Type-checking Injective Pure Type Systems*

Gilles Barthe

*Departamento de Informática*
*Universidade do Minho*
*Braga, Portugal*
*gilles@di.uminho.pt*

In memory of Yossi Shamir

**Abstract**

Injective Pure Type Systems form a large class of Pure Type Systems for which one can compute by purely syntactic means two sorts $\mathsf{e|mt}(\Gamma|M)$ and $\mathsf{sort}(\Gamma|M)$, where $\Gamma$ is a pseudo-context and $M$ is a pseudo-term, and such that for every sort $s$,

$$\Gamma \vdash M : A \quad \wedge \quad \Gamma \vdash A : s \quad \Rightarrow \quad \mathsf{e|mt}(\Gamma|M) = s$$
$$\Gamma \vdash M : s \quad \Rightarrow \quad \mathsf{sort}(\Gamma|M) = s$$

By eliminating the problematic clause in the (abstraction) rule in favor of constraints over $\mathsf{e|mt}(.|.)$ and $\mathsf{sort}(.|.)$, we provide a sound and complete type-checking algorithm for injective Pure Type Systems.

In addition, we prove Expansion Postponement for a variant of injective Pure Type Systems where the problematic clause in the (abstraction) rule is replaced in favor of constraints over $\mathsf{e|mt}(.|.)$ and $\mathsf{sort}(.|.)$.

### 1 Introduction

Pure Type Systems provide an elegant and general framework for the definition and study of typed $\lambda$-calculi (Barendregt, 1992; Berardi, 1990; Geuvers, 1993; Geuvers & Nederhof, 1991; Terlouw, 1989). One central issue in the theory of Pure Type Systems is the problem of type-checking. Given a Pure Type System $\lambda\mathbf{S}$, type-checking consists in deciding whether a judgment $\Gamma \vdash M : A$ is derivable according to the rules of Pure Type Systems. Although type-checking is undecidable in general (Coquand & Herbelin, 1994; Pollack, 1992), most systems of interest have decidable type-checking (Benthem Jutting, 1993). For such systems, the question remains whether it is possible to find reasonable, sound and complete, algorithms for type-checking. One crucial phase in the design of such algorithms is to find sound and complete syntax-directed presentations of Pure Type Systems (Pollack,

1995); informally, a set of rules is syntax-directed if, using this set of rules, there is at most one way to derive a type for a given expression in a given context—and the type is unique.

Over the last years, several authors have proposed such syntax-directed presentations for some specific classes of Pure Type Systems (Barthe, 1998; Benthem Jutting *et al.*, 1994; Poll, 1993; Pollack, 1994; Severi, 1996; Severi, 1998). However, the situation is in our view unsatisfactory because:

- these presentations either impose strong restrictions on the Pure Type Systems or make use of a complex derivability relation, see Section 6;
- the completeness of the most natural syntax-directed presentation, as formulated in (Pollack, 1992; Pollack, 1994), remains an open problem, see Section 3.

The aim of this paper is to show that a simplified variant of Pollack's natural syntax-directed presentation is sound and complete for injective Pure Type Systems, a class of Pure Type Systems that includes many of the systems occurring in the literature, in particular the systems of Barendregt's $\lambda$-cube (Barendregt, 1991; Barendregt, 1992). The idea is to define for every pseudo-context $\Gamma$ and pseudo-term $M$ two sorts $\mathsf{elmt}(\Gamma|M)$ and $\mathsf{sort}(\Gamma|M)$, which may be computed easily and without invoking conversion or substitution, and such that for every sort $s$,

$$\begin{array}{rcl} \Gamma \vdash M : A \quad \wedge \quad \Gamma \vdash A : s & \Rightarrow & \mathsf{elmt}(\Gamma|M) = s \\ \Gamma \vdash M : s & \Rightarrow & \mathsf{sort}(\Gamma|M) = s \end{array}$$

Then we use $\mathsf{elmt}(.|.)$ and $\mathsf{sort}(.|.)$ to eliminate the problematic clause in the (abstraction) rule of Pure Type Systems and obtain a sound and complete syntax-directed presentation. Besides, we show that the same idea also applies to the problem of Expansion Postponement (Barthe *et al.*, 1998; Poll, 1998; Pollack, 1994).

*Contents* The remaining of the paper is organized as follows: in Section 2, we provide a brief overview of Pure Type Systems. In Section 3, we present two motivating open problems, namely the completeness of Pollack's "natural" syntax-directed presentation and Expansion Postponement. In Section 4, we present a new derivability relation for injective Pure Type Systems and in Section 5, we show that this relation is the key to a neat syntax-directed presentation. Section 6 establishes a comparison with related work. Finally we conclude in Section 7.

*Acknowledgments* I am indebted to R. Pollack for suggesting significant improvements and finding a mistake in an earlier version of the paper. I am grateful to M.H. Sørensen for exchanges on the Classification Lemma, to H. Elbers, E. Poll and J. Zwanenburg for detailed comments on an earlier version of the paper, and to S. Peyton Jones for providing the initial motivation for this work through discussions on (Peyton Jones & Meijer, 1997).

Most of this work was carried while working at the Centrum voor Wiskunde en Informatica (CWI), Amsterdam, the Netherlands and at Chalmers Tekniska Högskola, Göteborg, Sweden. I acknowledge financial support from the Dutch Science Foundation (NWO) and from the European TMR programme.

## 2 Pure Type Systems

In this section, we present the basics of Pure Type Systems. Only crucial properties are considered. Other properties can be found in standard texts on Pure Type Systems (Barendregt, 1992; Geuvers, 1993; Geuvers & Nederhof, 1991).

### *2.1 Specifications*

Pure Type Systems provide a parametric framework for typed $\lambda$-calculi à la Church. Parametricity is achieved through the notion of specification, which consists of a set of universes and two relations expressing abstract dependencies between them.

*Definition 1 (Specification)*
A *specification* is a triple $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ where

- $\mathcal{S}$ is a set of *sorts*, ranged over by $s, s_1, s_2 \ldots$;
- $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ is a set of *axioms*;
- $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ is a set of *rules*; as usual, $(s_1, s_2)$ meant as a rule denotes $(s_1, s_2, s_2)$.

Every specification has a set of *typed sorts*, defined as

$$S^\bullet = \{s \in \mathcal{S} \mid \exists s' \in \mathcal{S}.\ (s, s') \in \mathcal{A}\}$$

In this paper, we will chiefly be concerned with injective specifications, which we introduce below. In Subsection 6.1, we consider further classes of specifications.

*Definition 2 (Injective)*
Let $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ be a specification.

- $\mathbf{S}$ is *functional* if for every $s_1, s_2, s_2', s_3, s_3' \in \mathcal{S}$,

$$\begin{array}{lcll} (s_1, s_2) \in \mathcal{A} & \wedge & (s_1, s_2') \in \mathcal{A} & \Rightarrow \quad s_2 = s_2' \\ (s_1, s_2, s_3) \in \mathcal{R} & \wedge & (s_1, s_2, s_3') \in \mathcal{R} & \Rightarrow \quad s_3 = s_3' \end{array}$$

- $\mathbf{S}$ is *injective* if it is functional and for every $s_1, s_1', s_2, s_2', s_3 \in \mathcal{S}$,

$$\begin{array}{lcll} (s_1, s_2) \in \mathcal{A} & \wedge & (s_1', s_2) \in \mathcal{A} & \Rightarrow \quad s_1 = s_1' \\ (s_1, s_2, s_3) \in \mathcal{R} & \wedge & (s_1, s_2', s_3) \in \mathcal{R} & \Rightarrow \quad s_2 = s_2' \end{array}$$

### *2.2 Pure Type Systems*

Every specification $\mathbf{S}$ yields a Pure Type System $\lambda\mathbf{S}$ as specified below. Throughout this section, $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ is a fixed specification.

*Definition 3 (Pure Type Systems)*

- The set $\mathcal{T}$ of *pseudo-terms* is given by the abstract syntax:

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

  where $V$ is a fixed countably infinite set of variables. We let $A, B, M \ldots$ range over $\mathcal{T}$ and $x, y, z \ldots$ range over $V$.

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

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

$$\text{(weakening)} \quad \frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B} \qquad \text{if } x \notin \mathrm{dom}(\Gamma) \text{ and } A \in V \cup \mathcal{S}$$

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

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

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

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

Fig. 1. Rules for Pure Type Systems

- Syntactic equality is denoted by $=$;
- *$\beta$-reduction* $\rightarrow_\beta$ is defined as the compatible closure of the contraction

$$(\lambda x{:}A.\, M)\ N \rightarrow_\beta M\{x := N\}$$

  where $\bullet\{\bullet := \bullet\}$ is the standard substitution operator. The reflexive transitive closure of $\rightarrow_\beta$ is denoted by $\twoheadrightarrow_\beta$.
- *$\beta$-equality* $=_\beta$ is the reflexive, symmetric, transitive closure of $\rightarrow_\beta$.
- A *pseudo-context* is a finite ordered list $x_1 : A_1,\ \ldots,\ x_n : A_n$ where $x_1, \ldots, x_n$ are pairwise distinct The empty context is denoted by $\langle\rangle$. The domain of a context $\Gamma$ is

$$\mathrm{dom}(\Gamma) = \{x \mid \exists t \in \mathcal{T}.\ x : t \in \Gamma\}$$

  The set of pseudo-contexts is denoted by $\mathcal{G}$. We let $\Gamma, \Gamma', \ldots$ range over $\mathcal{G}$.
- A *judgment* is a triple $\Gamma \vdash M : A$ .
- The *derivability* relation $\vdash$ is given by the rules of Figure 1. If $\Gamma \vdash M : A$ is derivable, then $\Gamma$, $M$ and $A$ are *legal*.

Note that, contrary to usual practice, we insist on variables being declared at most once in a pseudo-context. This is a mere technicality, adopted so as to disambiguate the definition of $\mathsf{sort}(.|.)$ and $\mathsf{elmt}(.|.)$ in Definition 9.

### *2.3 Properties*

Only a few properties of Pure Type Systems will be used crucially in the sequel. The first property ensures that the type of a term is itself typable, unless it is a sort.

*Lemma 4 (Correctness of types)*

$$\Gamma \vdash A : B \quad \Rightarrow \quad B \in \mathcal{S} \quad \vee \quad \exists s \in \mathcal{S}.\ \Gamma \vdash B : s$$

The second property ensures that types are closed under reduction.

*Proposition 5 (Subject Reduction)*

$$\Gamma \vdash M : A \quad \wedge \quad M \to_\beta N \quad \Rightarrow \quad \Gamma \vdash N : A$$

The remaining properties are concerned with specific classes of systems. The third property states that functional Pure Type Systems enjoy Uniqueness of Types.

*Lemma 6 (Uniqueness of Types)*
Assume **S** is functional.

$$\Gamma \vdash M : A \quad \wedge \quad \Gamma \vdash M : B \quad \Rightarrow \quad A =_\beta B$$

A useful consequence of Uniqueness of Types is that convertible types must inhabit the same sort.

*Corollary 7 (Preservation of Sorts)*
Assume **S** is functional.

$$\Gamma \vdash A : s \quad \wedge \quad \Gamma \vdash A' : s' \quad \wedge \quad A =_\beta A' \quad \Rightarrow \quad s = s'$$

A useful variant of the above result is

$$\Gamma \vdash M : A \quad \wedge \quad \Gamma \vdash A' : s \quad \wedge \quad A =_\beta A' \quad \Rightarrow \quad \Gamma \vdash A : s$$

The fourth and last property is the classification algorithm for injective Pure Type Systems. We begin with some preliminary definitions. For every set $A$, we let $A^\uparrow$ denote $A \cup \{\uparrow\}$, where it is assumed that $\uparrow$ is fresh. If $f \in A \to B^\uparrow$ and $a \in A$, we write $f\ a\ \downarrow$ to denote $f\ a \neq\ \uparrow$.

*Definition 8*
Assume **S** is injective.

- The map $.^- : \mathcal{S}^\uparrow \to \mathcal{S}^\uparrow$ is defined by

$$s^- = \begin{cases} s' & \text{if } (s', s) \in \mathcal{A} \\ \uparrow & \text{otherwise} \end{cases}$$

- The map $.^+ : \mathcal{S}^\uparrow \to \mathcal{S}^\uparrow$ is defined by

$$s^+ = \begin{cases} s' & \text{if } (s, s') \in \mathcal{A} \\ \uparrow & \text{otherwise} \end{cases}$$

- The map $\rho : \mathcal{S}^{\uparrow} \times \mathcal{S}^{\uparrow} \to \mathcal{S}^{\uparrow}$ is defined by

$$\rho(s_1, s_2) = \begin{cases} s_3 & \text{if } (s_1, s_2, s_3) \in \mathcal{R} \\ \uparrow & \text{otherwise} \end{cases}$$

- The map $\mu : \mathcal{S}^{\uparrow} \times \mathcal{S}^{\uparrow} \to \mathcal{S}^{\uparrow}$ is defined by

$$\mu(s_1, s_2) = \begin{cases} s_3 & \text{if } (s_1, s_3, s_2) \in \mathcal{R} \\ \uparrow & \text{otherwise} \end{cases}$$

The above definition and the classification algorithm, as defined below, exploit all the properties of injectivity.

*Definition 9 (Classification algorithm)*
The maps $\mathsf{sort}(.|.) : \mathcal{G} \times \mathcal{T}^{\uparrow} \to \mathcal{S}^{\uparrow}$ and $\mathsf{elmt}(.|.) : \mathcal{G} \times \mathcal{T}^{\uparrow} \to \mathcal{S}^{\uparrow}$ are defined as follows:

$$\begin{array}{rcll}
\mathsf{elmt}(\Gamma|\uparrow) & = & \uparrow & \\
\mathsf{sort}(\Gamma|\uparrow) & = & \uparrow & \\
\mathsf{elmt}(\Gamma|x) & = & \mathsf{sort}(\Gamma_0|A) & \text{if } \Gamma = \Gamma_0, x : A, \Gamma_1 \\
\mathsf{sort}(\Gamma|x) & = & \mathsf{elmt}(\Gamma|x)^- & \\
\mathsf{elmt}(\Gamma|s) & = & (\mathsf{sort}(\Gamma|s))^+ & \\
\mathsf{sort}(\Gamma|s) & = & s^+ & \\
\mathsf{elmt}(\Gamma|M\ N) & = & \mu(\mathsf{elmt}(\Gamma|N), \mathsf{elmt}(\Gamma|M)) & \\
\mathsf{sort}(\Gamma|M\ N) & = & (\mathsf{elmt}(\Gamma|M\ N))^- & \\
\mathsf{elmt}(\Gamma|\lambda x{:}A.\ M) & = & \rho(\mathsf{sort}(\Gamma|A), \mathsf{elmt}(\Gamma, x : A|M)) & \\
\mathsf{sort}(\Gamma|\lambda x{:}A.\ M) & = & (\mathsf{elmt}(\Gamma|\lambda x{:}A.\ M))^- & \\
\mathsf{elmt}(\Gamma|\Pi x{:}\ A.\ B) & = & (\mathsf{sort}(\Gamma|\Pi x{:}\ A.\ B))^+ & \\
\mathsf{sort}(\Gamma|\Pi x{:}\ A.\ B) & = & \rho(\mathsf{sort}(\Gamma|A), \mathsf{sort}(\Gamma, x : A|B)) &
\end{array}$$

If $\mathbf{S}$ is injective, then $\mathsf{sort}(\Gamma|M)$ and $\mathsf{elmt}(\Gamma|M)$ are well-defined. Moreover:

*Fact 10*
If $\mathbf{S}$ is injective, $\mathsf{sort}(\Gamma|M) \downarrow$ and $\mathsf{elmt}(\Gamma|M) \downarrow$ then

$$\begin{array}{rcl}
(\mathsf{elmt}(\Gamma|M))^- & = & \mathsf{sort}(\Gamma|M) \\
(\mathsf{sort}(\Gamma|M))^+ & = & \mathsf{elmt}(\Gamma|M)
\end{array}$$

*Proof*
By induction on the structure of $M$. □

The following result justifies the name of classification algorithm and is central to the remaining of the paper.

*Proposition 11 (Classification Lemma)*
Assume $\mathbf{S}$ is injective.

$$\begin{array}{lcl}
\Gamma \vdash M : A \ \wedge \ \Gamma \vdash A : s & \Rightarrow & \mathsf{elmt}(\Gamma|M) = s \\
\Gamma \vdash M : A \ \wedge \ A \in \mathcal{S} & \Rightarrow & \mathsf{sort}(\Gamma|M) = A
\end{array}$$

*Proof*
First prove by induction on the structure of $M$:

$$\begin{array}{lcl}
\mathsf{elmt}(\Gamma|M) \downarrow \ \wedge \ \Gamma \subseteq \Gamma' & \Rightarrow & \mathsf{elmt}(\Gamma'|M) \downarrow \ \wedge \ \mathsf{elmt}(\Gamma|M) = \mathsf{elmt}(\Gamma'|M) \\
\mathsf{sort}(\Gamma|M) \downarrow \ \wedge \ \Gamma \subseteq \Gamma' & \Rightarrow & \mathsf{sort}(\Gamma'|M) \downarrow \ \wedge \ \mathsf{sort}(\Gamma|M) = \mathsf{sort}(\Gamma'|M)
\end{array}$$

Then prove by induction on the derivation of $\Gamma \vdash M : A$ that

$$\begin{array}{lrcl} (1) & \Gamma \vdash A : s & \Rightarrow & \mathsf{elmt}(\Gamma|M) = s \\ (2) & A \in \mathcal{S} & \Rightarrow & \mathsf{sort}(\Gamma|M) = A \end{array}$$

**Axiom** Obvious.

**Start** Assume the last rule is

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

For (1), assume $\Gamma \vdash A : s'$ for some $s' \in \mathcal{S}$. By Uniqueness of Types, $s = s'$. By induction hypothesis, $\mathsf{sort}(\Gamma|A) = s$. By definition,

$$\mathsf{elmt}(\Gamma, x : A|x) = \mathsf{sort}(\Gamma|A) = s$$

For (2), assume $A \in \mathcal{S}$. Then $A = s^-$. Hence

$$\mathsf{sort}(\Gamma, x : A|x) = (\mathsf{elmt}(\Gamma, x : A|x))^- = s^- = A$$

**Weakening** Use the result above.

**Product** Assume the last rule is

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

For (2), observe that $\mathsf{sort}(\Gamma|A) = s_1$ and $\mathsf{sort}(\Gamma, x : A|B) = s_2$ by induction hypothesis. Hence

$$\mathsf{sort}(\Gamma|\Pi x{:}A.\ B) = \rho(\mathsf{sort}(\Gamma|A), \mathsf{sort}(\Gamma, x : A|B)) = \rho(s_1, s_2) = s_3$$

For (1), assume $\Gamma \vdash s_3 : s$ for some $s \in \mathcal{S}$. Necessarily $s = s_3^+$. By (2), $\mathsf{sort}(\Gamma|\Pi x{:}A.\ B) = s_3$. Hence

$$\mathsf{elmt}(\Gamma|\Pi x{:}A.\ B) = (\mathsf{sort}(\Gamma|\Pi x{:}A.\ B))^+ = s_3^+ = s$$

**Application** Assume the last rule is

$$\frac{\Gamma \vdash F : (\Pi x{:}A.\ B) \qquad \Gamma \vdash a : A}{\Gamma \vdash F\ a : B\{x := a\}}$$

For (1), assume $\Gamma \vdash B\{x := a\} : s$. To show $s = \mathsf{elmt}(\Gamma|Fa)$. By Correctness of Types, $\Gamma \vdash (\Pi x{:}A.B) : s'$ for some $s' \in \mathcal{S}$. By generation, $\Gamma \vdash A : s_1$ and $\Gamma, x : A \vdash B : s_2$ for $s_1, s_2 \in \mathcal{S}$ such that $(s_1, s_2, s') \in \mathcal{R}$. By induction hypothesis, $s_1 = \mathsf{elmt}(\Gamma|a)$ and $s' = \mathsf{elmt}(\Gamma|F)$. By substitution, $\Gamma \vdash B\{x := a\} : s_2$ and by Uniqueness of Types $s_2 = s$. By injectivity, $s_2 = \mu(\mathsf{elmt}(\Gamma|a), \mathsf{elmt}(\Gamma|F))$ so we are done.

For (2), assume $B\{x := a\} \in \mathcal{S}$. Necessarily $\Gamma \vdash B\{x := a\} : s'$ for some $s'$ and we can proceed as in (1) to conclude $s' = \mathsf{elmt}(\Gamma|F\ a)$. Moreover $B\{x := a\} = s'^-$. Hence

$$\mathsf{sort}(\Gamma|F\ a) = (\mathsf{elmt}(\Gamma|F\ a))^- = s'^- = B\{x := a\}$$

**Abstraction** Assume the last rule is

$$\frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash (\Pi x{:}A.\ B) : s}{\Gamma \vdash \lambda x{:}A.\ b : \Pi x{:}A.\ B}$$

For (1), assume $\Gamma \vdash \Pi x\!:\!A.\ B : s'$ for some $s' \in \mathcal{S}$. By Uniqueness of Types, $s = s'$. By induction hypothesis, $\mathsf{sort}(\Gamma|\Pi x\!:\!A.\ B) = s$. Moreover, the derivation of $\Gamma \vdash (\Pi x\!:\!A.\ B) : s$ must contain sub-derivations of $\Gamma \vdash A : s_1$ and $\Gamma, x : A \vdash B : s_2$ with $(s_1, s_2, s) \in \mathcal{R}$ hence by induction hypothesis $\mathsf{sort}(\Gamma|A) = s_1$. Also by induction hypothesis, $\mathsf{elmt}(\Gamma, x : A|b) = s_2$. Hence

$$\mathsf{elmt}(\Gamma|\lambda x\!:\!A.\ b) = \rho(\mathsf{sort}(\Gamma|A), \mathsf{elmt}(\Gamma, x : A|b)) = \rho(s_1, s_2) = s$$

For (2), note that $\Pi x\!:\!A.\ B \notin \mathcal{S}$.

**Conversion** Assume the last rule is

$$\frac{\Gamma \vdash M : B \qquad \Gamma \vdash B' : s}{\Gamma \vdash M : B'} \quad \text{if } B =_\beta B'$$

For (1), assume $\Gamma \vdash B' : s'$ for some $s' \in \mathcal{S}$. To show $\mathsf{elmt}(\Gamma|M) = s'$. Necessarily $\Gamma \vdash B : s'$ so we may apply the induction hypothesis to conclude directly.

For (2), assume $B' \in \mathcal{S}$. As in (1), we may conclude $\mathsf{elmt}(\Gamma|M) = s$. Moreover, $B' = s^-$ hence

$$\mathsf{sort}(\Gamma|M) = (\mathsf{elmt}(\Gamma|M))^- = s^- = B'$$

□

*Corollary 12 (Extended Classification Lemma)*
Assume **S** is injective.

1. If $\Gamma \vdash M : B$ and $\mathsf{elmt}(\Gamma|M) \downarrow$ then $\Gamma \vdash B : \mathsf{elmt}(\Gamma|M)$.
2. If $\Gamma, x : A \vdash M : B$ then $\mathsf{sort}(\Gamma|A) \downarrow$ and $\Gamma \vdash A : \mathsf{sort}(\Gamma|A)$.

*Proof*
(1) Prove by case analysis on $M$ that that

$$\Gamma \vdash M : B \quad \wedge \quad \mathsf{elmt}(\Gamma|M) \downarrow \quad \Rightarrow \quad \exists s \in \mathcal{S}.\ \Gamma \vdash B : s$$

Conclude from Lemma 11.
(2) By the Start Lemma,

$$\exists s \in \mathcal{S}.\ \Gamma \vdash A : s$$

Conclude from Lemma 11. □

Note that, unlike the traditional formulation of the Classification Lemma (Geuvers, 1993), our result does not require the use of sorted variables. Of course, the classification algorithm and the syntax-directed presentation of the coming sections remain correct (and can even be simplified) if one uses sorted variables.

## 3 Type-Checking and Expansion Postponement

In order to provide useful background for the remaining of this paper, we review two open problems for Pure Type Systems, namely the completeness of Pollack's "natural" syntax-directed presentation and Expansion Postponement.

(axiom) $\quad \langle\rangle \vdash_{\mathsf{nat}} s_1 : s_2 \quad$ if $(s_1, s_2) \in \mathcal{A}$

(start) $\quad \dfrac{\Gamma \vdash_{\mathsf{nat}} A :\twoheadrightarrow_{wh} s}{\Gamma, x : A \vdash_{\mathsf{nat}} x : A} \quad$ if $x \notin \mathrm{dom}(\Gamma)$

(weakening) $\quad \dfrac{\Gamma \vdash_{\mathsf{nat}} A : B \qquad \Gamma \vdash_{\mathsf{nat}} C :\twoheadrightarrow_{wh} s}{\Gamma, x : C \vdash_{\mathsf{nat}} A : B} \quad$ if $x \notin \mathrm{dom}(\Gamma)$ and $A \in V \cup \mathcal{S}$

(product) $\quad \dfrac{\Gamma \vdash_{\mathsf{nat}} A :\twoheadrightarrow_{wh} s_1 \qquad \Gamma, x : A \vdash_{\mathsf{nat}} B :\twoheadrightarrow_{wh} s_2}{\Gamma \vdash_{\mathsf{nat}} (\Pi x{:}A.\, B) : s_3} \quad$ if $(s_1, s_2, s_3) \in \mathcal{R}$

(application) $\quad \dfrac{\Gamma \vdash_{\mathsf{nat}} F :\twoheadrightarrow_{wh} (\Pi x{:}A'.\, B) \qquad \Gamma \vdash_{\mathsf{nat}} a : A}{\Gamma \vdash_{\mathsf{nat}} F\, a : B\{x := a\}} \quad$ if $A =_\beta A'$

(abstraction) $\quad \dfrac{\Gamma, x : A \vdash_{\mathsf{nat}} b : B \qquad \Gamma \vdash_{\mathsf{nat}} \Pi x{:}A.\, B : s}{\Gamma \vdash_{\mathsf{nat}} \lambda x{:}A.\, b : \Pi x{:}A.\, B}$

Fig. 2. Pollack's syntax-directed rules for Pure Type Systems

### *3.1 A natural type-checking algorithm*

An important aspect of type-checking is syntax-directedness (Benthem Jutting *et al.*, 1994). Informally, a set of rules is syntax-directed if, using this set of rules, there is at most one way to derive a type for a given pseudo-term $M$ in a given pseudo-context $\Gamma$—and the type is unique. The rules for Pure Type Systems are not syntax-directed, in particular because of the conversion rule which may be applied at any moment. In order to achieve syntax-directedness while maintaining essentially the same set of derivable judgments, one may want to distribute the (conversion) rule over the remaining rules of Pure Type Systems. The algorithm below, which is due to R. Pollack (1992; 1994), is the result of performing such a distribution—there is a certain irony in coining the next definition as Pollack's algorithm, since the algorithm is called *worse* in (Pollack, 1994).

*Definition 13 (Pollack's algorithm)*

- *Weak-head reduction* $\rightarrow_{wh}$ is the smallest relation such that for every $x \in V$ and $A, P, Q, \vec{R} \in \mathcal{T}$

$$(\lambda x : A.\ P)\ Q\ \vec{R} \quad \rightarrow_{wh} \quad P\{x := Q\}\ \vec{R}$$

  (Weak-head reduction differs from $\beta$-reduction by applying only at the top-level.)
- The reflexive-transitive closure of $\rightarrow_{wh}$ is denoted by $\twoheadrightarrow_{wh}$.
- We write $\Gamma \vdash_{\mathsf{nat}} M :\twoheadrightarrow_{wh} A$ for

$$\exists A' \in \mathcal{T}.\ \Gamma \vdash_{\mathsf{nat}} M : A' \ \wedge\ A' \twoheadrightarrow_{wh} A$$

- The derivability relation $\Gamma \vdash_{\mathsf{nat}} M : A$ is given by the rules of Figure 2.

A simple induction on the structure of derivations establishes *soundness*, i.e. $\vdash_{\mathsf{nat}} \subseteq \vdash$. The problem of *completeness* is defined as follows.

*Open Problem 14*
$\Gamma \vdash M : A \quad \Rightarrow \quad \exists A' \in \mathcal{T}.\ \Gamma \vdash_{\mathsf{nat}} M : A' \ \wedge \ A =_\beta A'$

As observed in (Pollack, 1992), we have to restrict ourselves to functional Pure Type Systems, otherwise $\vdash_{\mathsf{nat}}$ is not complete with respect to $\vdash$.

The main problem in proving the completeness of $\vdash_{\mathsf{nat}}$ is the second premise in the (abstraction) rule. When trying to prove the above implications by induction on the derivations, the induction step for the (abstraction) rule cannot be completed, precisely because of its second premise; see (Poll, 1993) for a careful analysis of the failure of the induction step.

One solution to this problem is to formulate a new, more liberal, (abstraction) rule and to perform the distribution of the (conversion) rule over this new set of rules. This solution, which is adopted in most works on type-checking Pure Type Systems, see Section 6, is implemented in the next sections where we propose a new (abstraction) rule inspired from the Classification Lemma and show that it yields a sound and complete syntax-directed system for injective Pure Type Systems.

### *3.2 Expansion Postponement*

The problem of Expansion Postponement, due to H. Barendregt, consists in determining whether one may replace the conversion rule of Pure Type Systems by a reduction rule, without 'essentially' affecting the set of derivable judgments.

*Definition 15 (Expansion Postponement)*

- The relation $\Gamma \vdash_{\mathsf{R}} M : A$ is defined by the rules of Figure 1, except for the rule (conversion) which is replaced by

$$\frac{\Gamma \vdash_{\mathsf{R}} A : B \qquad \Gamma \vdash_{\mathsf{R}} B' : s}{\Gamma \vdash_{\mathsf{R}} A : B'} \quad \text{if } B \twoheadrightarrow_\beta B'$$

- The relation $\Gamma \vdash_{\mathsf{r}} M : A$ is defined by the rules of Figure 1, except for the rule (conversion) which is replaced by

$$\frac{\Gamma \vdash_{\mathsf{r}} A : B}{\Gamma \vdash_{\mathsf{r}} A : B'} \quad \text{if } B \twoheadrightarrow_\beta B'$$

The above definition is taken from (Pollack, 1994) where it is remarked that

$$\vdash_{\mathsf{R}} \ \subseteq \ \vdash_{\mathsf{r}} \ \subseteq \ \vdash$$

The problem of Expansion Postponement has two variants:

*Open Problem 16*
Let $\Vdash$ be $\vdash_{\mathsf{R}}$ or $\vdash_{\mathsf{r}}$.

$$\Gamma \vdash M : A \quad \Rightarrow \quad \exists A' \in \mathcal{T}.\ \Gamma \Vdash M : A' \ \wedge \ A \twoheadrightarrow_\beta A'$$

Proving Expansion Postponement by induction on the structure of derivations fails exactly for the same reason as proving the completeness of $\vdash_{\mathsf{nat}}$, i.e. for the second premise in the (abstraction) rule (Poll, 1998; Pollack, 1994). In 1995, E. Poll (1998) solved the $\vdash_{\mathsf{r}}$-variant of the problem for normalizing Pure Type Systems, i.e. for Pure Type Systems such that

$$\Gamma \vdash M : A \quad \Rightarrow \quad M \text{ is weakly } \beta\text{-normalizing}$$

In 1998, G. Barthe, J. Hatcliff, and M.H.B. Sørensen (1998) solved the $\vdash_{\mathsf{R}}$-variant of the problem (and a fortiori the $\vdash_{\mathsf{r}}$-variant) for a large class Pure Type Systems.

In the next section, we define a new set of rules which, in the case of injective specifications, has the same derivable judgments as $\vdash$ and has the Expansion Postponement property for its $\vdash_{\mathsf{r}}$-variant. This supports the observation in (Benthem Jutting *et al.*, 1994) that Expansion Postponement is relative to a set of rules rather than to a derivability relation.

## 4 A new derivability relation for PTSs

### *4.1 Assessment*

The derivability relation for Pure Type Systems contains a certain amount of redundancy. For example, one can derive from the first premise of the (abstraction) rule

$$\Gamma, x : A \vdash M : B$$

that

$$(H_1) \qquad \exists s_1 \in \mathcal{S}.\ \Gamma \vdash A : s_1$$
$$(H_2) \qquad (\exists s_2 \in \mathcal{S}.\ \Gamma, x : A \vdash B : s_2) \ \vee \ B \in \mathcal{S}$$

There is only a small step to infer the second premise of the (abstraction) rule (the existential quantification below is usually left implicit)

$$\exists s \in \mathcal{S}.\ \Gamma \vdash \Pi x{:}A.\ B : s$$

While we do not know how to make this small step in general, the Classification Lemma allows us to make this small step for injective specifications. Indeed, for such specifications, we can rephrase $(H_1)$ and $(H_2)$ as

$$(H_1') \qquad \Gamma \vdash A : \mathsf{sort}(\Gamma|A)$$
$$(H_2') \qquad \Gamma, x : A \vdash B : \mathsf{elmt}(\Gamma, x : A|b) \vee \mathsf{elmt}(\Gamma, x : A|b) \ \uparrow$$

Under these hypotheses, the second premise of the (abstraction) rule becomes equivalent to $\mathsf{elmt}(\Gamma|\lambda x{:}A.\ b) \ \downarrow$. This is the key to the new derivability relation.

As suggested by R. Pollack, one can proceed similarly with the first premise of the (product) rule. Indeed, the premise $\Gamma \vdash A : s_1$ of the (product) rule is redundant and may be replaced by $\mathsf{sort}(\Gamma|A) = s_1$. Equivalently, the premise $\Gamma \vdash A : s_1$ and the constraint $(s_1, s_2, s_3) \in \mathcal{R}$ may be replaced by $s_3 = \rho(\mathsf{sort}(\Gamma|A), s_2)$.

### *4.2 Classification-based rules*

Throughout this subsection, we let $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ be a fixed injective specification.

(axiom) $\langle\rangle \vdash_{\mathsf{cl}} s_1 : s_2$ if $(s_1, s_2) \in \mathcal{A}$

(start) $$\frac{\Gamma \vdash_{\mathsf{cl}} A : s}{\Gamma, x : A \vdash_{\mathsf{cl}} x : A}$$ if $x \notin \mathrm{dom}(\Gamma)$

(weakening) $$\frac{\Gamma \vdash_{\mathsf{cl}} A : B \quad \Gamma \vdash_{\mathsf{cl}} C : s}{\Gamma, x : C \vdash_{\mathsf{cl}} A : B}$$ if $x \notin \mathrm{dom}(\Gamma)$ and $A \in V \cup \mathcal{S}$

(product) $$\frac{\Gamma, x : A \vdash_{\mathsf{cl}} B : s_2}{\Gamma \vdash_{\mathsf{cl}} (\Pi x{:}A.\, B) : s_3}$$ if $s_3 = \rho(\mathsf{sort}(\Gamma|A), s_2)$

(application) $$\frac{\Gamma \vdash_{\mathsf{cl}} F : (\Pi x{:}A.\, B) \quad \Gamma \vdash_{\mathsf{cl}} a : A}{\Gamma \vdash_{\mathsf{cl}} F\, a : B\{x := a\}}$$

(abstraction) $$\frac{\Gamma, x : A \vdash_{\mathsf{cl}} b : B}{\Gamma \vdash_{\mathsf{cl}} \lambda x{:}A.\, b : \Pi x{:}A.\, B}$$ if $\mathsf{elmt}(\Gamma|\lambda x{:}A.\, b) \downarrow$

(conversion) $$\frac{\Gamma \vdash_{\mathsf{cl}} A : B \quad \Gamma \vdash_{\mathsf{cl}} B' : s}{\Gamma \vdash_{\mathsf{cl}} A : B'}$$ if $B =_\beta B'$

Fig. 3. Classification-Based Rules for injective Pure Type Systems

*Definition 17*
The relation $\Gamma \vdash_{\mathsf{cl}} M : A$ is defined by the rules of Figure 3.

The new derivability relation is sound and complete with respect to the original one.

*Proposition 18*
$\Gamma \vdash M : A \quad \Leftrightarrow \quad \Gamma \vdash_{\mathsf{cl}} M : A$

*Proof*
By induction on the structure of derivations. We treat the reverse implication for the (abstraction) and (product) rules.

- (abstraction): assume that the last rule of the derivation is

$$\frac{\Gamma, x : A \vdash_{\mathsf{cl}} b : B}{\Gamma \vdash_{\mathsf{cl}} \lambda x{:}A.\, b : \Pi x{:}A.\, B} \quad \mathsf{elmt}(\Gamma|\lambda x{:}A.\, b) \downarrow$$

  Set $s_1 = \mathsf{sort}(\Gamma|A)$, $s_2 = \mathsf{elmt}(\Gamma, x : A|b)$ and $s_3 = \rho(s_1, s_2)$. By induction hypothesis,

$$\Gamma, x : A \vdash b : B$$

  By Lemma 12(2),

$$\Gamma \vdash A : s_1$$

  By Lemma 12(1),

$$\Gamma, x : A \vdash B : s_2$$

By (product),

$$\Gamma \vdash \Pi x{:}A.\ B : s_3$$

By (abstraction),

$$\Gamma \vdash \lambda x{:}A.\ b : \Pi x{:}A.\ B$$

- (product): assume that the last rule of the derivation is

$$\frac{\Gamma, x : A \vdash_{\mathsf{cl}} B : s_2}{\Gamma \vdash_{\mathsf{cl}} \Pi x{:}A.\ B : s_3} \quad s_3 = \rho(\mathsf{sort}(\Gamma|A), s_2)$$

Set $s_1 = \mathsf{sort}(\Gamma|A)$. By induction hypothesis,

$$\Gamma, x : A \vdash B : s_2$$

By Lemma 12(2),

$$\Gamma \vdash A : s_1$$

By product

$$\Gamma \vdash \Pi x{:}A.\ B : s_3$$

□

While the relation $\vdash_{\mathsf{cl}}$ is not well-defined for non-injective specifications, one may wonder whether it is possible to modify the definitions of $\mathsf{elmt}(.|.)$ and $\mathsf{sort}(.|.)$ so as to define a classification-based derivability relation that is sound and complete with respect to $\vdash$ for a larger class of Pure Type Systems. In the appendix, we show that one can indeed do so for the class of M-injective specifications.

## 5 Classification, Type-Checking and Expansion Postponement

The classification-based rules enjoy Expansion Postponement and are the key to a simple type-checking algorithm for injective PTSs. Throughout this section, we let $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ be a fixed injective specification.

### *5.1 Type-checking*

The rules for $\vdash_{\mathsf{nat}}$ are obtained by distributing the (conversion) rule over the other rules for $\vdash$. In this subsection, we define a new derivability relation $\vdash_{\mathsf{sd}}$ by distributing the (conversion) rule over the other rules for $\vdash_{\mathsf{cl}}$. Moreover, we show that $\vdash_{\mathsf{sd}}$ is sound and complete with respect to $\vdash_{\mathsf{cl}}$ and hence with respect to $\vdash$.

*Definition 19*
We write $\Gamma \vdash_{\mathsf{sd}} M :\twoheadrightarrow_{wh} A$ for

$$\exists A' \in \mathcal{T}.\ \Gamma \vdash_{\mathsf{sd}} M : A' \ \wedge\ A' \twoheadrightarrow_{wh} A$$

The derivability relation $\Gamma \vdash_{\mathsf{sd}} M : A$ is given by the rules of Figure 4.

We claim that, for injective Pure Type Systems, our algorithm is as natural as $\vdash_{\mathsf{nat}}$; moreover, the side conditions for $\vdash_{\mathsf{sd}}$ are simpler than those for $\vdash_{\mathsf{nat}}$.

| | | |
|---|---|---|
| (axiom) | $\langle\rangle \vdash_{\mathsf{sd}} s_1 : s_2$ | if $(s_1, s_2) \in \mathcal{A}$ |
| (start) | $\dfrac{\Gamma \vdash_{\mathsf{sd}} A :\twoheadrightarrow_{wh} s}{\Gamma, x : A \vdash_{\mathsf{sd}} x : A}$ | if $x \notin \mathrm{dom}(\Gamma)$ |
| (weakening) | $\dfrac{\Gamma \vdash_{\mathsf{sd}} A : B \quad \Gamma \vdash_{\mathsf{sd}} C :\twoheadrightarrow_{wh} s}{\Gamma, x : C \vdash_{\mathsf{sd}} A : B}$ | if $x \notin \mathrm{dom}(\Gamma)$ and $A \in V \cup \mathcal{S}$ |
| (product) | $\dfrac{\Gamma, x : A \vdash_{\mathsf{sd}} B :\twoheadrightarrow_{wh} s_2}{\Gamma \vdash_{\mathsf{sd}} (\Pi x{:}A.\, B) : s_3}$ | if $s_3 = \rho(\mathsf{sort}(\Gamma \mid A), s_2)$ |
| (application) | $\dfrac{\Gamma \vdash_{\mathsf{sd}} F :\twoheadrightarrow_{wh} (\Pi x{:}A'.\, B) \quad \Gamma \vdash_{\mathsf{sd}} a : A}{\Gamma \vdash_{\mathsf{sd}} F\ a : B\{x := a\}}$ | if $A =_\beta A'$ |
| (abstraction) | $\dfrac{\Gamma, x : A \vdash_{\mathsf{sd}} b : B}{\Gamma \vdash_{\mathsf{sd}} \lambda x{:}A.\, b : \Pi x{:}A.\, B}$ | if $\mathsf{elmt}(\Gamma \mid \lambda x{:}A.\, b) \downarrow$ |

Fig. 4. Syntax-directed rules for Pure Type Systems

*Lemma 20*
$\Gamma \vdash_{\mathsf{sd}} M : A \quad \Rightarrow \quad \Gamma \vdash_{\mathsf{cl}} M : A$

*Proof*
By induction on the structure of derivations. □

The completeness of $\vdash_{\mathsf{sd}}$ is stated as follows:

*Proposition 21*
$\Gamma \vdash_{\mathsf{cl}} M : A \quad \Rightarrow \quad \exists A' \in \mathcal{T}.\ \Gamma \vdash_{\mathsf{sd}} M : A' \ \wedge\ A =_\beta A'$

*Proof*
By induction on the structure of derivations. We treat the (application) and (abstraction) rules.

- (application): assume that the last rule of the derivation is

$$\frac{\Gamma \vdash_{\mathsf{cl}} F : (\Pi x{:}A.\, B) \quad \Gamma \vdash_{\mathsf{cl}} a : A}{\Gamma \vdash_{\mathsf{cl}} F\ a : B\{x := a\}}$$

  By induction hypothesis, there exists $C, D$ such that

$$C =_\beta \Pi x{:}A.\, B$$
$$D =_\beta A$$
$$\Gamma \vdash_{\mathsf{sd}} F : C$$
$$\Gamma \vdash_{\mathsf{sd}} a : D$$

  Necessarily there exists $E, G$ such that

$$C \twoheadrightarrow_{wh} \Pi x{:}E.\, G$$
$$E =_\beta A$$
$$G =_\beta B$$

Hence we have

$$\Gamma \vdash_{\mathsf{sd}} F :\twoheadrightarrow_{wh} \Pi x{:}\, E.\, G$$
$$\Gamma \vdash_{\mathsf{sd}} a : D$$
$$E =_\beta D$$

By (application), we get

$$\Gamma \vdash_{\mathsf{sd}} F\ a : G\{x := a\}$$

Moreover, $G\{x := a\} =_\beta B\{x := a\}$ so we are done.

- (abstraction): assume that the last rule of the derivation is

$$\frac{\Gamma, x : A \vdash_{\mathsf{cl}} b : B}{\Gamma \vdash_{\mathsf{cl}} \lambda x{:}A.\, b : \Pi x{:}\, A.\, B} \quad \mathsf{elmt}(\Gamma | \lambda x{:}A.\, b) \downarrow$$

By induction hypothesis, there exists $B'$ such that $B =_\beta B'$ and

$$\Gamma, x : A \vdash_{\mathsf{sd}} b : B'$$

By (abstraction),

$$\Gamma \vdash_{\mathsf{sd}} \lambda x{:}A.\, b : \Pi x{:}\, A.\, B'$$

We have $\Pi x{:}\, A.\, B =_\beta \Pi x{:}\, A.\, B'$, so we are done.

□

### *5.2 Expansion Postponement*

We prove that the weak variant of Expansion Postponement holds for $\vdash_{\mathsf{cl}}$.

*Definition 22*
The relation $\Gamma \vdash_{\mathsf{ep}} M : A$ is defined by the rules of Figure 3, except for the (conversion) rule which is replaced by

$$\frac{\Gamma \vdash_{\mathsf{ep}} A : B}{\Gamma \vdash_{\mathsf{ep}} A : B'} \quad \text{if } B \twoheadrightarrow_\beta B'$$

$\vdash_{\mathsf{ep}}$ is sound with respect to $\vdash_{\mathsf{cl}}$ and hence with respect to $\vdash$.

*Lemma 23*
$\Gamma \vdash_{\mathsf{ep}} M : A \quad \Rightarrow \quad \Gamma \vdash_{\mathsf{cl}} M : A$

*Proof*
By induction on the structure of derivations. □

The Expansion Postponement property is stated as follows:

*Proposition 24*
$\Gamma \vdash_{\mathsf{cl}} M : A \quad \Rightarrow \quad \exists A' \in \mathcal{T}.\ \Gamma \vdash_{\mathsf{ep}} M : A' \ \wedge\ A \twoheadrightarrow_\beta A'$

*Proof*
By induction on the structure of derivations. We treat the (abstraction) and (conversion) rules.

- (abstraction): assume that the last rule of the derivation is

$$\frac{\Gamma, x : A \ \vdash_{\mathsf{cl}} \ b : B}{\Gamma \ \vdash_{\mathsf{cl}} \ \lambda x{:}A.\ b : \Pi x{:}\, A.\ B} \quad \mathsf{e|mt}(\Gamma|\lambda x{:}A.\ b) \ \downarrow$$

  By induction hypothesis, there exists $B'$ such that $B \twoheadrightarrow_\beta B'$ and

$$\Gamma, x : A \ \vdash_{\mathsf{ep}} \ b : B'$$

  By (abstraction),

$$\Gamma \ \vdash_{\mathsf{ep}} \ \lambda x{:}A.\ b : \Pi x{:}\, A.\ B'$$

  We have $\Pi x{:}\, A.\ B \twoheadrightarrow_\beta \Pi x{:}\, A.\ B'$, so we are done.

- (conversion): assume that the last rule of the derivation is

$$\frac{\Gamma \ \vdash_{\mathsf{cl}} \ A : B \qquad \Gamma \ \vdash_{\mathsf{cl}} \ B' : s}{\Gamma \ \vdash_{\mathsf{cl}} \ A : B'} \quad B =_\beta B'$$

  By induction hypothesis, there exists $B''$ such that $B \twoheadrightarrow_\beta B''$ and

$$\Gamma \ \vdash_{\mathsf{ep}} \ A : B''$$

  By Church-Rosser, there exists $C$ such that $B', B'' \twoheadrightarrow_\beta C$. By (conversion),

$$\Gamma \ \vdash_{\mathsf{ep}} \ A : C$$

  so we are done.

□

## 6 Other type-checking algorithms

Before proceeding with related work, we briefly review some background material on Pure Type Systems.

### *6.1 Classes of specifications*

In this subsection, we introduce some of the most important specifications as well as specific classes of specifications that have been considered in the context of type-checking.

We begin by introducing Barendregt's $\lambda$-cube (Barendregt, 1991; Barendregt, 1992), which gives a fine-grain analysis of the Calculus of Constructions and provide the archetypical examples of specifications.

*Definition 25 (Cube-specifications)*

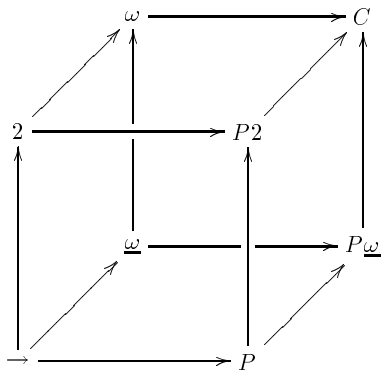


Fig. 5. THE CUBE-SPECIFICATIONS

Let $\mathcal{S} = \{*, \Box\}$ and $\mathcal{A} = \{(*, \Box)\}$. The *cube-specifications* are

$$\begin{array}{lcl}
\rightarrow & = & (\mathcal{S}, \mathcal{A}, \{(*,*)\}) \\
2 & = & (\mathcal{S}, \mathcal{A}, \{(*,*), (\Box,*)\}) \\
\underline{\omega} & = & (\mathcal{S}, \mathcal{A}, \{(*,*), (\Box,\Box)\}) \\
P & = & (\mathcal{S}, \mathcal{A}, \{(*,*), (*,\Box)\}) \\
\omega & = & (\mathcal{S}, \mathcal{A}, \{(*,*), (\Box,*), (\Box,\Box)\}) \\
P2 & = & (\mathcal{S}, \mathcal{A}, \{(*,*), (\Box,*), (*,\Box)\}) \\
P\underline{\omega} & = & (\mathcal{S}, \mathcal{A}, \{(*,*), (\Box,\Box), (*,\Box)\}) \\
C & = & (\mathcal{S}, \mathcal{A}, \{(*,*), (\Box,*), (\Box,\Box), (*,\Box)\})
\end{array}$$

The specifications may be organized into a cube as shown in Figure 5. Most specifications correspond to well-known systems as shown in the following table:

| | |
|---|---|
| $\rightarrow$ | simply typed $\lambda$-calculus |
| 2 | System $F$ (second-order $\lambda$-calculus) |
| $P$ | LF, Automath |
| $\omega$ | System $F^\omega$ (higher-order $\lambda$-calculus) |
| $C$ | Calculus of Constructions |

Another important example of specification is $*$.

| | |
|---|---|
| **Sorts** | * |
| **Axioms** | *: * |
| **Rules** | $(*,*)$ |

THE SPECIFICATION $*$

All the above mentioned specifications have rules of the form $(s_1, s_2, s_2)$. This motivates the following definition.

*Definition 26 (Persistent)*

A specification $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ is *persistent* if for every $s_1, s_1', s_2, s_2', s_3 \in \mathcal{S}$,

$$\begin{array}{rcrcl} & & (s_1, s_2, s_3) \in \mathcal{R} & \Rightarrow & s_2 = s_3 \\ (s_1, s_2) \in \mathcal{A} & \wedge & (s_1, s_2') \in \mathcal{A} & \Rightarrow & s_2 = s_2' \\ (s_1, s_2) \in \mathcal{A} & \wedge & (s_1', s_2) \in \mathcal{A} & \Rightarrow & s_1 = s_1' \end{array}$$

Persistent specifications form an important class of injective specifications for which it is possible to simplify the classification algorithm.

Other well-known classes characterize specifications that have "enough" rules.

*Definition 27 (Full, semi-full)*
Let $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ be a specification.

- $\mathbf{S}$ is *full* if for every $s_1, s_2 \in \mathcal{S}$ there exists $s_3 \in \mathcal{S}$ s.t. $(s_1, s_2, s_3) \in \mathcal{R}$.
- $\mathbf{S}$ is *semi-full* if for every $s_1, s_2, s_3, s_2' \in \mathcal{S}$,

$$(s_1, s_2, s_3) \in \mathcal{R} \quad \Rightarrow \quad \exists s_3' \in \mathcal{S}.\ (s_1, s_2', s_3') \in \mathcal{R}$$

Note that every full specification is semi-full and that not all specifications of interest are full or semi-full. In the case of the cube-specifications, only $P$ and $P\omega$ are semi-full. Beyond Barendregt's $\lambda$-cube, the Calculus of Constructions with Universes $\lambda CU$ is full.

| | | |
|---|---|---|
| **Sorts** | $*^i$ | $(i \in \mathbf{N})$ |
| **Axioms** | $*^i : *^{i+1}$ | $(i \in \mathbf{N})$ |
| **Rules** | $(*^i, *^j, *^k)$ | $(i, j, k \in \mathbf{N} \ \wedge \ (i, j \leq k \vee j = k = 0))$ |

THE SPECIFICATION $CU$

The next definition, due to E. Poll (1993), provides a mild weakening of the notion of injectivity.

*Definition 28 (Weakly injective)*
Let $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ be a specification. $\mathbf{S}$ is *weakly injective* if it is functional and for every $s_1, s_2, s, s' \in \mathcal{S}$,

$$(s_1, s, s_2) \in \mathcal{R} \quad \wedge \quad (s_1, s', s_2) \in \mathcal{R} \quad \Rightarrow \quad s = s'$$

Note that every injective specification is weakly injective.

The relationship between the various classes is depicted in Figure 6 (the notion of M-injective specification and the specifications $C'$, $X$ and $Q$ are defined in the appendix). All inclusions are strict. However, there may not always be a specification of independent interest that belongs to a class $K$ but not to a subclass $K'$; for example, we do not know of any standard semi-full specification that is not M-injective.

*Digression on terminology* The nomenclature for the classes of specifications considered in this paper is rather unfortunate. The notion of injective specification, as presented in this paper, occurs in (Geuvers & Nederhof, 1991). However, Geuvers (1993) uses the word injective for a stronger notion. We prefer to use the terminology strongly injective for this stronger notion—which we do not use in this

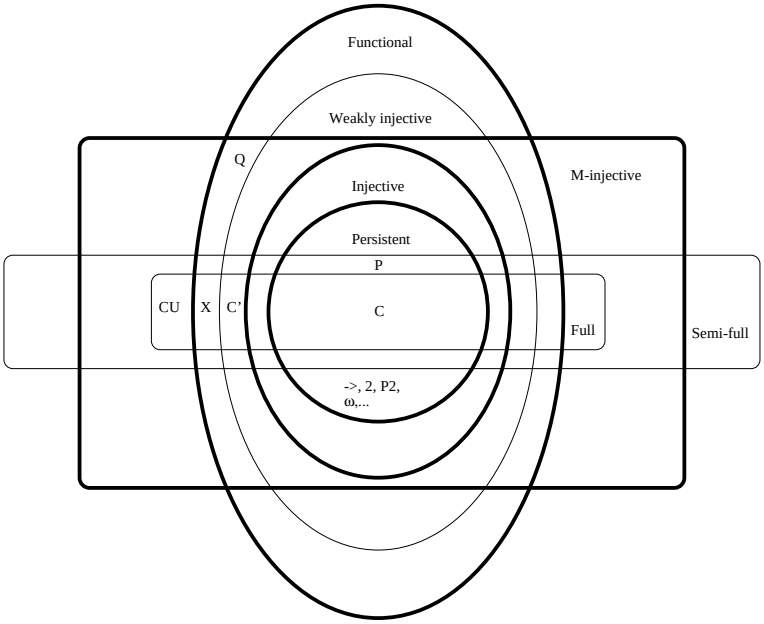


Fig. 6. Classes of Pure Type Systems

paper. Finally, E. Poll (1993) uses the word bijective instead of weakly injective. This is counter-intuitive since every injective specification is bijective but there are bijective specifications that are not injective.

### *6.2 Related work*

Several sound and complete (for certain classes of specifications) syntax-directed presentations of Pure Type Systems may be found in the literature. Without any claim to being exhaustive, we mention:

- full Pure Type Systems: R. Pollack (1992) gives a sound and complete syntax-directed presentation for full Pure Type Systems. The key observation is that, for full Pure Type Systems, the problematic side-condition in the (abstraction) rule can be eliminated. Indeed, for such systems, one can use the (abstraction) rule
$$\frac{\Gamma, x : A \vdash b : B \qquad B \in \mathcal{S} \Rightarrow B \in \mathcal{S}^{\bullet}}{\Gamma \vdash \lambda x{:}A.\, b : \Pi x{:}A.\, B}$$
As in this paper, one may obtain a sound a complete algorithm by replacing the (abstraction) rule of Pure Type Systems with this new rule and by distributing the (conversion) rule over the other rules.
- semi-full Pure Type Systems: as observed by R. Pollack (1992), the above (abstraction) rule may be modified slightly so as to be sound and complete for semi-full Pure Type Systems. Indeed, for such systems it is enough to know what the sort $s_1$ of $A$ is in the (abstraction) rule and whether $(s_1, s_2, s_3) \in \mathcal{R}$

for some $s_2, s_3 \in \mathcal{S}$. This suggests the (abstraction) rule

$$\frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash A : s_1 \quad B \in \mathcal{S} \Rightarrow B \in \mathcal{S}^\bullet}{\Gamma \vdash \lambda x{:}A.\, b : \Pi x{:}A.\, B} \quad (s_1, s_2, s_3) \in \mathcal{R}$$

which indeed yields a sound and complete syntax-directed presentation for semi-full Pure Type Systems.

- functional Pure Type Systems: van Benthem Jutting *et al.* (1994), Pollack (1994), Severi (1996; 1998) and the author (Barthe, 1998) give sound and complete syntax-directed presentations for functional Pure Type Systems. All presentations make use of a more liberal derivability relation for the second premise of the (abstraction) rule. Formally, their (abstraction) rule is of the form

$$\frac{\Gamma, x : A \vdash_{\mathsf{sd}} b : B \quad \Gamma \vdash_{\mathsf{fun}} \Pi x{:}A.\, B : s}{\Gamma \vdash_{\mathsf{sd}} \lambda x{:}A.\, b : \Pi x{:}A.\, B}$$

  where $\vdash_{\mathsf{fun}}$ is an auxiliary derivability relation.
  We see three mild disadvantages to this approach:
  - — conceptual clarity: our side-conditions arise from an elementary analysis of the derivability relation for Pure Type Systems and from a well-established result, i.e. the Classification Lemma, whereas the other derivability relations considered in the above mentioned works appear to be somewhat more intricate;
  - — implementation: the functions $\mathsf{elmt}(.|.)$ and $\mathsf{sort}(.|.)$ are easier to implement than $\vdash_{\mathsf{fun}}$ since they do not involve conversion or substitution;
  - — feasibility: since type-checking for $\vdash$ relies on type-checking for $\vdash_{\mathsf{fun}}$, one needs to prove, in addition to the soundness and completeness results, that weak normalization for $\vdash$ implies weak normalization for $\vdash_{\mathsf{fun}}$.
- weakly injective Pure Type Systems: E. Poll (1993) gives a sound and complete syntax-directed presentation for weakly injective Pure Type Systems. The presentation is concerned with judgments of the form $\Gamma \vdash M : A : s$. Intuitively, such judgments combine in a single relation the usual derivability relation $\Gamma \vdash M : A$ and the function $\mathsf{elmt}(.|.)$. As a result, Poll's presentation is sound and complete for weakly injective specifications rather than for injective specifications. On the other hand, Poll's presentation does not attempt to eliminate redundant information. We conjecture that the second premise of the (abstraction) and (product) rules may be eliminated without affecting soundness and completeness.

## 7 Conclusion

We have presented a sound and complete type-checking algorithm for injective Pure Type Systems and indicated how to adapt the algorithm to $\mathsf{M}$-injective Pure Type Systems. To our knowledge the algorithm, which is a simplification of Pollack's natural type-checking algorithm, is the simplest algorithm for type-checking injective Pure Type Systems. Although almost all standard Pure Type Systems are injective, it is of theoretical interest to determine whether our approach may be generalized

to larger classes of Pure Type Systems, e.g. by relying on the results of (Benthem Jutting, 1993). It may also prove worthwile to extend our results to Cumulative Type Systems, i.e. Pure Type Systems with universe inclusion, see e.g. (Pollack, 1994).

## References


Barendregt, H. (1991). Introduction to Generalised Type Systems. *Journal of Functional Programming*, 1(2):125–154.

Barendregt, H. (1992). Lambda calculi with types. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, pages 117–309. Oxford Science Publications, Volume 2.

Barthe, G. (1998). The semi-full closure of Pure Type Systems. *Pages 316–325 of:* Brim, L., Gruska, J., and Zlatuska, J. (eds), *Proceedings of MFCS'98*. Lecture Notes in Computer Science, volume 1450. Springer-Verlag.

Barthe, G., Hatcliff, J., and Sørensen, M.H.B. (1998). An induction principle for Pure Type Systems. Manuscript.

Benthem Jutting, L.S. van (1993). Typing in pure type systems. *Information and Computation*, 105(1):30–41.

Benthem Jutting, L.S. van, McKinna, J. and Pollack, R. (1994). Checking algorithms for pure type systems. *Pages 19–61 of:* Barendregt, H. and Nipkow, T. (eds), *Proceedings of TYPES'93*. Lecture Notes in Computer Science, volume 806. Springer-Verlag.

Berardi, S. (1990). *Type dependence and Constructive Mathematics*. PhD thesis, University of Torino.

Coquand, T., & Herbelin, H. (1994). A-translation and looping combinators in pure type systems. *Journal of functional programming*, **4**(1), 77–88.

Geuvers, J.H. (1993). *Logics and type systems*. Ph.D. thesis, University of Nijmegen.

Geuvers, J.H. & Nederhof, M.J. (1991). A Modular Proof of Strong Normalization for the Calculus of Constructions. *Journal of Functional Programming*, **1**(2), 155–189.

Harper, R. and Mitchell, J.C. (1993). On the type structure of Standard ML. *ACM Transactions on Programming Languages and Systems*, 15(2):211–252.

Peyton Jones, S. and Meijer, E. (1997). Henk: a typed intermediate language. Proceedings of the ACM Workshop on Types in Compilation, 1997.

Poll, E. (1993). A typechecker for bijective pure type systems. Technical Report CSN93/22, Technical University of Eindhoven, June 1993.

Poll, E. (1998). Theoretical pearl: Expansion postponement for normalizing pure type systems. *Journal of Functional Programming*, 8(10):89–96.

Pollack, R. (1992). Typechecking in pure type systems. In B. Nordström, editor, *Informal proceedings of Logical Frameworks'92*, pages 271–288.

Pollack., R. (1994). *The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions*. PhD thesis, University of Edinburgh.

Pollack., R. (1995). A verified type-checker. *Pages 365–380 of:* Dezani-Ciancaglini, M., & Plotkin, G. (eds), *Proceedings of TLCA'95*. Lecture Notes in Computer Science, volume 902. Springer-Verlag.

Severi, P. (1996). *Normalisation in lambda calculus and its relation to type inference*. PhD thesis, Technical University of Eindhoven.

Severi, P. (1998). Type inference for pure type systems. *Information and Computation*, 143(1):1–23.

Terlouw, J. (1989). Een nadere bewijstheoretische analyse van GSTT's. Manuscript (in Dutch).

## Appendix. Beyond injectivity

We show that one can define a classification-based derivability relation that is sound and complete with respect to $\vdash$ for the class of M-injective specifications, which we introduce below.

*Definition 29*
Let $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ be a specification.

- Let $\mathbf{S}' = (\mathcal{S}', \mathcal{A}', \mathcal{R}')$ be a specification. $H : \mathcal{S} \to \mathcal{S}'$ is a *morphism of specifications*, notation $H : \mathbf{S} \to \mathbf{S}'$, if for every $s_1, s_2, s_3 \in \mathcal{S}$,

$$\begin{array}{rcl} (s_1, s_2) \in \mathcal{A} & \Rightarrow & (H\ s_1, H\ s_2) \in \mathcal{A}' \\ (s_1, s_2, s_3) \in \mathcal{R} & \Rightarrow & (H\ s_1, H\ s_2, H\ s_3) \in \mathcal{R}' \end{array}$$

- A morphism $H : \mathbf{S} \to \mathbf{S}'$ is *faithful* if for every $s_1, s_2 \in \mathcal{S}$,

$$\exists s_3 \in \mathcal{S}.\ (s_1, s_2, s_3) \in \mathcal{R} \ \Leftrightarrow\ \exists s_3' \in \mathcal{S}.\ (H\ s_1, H\ s_2, s_3') \in \mathcal{R}'$$

- $\mathbf{S}$ is *injective modulo* or M*-injective* if there exists a faithful morphism of specifications $H : \mathbf{S} \to \mathbf{S}'$ with $\mathbf{S}'$ an injective specification.

There are several examples of specifications that are not injective but are M-injective, including:

- the predicative variant of $F^\omega$, defined in (Peyton Jones & Meijer, 1997), which we call $Q$;
- the inclusion-free variant of $XML$ (Harper & Mitchell, 1993), which we call $X$;
- the specification $C'$ (Barendregt, 1992), which is full and weakly injective but not injective.

| | |
|---|---|
| **Sorts** | Simp Poly Kind |
| **Axioms** | Simp: Kind |
| **Rules** | (Simp, Simp) (Poly, Poly) (Kind, Kind) |
| | (Kind, Simp, Poly) (Kind, Poly, Poly) |
| | (Simp, Poly, Poly) (Poly, Simp, Poly) |

THE SPECIFICATION $Q$

| | |
|---|---|
| **Sorts** | $\star$ $\Box$ |
| **Axioms** | $\star : \Box$ |
| **Rules** | $(\star, \star)$ $(\Box, \Box)$ $(\Box, \star, \Box)$ $(\star, \Box)$ |

THE SPECIFICATION $X$

| | | |
|---|---|---|
| **Sorts** | $\star_0$ $\star_1$ $\Box$ | |
| **Axioms** | $\star_i : \Box$ | $(i \in \{0,1\})$ |
| **Rules** | $(\star_i, \star_j)$ $(\Box, \Box)$ $(\Box, \star_i)$ $(\star_i, \Box)$ | $(i, j \in \{0,1\})$ |

THE SPECIFICATION $C'$

*Fact 30*

1. Every injective specification is M-injective.
2. Every full specification is M-injective.
3. $Q$, $C'$ and $X$ are M-injective.

*Proof*
(1) An injective specification $\mathbf{S}$ is injective modulo the identity morphism on $\mathbf{S}$. (2) Every full specification is M-injective modulo the unique morphism $!_{\mathbf{S}} : \mathbf{S} \to *$ since $*$ is injective and $!_{\mathbf{S}}$ is faithful. (3) $C'$ and $X$ are full, hence M-injective. $Q$ is injective modulo $H : Q \to \omega$ given by

$$\begin{aligned} H\ \mathsf{Simp} = H\ \mathsf{Poly} &= * \\ H\ \mathsf{Kind} &= \Box \end{aligned}$$

□

In the remaining of this subsection, we show that a suitable adaptation of the classification-based rules for M-injective specifications is sound and complete with respect to the rules of Pure Type Systems. Throughout the remaining of this subsection, assume $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ is injective modulo $H : \mathbf{S} \to \mathbf{S}'$ with $\mathbf{S}' = (\mathcal{S}', \mathcal{A}', \mathcal{R}')$.

*Definition 31*
The relation $\Gamma \vdash_{\mathsf{clm}} M : A$ is defined by the rules of Figure 7.

Note that $\vdash_{\mathsf{clm}}$ does not use Pollack's optimization in the (product) rule, which would be unsound.

The new derivability relation is sound and complete with respect to the original one.

*Proposition 32*
$\Gamma \vdash M : A \quad \Leftrightarrow \quad \Gamma \vdash_{\mathsf{clm}} M : A$

*Proof*
By induction on the structure of derivations. We treat the reverse implication for the (abstraction) rule only. So assume that the last rule of the derivation is

$$\frac{\Gamma, x : A \vdash_{\mathsf{clm}} b : B}{\Gamma \vdash_{\mathsf{clm}} \lambda x{:}A.\ b : \Pi x{:}A.\ B} \quad (B \in \mathcal{S} \Rightarrow B \in \mathcal{S}^{\bullet}) \text{ and } \mathsf{elmt}(H\ \Gamma | H\ (\lambda x{:}A.\ b)) \downarrow$$

By induction hypothesis,

$$\Gamma, x : A \vdash b : B$$

By Correctness of Types, there exists $s_1 \in \mathcal{S}$ such that

$$\Gamma \vdash A : s_1$$

Typing is preserved along morphisms of specifications and hence by Lemma 12(2),

$$H\ s_1 = \mathsf{sort}(H\ \Gamma | H\ A)$$

By Correctness of Types and the assumption $B \in \mathcal{S} \Rightarrow B \in \mathcal{S}^{\bullet}$, we conclude

$$\Gamma, x : A \vdash B : s_2$$

| | | |
|---|---|---|
| (axiom) | $\langle\rangle \vdash_{\mathsf{clm}} s_1 : s_2$ | if $(s_1, s_2) \in \mathcal{A}$ |
| (start) | $\dfrac{\Gamma \vdash_{\mathsf{clm}} A : s}{\Gamma, x : A \vdash_{\mathsf{clm}} x : A}$ | if $x \notin \mathrm{dom}(\Gamma)$ |
| (weakening) | $\dfrac{\Gamma \vdash_{\mathsf{clm}} A : B \quad \Gamma \vdash_{\mathsf{clm}} C : s}{\Gamma, x : C \vdash_{\mathsf{clm}} A : B}$ | if $x \notin \mathrm{dom}(\Gamma)$ and $A \in V \cup \mathcal{S}$ |
| (product) | $\dfrac{\Gamma \vdash_{\mathsf{clm}} A : s_1 \quad \Gamma, x : A \vdash_{\mathsf{clm}} B : s_2}{\Gamma \vdash_{\mathsf{clm}} (\Pi x{:}A.\, B) : s_3}$ | if $(s_1, s_2, s_3) \in \mathcal{R}$ |
| (application) | $\dfrac{\Gamma \vdash_{\mathsf{clm}} F : (\Pi x{:}A.\, B) \quad \Gamma \vdash_{\mathsf{clm}} a : A}{\Gamma \vdash_{\mathsf{clm}} F\, a : B\{x := a\}}$ | |
| (abstraction) | $\dfrac{\Gamma, x : A \vdash_{\mathsf{clm}} b : B}{\Gamma \vdash_{\mathsf{clm}} \lambda x{:}A.\, b : \Pi x{:}A.\, B}$ | if $B \in \mathcal{S} \Rightarrow B \in \mathcal{S}^\bullet$ and $\mathsf{elmt}(H\ \Gamma | H\ (\lambda x{:}A.\, b))\ \downarrow$ |
| (conversion) | $\dfrac{\Gamma \vdash_{\mathsf{clm}} A : B \quad \Gamma \vdash_{\mathsf{clm}} B' : s}{\Gamma \vdash_{\mathsf{clm}} A : B'}$ | if $B =_\beta B'$ |

Fig. 7. Classification-based rules for M-injective Pure Type Systems

Typing is preserved along morphisms of specifications and hence by Lemma 12(1),

$$H\ s_2 = \mathsf{elmt}(H\ \Gamma, x : H\ A | H\ b)$$

Now $\mathsf{elmt}(H\ \Gamma | H\ (\lambda x{:}A.b))\ \downarrow$ hence there exists $s_3' \in \mathcal{S}'$ such that $(H\ s_1, H\ s_2, s_3') \in \mathcal{R}'$. By faithfulness there exists $s_3 \in \mathcal{S}$ such that $(s_1, s_2, s_3) \in \mathcal{R}$. By (product),

$$\Gamma \vdash \Pi x{:}A.\, B : s_3$$

By (abstraction),

$$\Gamma \vdash \lambda x{:}A.\, b : \Pi x{:}A.\, B$$

□

The results of Proposition 32 suggest that Propositions 21 and 24 scale up to M-injective PTSs for suitably modified relations $\vdash_{\mathsf{sd}}$ and $\vdash_{\mathsf{ep}}$.