The Structural Theory of Pure Type Systems

Cody Roux and Floris van Doorn

Carnegie Mellon University
Pittsburgh, PA 15213

Abstract. We investigate possible extensions of arbitrary given Pure Type Systems with additional sorts and rules which preserve the normalization property. In particular we identify the following interesting extensions: the disjoint union $\mathcal{P} + \mathcal{Q}$ of two PTSs $\mathcal{P}$ and $\mathcal{Q}$, the PTS $\forall\mathcal{P}.\mathcal{Q}$ which intuitively captures the "$\mathcal{Q}$-logic of $\mathcal{P}$-terms" and $\mathcal{P}_{\text{Poly}}$ which intuitively denotes the predicative polymorphism extension of $\mathcal{P}$. These results suggest a new approach to the study of the meta-theory of PTSs, by examination of the relationships between different calculi and predicative extensions which allow more expressiveness with equivalent logical strength.

Keywords: Pure Type Systems, Type Theory, weak normalization, conservative extension, predicativity

1 Introduction

When describing a logical system or, as is equivalent through the Curry-Howard lens, a type system, one often wishes to describe a generic situation, in which one wishes not to describe a single construct, but a *family* of constructs ranging over a set of *parameters*, which themselves are particular to the constructs being defined. This is often referred to as a *schema* in logic, as in the description of the induction rule in the usual presentation of Peano Arithmetic. This can be seen as a *meta-level* quantification: the rule is defined for all possible instance of the quantifier. It is then very natural to ask the following question: "is it possible to reify this meta-level quantification?". The immediate practical advantage to such a reification is that it now has a finite description: the meta-level quantification, which can be seen as an infinite conjunction at the object level, is now encapsulated in a single construct of the theory.

In the case that such a reification is possible, the next natural question is this: "is the resulting theory a conservative extension to the original theory?". This question can be quite tricky, and in general depends on what we mean by "reification". Is the reification of the implicit quantification over propositions in Peano Arithmetic second-order Arithmetic or ACA_0, where comprehension is restricted to first-order formulas? In the first case we have a very powerful extension to arithmetic, whereas in the second case, the extension is conservative, comforting us in the feeling that such an extension does not "add anything" to our logic (in particular, the enriched theory is consistent if and only if the original theory is).

Our first contribution is to formalize, in part, a process which allows us to perform such an enrichment. We place ourselves in the framework of Pure Type Systems (PTSs) as described by Barendregt [3]. This framework has the advantage of allowing a very fine and rich account of quantification and dependency. In this framework, there is in general no clear notion of consistency; for this reason we concentrate on normalization/cut elimination, which generally implies consistency in the frameworks in which both concepts exist (by showing that no well-typed normal proof of falsity exists).

The second observation is that there are modular constructions which allow us to combine or extend pure type systems into new systems, and identify certain transformations which preserve weak (and strong) normalization. This suggests a novel approach to describing a logical framework: first identify the components of the framework, e.g. the proof language and the term language, and the relationships between them with respect to quantification. Then use one or several of the combination methods to construct the desired framework. We identify two particularly interesting such constructs: the first takes two PTSs $\mathcal{P}$ and $\mathcal{Q}$, and forms the PTS $\forall\mathcal{P}.\mathcal{Q}$ which informally captures the "$\mathcal{Q}$-logic of $\mathcal{P}$-terms". The second takes a single PTS $\mathcal{P}$ and forms the PTS $\mathcal{P}_{\text{Poly}}$ which adds predicative quantification over every sort of $\mathcal{P}$.

2 Pure Type Systems

Pure type systems are defined as a set of *type assignment systems*, parametrized by the types one is allowed to form. This is prescribed by the dependent function space formation rule, itself entirely described by a triple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ consisting of a set $s, k \in \mathcal{S}$ of *Sorts*, a set $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ of *Axioms* and a set $R \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ of *Rules*.

We use these rules to assign types to terms. The untyped terms and types have the same syntax, which is given by the BNF

$$t, u, A, B \in \Lambda := s \mid x \mid \lambda x\!:\!A.\ t \mid t\ u \mid \Pi x\!:\!A.\ B\,.$$

Conversion is restricted to β-*conversion*: the equivalence relation generated by the contextual closure of the rule

$$(\lambda x : A.\ t)\ u \to_\beta t\{x \mapsto u\}\,.$$

We adopt the usual Barendregt convention for renaming variables in terms and contexts. The typing rules are standard and are given in Figure 1.

Given a PTS $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$, we will write $s_1 : s_2$ for axioms (s_1, s_2) and $s_1 \overset{s_3}{\rightsquigarrow} s_2$ to denote rules (s_1, s_2, s_3). We say that t *has sort* s if there are Γ, A such that $\Gamma \vdash t : A$ and $\Gamma \vdash A : s$ in $\mathcal{P}$.

This deceptively simple framework is in fact quite expressive: it is possible to find instances of PTSs that allow the encoding of very expressive logics like higher-order arithmetic or Zermelo set-theory [16]. In general soundness of these logics can be proven by proving normalization of the corresponding PTS.

$$\textbf{leaf}\ \frac{}{\vdash} \qquad \textbf{wf}\ \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash}\ s \in \mathcal{S},\ x \notin \mathrm{dom}(\Gamma)$$

$$\textbf{var}\ \frac{\Gamma, x : A, \Delta \vdash}{\Gamma, x : A, \Delta \vdash x : A} \qquad \textbf{axiom}\ \frac{\Gamma \vdash}{\Gamma \vdash s_1 : s_2}\ (s_1, s_2) \in \mathcal{A}$$

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

$$\textbf{abs}\ \frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash \Pi x{:}A.\ B : s}{\Gamma \vdash \lambda x{:}A.\ t : \Pi x{:}A.\ B}\ s \in \mathcal{S}$$

$$\textbf{app}\ \frac{\Gamma \vdash t : \Pi x{:}A.\ B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B\{x \mapsto u\}}$$

$$\textbf{conv}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash A' : s}{\Gamma \vdash t : A'}\ A \simeq_\beta A',\ s \in \mathcal{S}$$

Fig. 1. Typing Rules for PTS

Definition 1. *Let $\mathcal{P}$ be a Pure Type System. A term is* well typed *in $\mathcal{P}$ if there is a context Γ and a type A such that*

$$\Gamma \vdash t : A.$$

The PTS $\mathcal{P}$ is weakly normalizing *(resp.* strongly normalizing*) if every well-typed term t in $\mathcal{P}$ has a normal form (resp. there is no infinite chain of reductions starting with t).*

In the remainder of the article, we use normalizing interchangeably with weakly normalizing.

We can consider a PTS to be fully described simply by the triple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$. Using this fact, the class of PTSs can be seen as a *category* where the morphisms between $\mathcal{P}$ and $\mathcal{Q}$ is the set of functions $\phi\colon \mathcal{S}_\mathcal{P} \to \mathcal{S}_\mathcal{Q}$ such that $\phi(s_1) : \phi(s_2)$ whenever $s_1 : s_2$ and $\phi(s_1) \overset{\phi(s_3)}{\rightsquigarrow} \phi(s_2)$ when $s_1 \overset{s_3}{\rightsquigarrow} s_2$. Any such function induces a morphism on terms and contexts which we denote ϕ as well. We then have by simple induction, for every morphism of PTSs $\phi : \mathcal{P} \to \mathcal{Q}$ that

$$\Gamma \vdash_\mathcal{P} t : T \quad \Rightarrow \quad \phi(\Gamma) \vdash_\mathcal{Q} \phi(t) : \phi(T)\,.$$

We can now make our first non-trivial remark:

Remark 1. (Morphisms preserve non-normalization). Let $\mathcal{P}, \mathcal{Q}$ be PTSs. If $\mathcal{Q}$ is WN (resp. SN) and if there is a morphism $\phi : \mathcal{P} \to \mathcal{Q}$, then $\mathcal{P}$ is WN (resp. SN).

This can be seen simply by observing that ϕ preserves β-reduction steps:

$$t \to_\beta t' \Leftrightarrow \phi(t) \to_\beta \phi(t')\,.$$

The converse does not hold, since the terminal object in this category is not normalizing.

Now it is interesting, if not terribly useful, to observe that this category inherits rich structure from that of sets: it admits all limits and co-limits! In particular, it admits products and co-products.

It is immediate that if sorts, axioms and rules are simply restricted, then there is an inclusion morphism from the restricted system to the full system.

The terminal object of the category of PTSs is the "Martin-Löf inconsistent type theory" (see Martin-Löf [14]), which we note $* : *$, which has the unique sort $*$, the axiom $* : *$ and the rule $* \overset{*}{\rightsquigarrow} *$, and was shown to be non-normalizing by Girard [10] (see also Hurkens [11]).

Finally the fact that the co-product of PTSs preserves normalization is non-trivial, and is the object of Theorem 1. It is intuitively clear, however, that every term typed in $\mathcal{P} + \mathcal{Q}$ must be well-typed in either $\mathcal{P}$ or $\mathcal{Q}$. We will make this kind of reasoning precise, and extend it to prove the main results of this work. More generally, we show that we may track the rules which give rise to each redex and show that the subterm which contains the redex is either typable in one of the original systems, or obeys certain combinatorial commutation properties which allow such a redex to be safely eliminated.

The results we prove allow extending pure type systems with certain forms of quantifications while preserving normalization. This is the first step of a *structuralist program* to study pure type systems: rather than trying to find properties that are true of each PTS independently of the others, we study some particular pure type systems, like system F, F_ω or the ECC of Luo [13], which have "atomic" complexity and show that the systems we are interested in can be built using known transformations such as those described above. This approach is quite natural in other fields of algebra, as for example how representations of groups can be classified in terms of irreducible representations.

While we believe that this is the first time such a program has explicitly been stated, there are several instances of such an approach being used in the study of the meta-theory of pure type systems: most notably the work of Bernardy and Lasson [4] has served as inspiration for this approach.

However there have been other instances, as for example the work of Peyton-Jones and Meijer [12] who propose a particular PTS (the Calculus of Constructions) as a possible intermediate language for the `Haskell` programming language. Uncomfortable with the power of the impredicative quantification, they then define a predicative variant by duplicating certain sorts and restricting product formation. We argue that our second main theorem (Theorem 3) addresses exactly this step: the addition of predicative quantification over any given sort of the system preserves normalization.

3 Disjoint Union

To introduce the basic lemmas and techniques that will be used in the next section, we first prove that disjoint unions of PTSs (co-products in the PTS cat-

egory) preserve normalization. To the knowledge of the authors, this observation has never explicitly been stated in the literature.

Theorem 1. *Suppose $\mathcal{P}$ and $\mathcal{Q}$ are two PTSs such that their respective sets of sorts are disjoint. Then the PTS $\mathcal{P}+\mathcal{Q}$ formed by*

$$\mathcal{S}_{\mathcal{P}+\mathcal{Q}} = \mathcal{S}_{\mathcal{P}} \uplus \mathcal{S}_{\mathcal{Q}} \qquad \mathcal{A}_{\mathcal{P}+\mathcal{Q}} = \mathcal{A}_{\mathcal{P}} \uplus \mathcal{A}_{\mathcal{Q}} \qquad \mathcal{R}_{\mathcal{P}+\mathcal{Q}} = \mathcal{R}_{\mathcal{P}} \uplus \mathcal{R}_{\mathcal{Q}}$$

is WN if and only if $\mathcal{P}$ and $\mathcal{Q}$ are WN.

The main difficulty for proving this theorem is to prove that by applying the conversion rule, one cannot move from one PTS to the other. It is possible to prove this directly using subject reduction. However, we prove it using techniques which can more easily generalize to the results in the following sections. We work in a modified presentation of PTSs in which we label well-typed terms with information about which rules and sorts were involved in their construction.

The labeling is similar to a number of labellings used for meta-theoretical studies of pure type systems, see e.g. Melliès and Werner [15], in which they note that it is a crucial device for building models for the Calculus of Constructions.

We want to label each variable $x : A$ with its sort. However, it is not possible in general to attribute a *unique* sort s to A. To circumvent this failure, we refine the classical result on uniqueness of types on functional PTSs in order to characterize the ways in which it may fail in the non-functional case. We define a relation $\sim_\kappa$ on $\mathcal{S}$ that will have the property that if $\Gamma \vdash A : s$ and $\Gamma \vdash A : s'$, then $s \sim_\kappa s'$.

Definition 2. *Given a PTS $(\mathcal{S}, \mathcal{A}, \mathcal{R})$, we define $\sim_\kappa \subseteq \mathcal{S} \times \mathcal{S}$ inductively:*

$$\begin{array}{lllllllll} & & & & & & & & s \sim_\kappa s \\ k : s & \wedge & k' : s' & \wedge & k \sim_\kappa k' & & & \Rightarrow & s \sim_\kappa s' \\ k_1 \sim_\kappa k_1' & \wedge & k_2 \sim_\kappa k_2' & \wedge & k_1 \overset{s}{\rightsquigarrow} k_2 & \wedge & k_1' \overset{s'}{\rightsquigarrow} k_2' & \Rightarrow & s \sim_\kappa s' \,. \end{array}$$

Note that this relation is reflexive and symmetric, but not transitive in general. However, we may easily turn $\sim_\kappa$ into an equivalence relation by taking the transitive closure $\sim_\kappa^*$. This allows us to take the equivalence classes of sorts modulo $\sim_\kappa^*$; the class of a sort s will be denoted $\overline{s}$. Similarly, for rules $r = s_1 \overset{s_3}{\rightsquigarrow} s_2$ we write $\overline{r} = \overline{s_1} \overset{\overline{s_3}}{\rightsquigarrow} \overline{s_2}$. In the rest of this document, we write $\sim$ instead of $\sim_\kappa^*$.

We notice that taking equivalence classes of sorts gives rise to a functional PTS $\mathcal{P}_{\text{fun}}$ defined by

$$\overline{\mathcal{S}} = \left\{ \overline{s} \mid s \in \mathcal{S} \right\} \qquad \overline{\mathcal{A}} = \left\{ \overline{k} : \overline{s} \mid (k, s) \in \mathcal{A} \right\} \qquad \overline{\mathcal{R}} = \left\{ \overline{r} \mid r \in \mathcal{R} \right\}.$$

It is straightforward to verify that this is indeed a functional PTS, and that there is a morphism $\phi : \mathcal{P} \to \mathcal{P}_{\text{fun}}$ that sends s to $\overline{s}$.

Lemma 1. *In every PTS $\mathcal{P}$, if $\Gamma \vdash A : s, s'$, then $s \sim s'$.*

Similarly, if t has both sorts s and s' then $s \sim s'$.

Proof of Lemma 1. We only prove the first statement, the second is proven similarly. Given the morphism ϕ defined above, from the statement

$$\Gamma \vdash A : s, s'$$

we have

$$\phi(\Gamma) \vdash \phi(A) : \phi(s), \phi(s')$$

in $\mathcal{P}_{\text{fun}}$. But in functional PTSs, we have unicity of types modulo β-conversion (Barendregt [3], Lemma 5.2.21), which gives:

$$\phi(s) = \overline{s} \simeq_\beta \overline{s'} = \phi(s)$$

confluence of β-conversion gives $\overline{s} = \overline{s'}$, which is what we needed. □

This observation allows us to give an alternative version of PTSs with rule-labeled abstraction, application and products and sort-labeled variables. This system will allow extraction of sort information by straightforward induction on terms.

Definition 3. *Let $\mathcal{P}$ be a PTS. Define the* $\mathcal{P}$-labeled calculus $\overline{\mathcal{P}}$ *with the* labeled terms

$$t, u, A, B \in \Lambda_{\text{lab}} := s \mid x^{\overline{s}} \mid \lambda^{\overline{r}} x^{\overline{s}} : A.\ t \mid (t\ u)^{\overline{r}} \mid \Pi^{\overline{r}} x^{\overline{s}} : A.\ B$$

where $s \in \mathcal{S}$ and $r \in \mathcal{R}$.

We define the unlabeling $|t|$ *to be the term t in which all sort and rule labels are removed.*

We define the typing judgment $\vdash_{\text{lab}}$ as consisting of (the obvious labeling of) the rules given in Figure 1 with the following modifications:

$$\textbf{wf}\ \frac{\Gamma \vdash_{\text{lab}} A : s}{\Gamma, x^{\overline{s}} : A \vdash_{\text{lab}}}\ s \in \mathcal{S},\ x \notin \text{dom}(\Gamma)$$

$$\textbf{prod}\ \frac{\Gamma \vdash_{\text{lab}} A : s_1 \qquad \Gamma, x^{\overline{s_1}} : A \vdash_{\text{lab}} B : s_2}{\Gamma \vdash_{\text{lab}} \Pi^{\overline{r}} x^{\overline{s_1}} : A.\ B : s_3}\ r = s_1 \overset{s_3}{\rightsquigarrow} s_2 \in \mathcal{R}$$

$$\textbf{abs}\ \frac{\Gamma, x^{\overline{s_1}} : A \vdash_{\text{lab}} t : B \qquad \Gamma \vdash_{\text{lab}} \Pi^{\overline{r}} x : A.\ B : s_3}{\Gamma \vdash_{\text{lab}} \lambda^{\overline{r}} x^{\overline{s_1}} : A.\ t : \Pi^{\overline{r}} x^{\overline{s_1}} : A.\ B}\ r = s_1 \overset{s_3}{\rightsquigarrow} s_2 \in \mathcal{R}$$

$$\textbf{app}\ \frac{\Gamma \vdash_{\text{lab}} t : \Pi^{\overline{r}} x^{\overline{s_1}} : A.\ B \qquad \Gamma \vdash_{\text{lab}} u : A}{\Gamma \vdash_{\text{lab}} (t\ u)^{\overline{r}} : B\{x^{\overline{s_1}} \mapsto u\}}\ r = s_1 \overset{s_3}{\rightsquigarrow} s_2$$

$$\textbf{conv}\ \frac{\Gamma \vdash_{\text{lab}} t : A \qquad \Gamma \vdash_{\text{lab}} A : s_1 \qquad \Gamma \vdash_{\text{lab}} A' : s_2}{\Gamma \vdash_{\text{lab}} t : A'}\ |A| \simeq_\beta |A'|,\ s_1 \sim s_2$$

We remove labels in the conversion to simplify the meta-theory: if we had introduced the rule $((\lambda^{\overline{r}} x : A.\ t)\ u)^{\overline{r}} \to_\beta t\{x \mapsto u\}$ then confluence would fail on ill-typed terms, making the meta-theory more complex, and our completeness result below significantly more difficult to prove.

A *labeling* of an unlabeled term t is a labeled term $\hat{t}$ such that $\left|\hat{t}\right| = t$. We extend labeling and unlabeling to contexts in the obvious manner.

In the following section, we fix a given PTS $\mathcal{P}$. The next lemma is immediate by induction on the derivation.

Lemma 2. *Suppose that* $\Gamma \vdash_{\text{lab}} t : A$ *in* $\mathcal{P}$. *Then* $|\Gamma| \vdash |t| : |A|$.

There is a clear characterization of the sort of a well-typed term, that is on the type of its type, or simply its type if that is a top-sort.

Lemma 3. *Suppose that* $\Gamma \vdash_{\text{lab}} t : A$. *Then there is* $s \in \mathcal{S}_\mathcal{P}$ *such that either* $\Gamma \vdash_{\text{lab}} A : s$ *or* $A = s$.

Lemma 4. *The judgment* $\Gamma \vdash t : A$ *is derivable in* $\mathcal{P}$ *if and only if there is a (unique) labeling* $\widehat{\Gamma}, \hat{t}$ *and* $\widehat{A}$ *such that* $\widehat{\Gamma} \vdash_{\text{lab}} \hat{t} : \widehat{A}$ *in* $\overline{\mathcal{P}}$.

Given this theorem, we will often write t for both $|t|$ and $\hat{t}$ indistinguishably for a given well-typed term t, and $\vdash$ instead of $\vdash_{\text{lab}}$. This more explicit type-system allows us to give a straightforward proof of Theorem 1 by induction over the labeled type derivation.

Proof of Theorem 1. Suppose $\mathcal{P} + \mathcal{Q}$ is WN. The inclusions $i_1 \colon \mathcal{P} \to \mathcal{P} + \mathcal{Q}$ and $i_2 \colon \mathcal{Q} \to \mathcal{P} + \mathcal{Q}$ are morphisms, which implies that $\mathcal{P}$ and $\mathcal{Q}$ are WN.

Now suppose that $\mathcal{P}$ and $\mathcal{Q}$ are WN and let $\Gamma \vdash_{\mathcal{P}+\mathcal{Q}} t : A$. Then we have $\Gamma \vdash A : s$ or $A = s$. W.l.o.g. we may suppose that $s \in \mathcal{S}_\mathcal{P}$. Let Δ be the subset of Γ with only the type declarations of the form

$$x^{\overline{k}} : B$$

for $k \in \mathcal{S}_\mathcal{P}$. We show by induction on the typing derivation of t in $\overline{\mathcal{P} + \mathcal{Q}}$ that

$$\Delta \vdash_\mathcal{P} t : A$$

is derivable.

Now we can conclude that t is weakly normalizable, as it is typable in $\mathcal{P}$. □

4 The PTS $\forall\mathcal{P}.\mathcal{Q}$

The main result of this section is an extension of the previous one. We wish for not only $\mathcal{P}$ and $\mathcal{Q}$ to coexist, but for $\mathcal{Q}$ types to be built by quantification over $\mathcal{P}$ types.

Theorem 2. *Let $\mathcal{P}$ and $\mathcal{Q}$ be as in Theorem 1. Let $\forall\mathcal{P}.\mathcal{Q}$ be the PTS $\mathcal{P}+\mathcal{Q}$ with the additional rules:*

$$\mathcal{I} = \left\{ s \overset{k}{\rightsquigarrow} k \mid s \in \mathcal{S}_\mathcal{P},\ k \in \mathcal{S}_\mathcal{Q} \right\}.$$

Then $\forall\mathcal{P}.\mathcal{Q}$ is WN iff $\mathcal{P}$ and $\mathcal{Q}$ are WN.

Now suppose we call $\mathcal{P}$-types *term(-sets)* and $\mathcal{Q}$-types *propositions*. Then the rules we introduced allow us to build $\mathcal{Q}$-propositions which depend on $\mathcal{P}$-terms.

The proof of Theorem 2 is more involved than that of Theorem 1 as there are non-trivial interactions between the two systems generated by the rules in $\mathcal{I}$. Our proof is directly adapted from Bernardy and Lasson [4].

The idea is to split each term into subterms typable in $\mathcal{P}$ and *erased terms* typable in $\mathcal{Q}$. The "interaction redexes" built by the rules in $\mathcal{I}$ will be handled separately, as they strictly decrease in number after each such β-reduction, and can not duplicate $\mathcal{Q}$-redexes.

This proof bears many similarities with the Geuvers and Nederhof's [9] proof that normalization of system F_ω implies that of the Calculus of Constructions (Barendregt [3] Theorem 5.3.14), which tends to indicate that these proofs are instances of a general approach based on erasure and labeling.

Definition 4 (Erasure). *Let $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ be a PTS and $D \subseteq \mathcal{R}$ (a set of* dependencies*). Suppose in addition that D is closed under $\sim$; that is if $r \in D$ and $\overline{r} = \overline{r'}$ then $r' \in D$. The D-erasure $\lfloor t \rfloor^D$ of a term t is defined by induction on the labeled term:*

$$\begin{array}{rcll}
\lfloor s \rfloor^D & = & s & \\
\lfloor x^{\overline{s}} \rfloor^D & = & x^{\overline{s}} & \\
\lfloor \Pi^{\overline{r}} x^{\overline{s}} : t.\ u \rfloor^D & = & \lfloor u \rfloor^D & \text{if } r \in D \\
\lfloor \Pi^{\overline{r}} x^{\overline{s}} : t.\ u \rfloor^D & = & \Pi^{\overline{r}} x^{\overline{s}} : \lfloor t \rfloor^D.\ \lfloor u \rfloor^D & \text{otherwise} \\
\lfloor \lambda^{\overline{r}} x^{\overline{s}} : t.\ u \rfloor^D & = & \lfloor u \rfloor^D & \text{if } r \in D \\
\lfloor \lambda^{\overline{r}} x^{\overline{s}} : t.\ u \rfloor^D & = & \lambda^{\overline{r}} x^{\overline{s}} : \lfloor t \rfloor^D.\ \lfloor u \rfloor^D & \text{otherwise} \\
\lfloor (t\ u)^{\overline{r}} \rfloor^D & = & \lfloor t \rfloor^D & \text{if } r \in D \\
\lfloor (t\ u)^{\overline{r}} \rfloor^D & = & (\lfloor t \rfloor^D\ \lfloor u \rfloor^D)^{\overline{r}} & \text{otherwise.}
\end{array}$$

We sometimes omit the superscript if it is clear in the context.

Note that in general, the D-erasure of a well-typed term is not well typed, or indeed, even stable by reduction (or variable binding). However in the current case, we have enough structure to guarantee typability of erased terms. In the following section we fix $D = \mathcal{R}_\mathcal{P} \cup \mathcal{I}$.

We want to distinguish "$\mathcal{P}$-terms" from "$\mathcal{Q}$-terms."

Definition 5. *Suppose that $\Gamma \vdash_{\forall\mathcal{P}.\mathcal{Q}} t : A$. We say that t* has a sort in $\mathcal{P}$ *(resp. $\mathcal{Q}$) when there is a sort $s \in \mathcal{S}_\mathcal{P}$ (resp. $\mathcal{S}_\mathcal{Q}$) such that either $\Gamma \vdash_{\forall\mathcal{P}.\mathcal{Q}} A : s$ or $A = s$.*

By Lemma 3 we know that every well-typed term in $\forall\mathcal{P}.\mathcal{Q}$ has a sort in either $\mathcal{P}$ or $\mathcal{Q}$. Using Lemma 1 and Lemma 5 (below) we know that a term cannot have both.

Lemma 5. *Suppose $s \sim s'$ in $\forall\mathcal{P}.\mathcal{Q}$. Then $s, s' \in \mathcal{S}_\mathcal{P}$ or $s, s' \in \mathcal{S}_\mathcal{Q}$.*

We have three kind of redexes in terms, the redexes from rules in $\mathcal{P}$, from rules in $\mathcal{Q}$ and the redexes in $\mathcal{I}$.

Definition 6. *Given a term t well typed in $\forall\mathcal{P}.\mathcal{Q}$, we say that the redex at position p is a $\mathcal{P}$-redex, resp. $\mathcal{Q}$-redex, $\mathcal{I}$-redex if the redex is of shape*

$$((\lambda^{\overline{r}} x^{\overline{s}} : A.t)\ u)^{\overline{r}}$$

with $r \in \mathcal{R}_\mathcal{P}$, resp. $\mathcal{R}_\mathcal{Q}$, resp. $\mathcal{I}$.

We will sometimes write $t \to_\mathcal{P} t'$ if t' is obtained from t by contraction of a $\mathcal{P}$-redex (similarly for $\mathcal{Q}$- and $\mathcal{I}$-redexes).

Furthermore, conversion is preserved by erasure on well-typed terms.

Lemma 6. *Suppose t, t' are well typed in $\forall\mathcal{P}.\mathcal{Q}$ with sort in $\mathcal{Q}$. Then*

$$t \to_\mathcal{Q} t' \quad \Rightarrow \quad \lfloor t \rfloor \to_\beta \lfloor t' \rfloor .$$

Proof of Lemma 6. We only treat head reduction: in that case we have $t = ((\lambda^{\overline{r}} x^{\overline{s}} : A.\ t_1)\ t_2)^{\overline{r'}}$. Well typedness gives $\overline{r} = \overline{r'}$. Again we treat the three cases:

1. $r \in \mathcal{R}_\mathcal{P}$. This is not possible as t has a sort in $\mathcal{Q}$.
2. $r \in \mathcal{R}_\mathcal{Q}$. In this case we have

$$\lfloor t \rfloor = ((\lambda^{\overline{r}} x^{\overline{s}} : \lfloor A \rfloor.\ \lfloor t_1 \rfloor)\ \lfloor t_2 \rfloor)^{\overline{r}} \to_\beta \lfloor t_1 \rfloor \{x^{\overline{s}} \mapsto \lfloor t_2 \rfloor\} = \lfloor t_1 \{x^{\overline{s}} \mapsto t_2\} \rfloor$$

   where the last equality is proven by simple induction over the structure of t_1.
3. $r \in \mathcal{I}$. In this case we have $x^{\overline{s}} \notin \mathrm{FV}(\lfloor t_1 \rfloor)$ This gives

$$\lfloor t \rfloor = \lfloor t_1 \rfloor = \lfloor t_1 \rfloor \{x^{\overline{s}} \mapsto \lfloor t_2 \rfloor\} = \lfloor t_1 \{x^{\overline{s}} \mapsto t_2\} \rfloor = \lfloor t' \rfloor .$$

□

In particular, due to confluence of β-reduction, we have

$$t \simeq_\beta t' \quad \Rightarrow \quad \lfloor t \rfloor \simeq_\beta \lfloor t' \rfloor$$

for terms with sort in $\mathcal{Q}$.

Now this allows us to show that well-typed terms in $\forall\mathcal{P}.\mathcal{Q}$ are either well typed in $\mathcal{P}$ or their erasure is well typed in $\mathcal{Q}$.

Proposition 1. *Suppose $\Gamma \vdash_{\forall\mathcal{P}.\mathcal{Q}} t : A$. We have the following:*

1. *If t has a sort in $\mathcal{P}$, then there is a subcontext Δ of Γ such that $\Delta \vdash_\mathcal{P} t : A$.*
2. *If t has a sort in $\mathcal{Q}$, then we have $\lfloor \Gamma \rfloor \vdash_\mathcal{Q} \lfloor t \rfloor : \lfloor A \rfloor$.*

Proof of Proposition 1. First suppose t has a sort in $\mathcal{P}$. Choose Δ to be the declarations $x^k : B$ in Γ with $k \in \mathcal{S}_\mathcal{P}$. Then we proceed by induction on the type derivation of the labeled term t. The proof is similar to that of Theorem 1.

Now if t has a sort in $\mathcal{Q}$ we proceed similarly. The only difficult case is conversion, which is handled by appeal to Lemma 6. □

Lemma 7. *If t is well-typed term in $\forall\mathcal{P}.\mathcal{Q}$ then $\to_{\mathcal{I}}$ reductions of t are finite.*

Proof of Lemma 7. We will show that the number of $\mathcal{I}$-abstractions will strictly decrease when contracting a $\mathcal{I}$-redex. Let $((\lambda^{\overline{r}}x^{\overline{s}} : T.t)\ u)^{\overline{r}}$ be the redex in question. We have by Inversion and Proposition 1 that

- u is of sort $s \in \mathcal{S}_{\mathcal{P}}$
- u is well typed in $\mathcal{P}$

In particular, u can contain no subterm of the form $\lambda^{\overline{r'}}x^{\overline{s'}} : A.v$ with $r' \in \mathcal{I}$. This means that no $\mathcal{I}$-abstraction can be duplicated. So the number of $\mathcal{I}$-abstractions in t must strictly decrease at each $\mathcal{I}$-reduction step, which implies termination of $\mathcal{I}$-reductions. □

The converse of Lemma 6 is not true in general, as $\mathcal{I}$-redexes can "hide" possible $\mathcal{Q}$-redexes, illustrated in the following example.

Example 1. Consider the following term:

$$t = (\lambda x^{\mathcal{P}} : A.(\lambda y^{\mathcal{Q}} : B.u_1^{\mathcal{Q}})^{\mathcal{R}_{\mathcal{Q}}})^{\mathcal{I}} u_2^{\mathcal{P}} u_3^{\mathcal{Q}}.$$

The sort labels mean that the variable/term belongs to a sort in that set, and the rule annotations means the rule belongs to that set (for clarity we didn't annotate the applications). It is possible to make explicit choices such that t is well-typed. Note that

$$\lfloor t \rfloor = (\lambda y^{\mathcal{Q}} : B.u_1)^{\mathcal{R}_{\mathcal{Q}}} u_3 \ \to_\beta \ u_1\{y \mapsto u_3\}.$$

However, t is in $\to_{\mathcal{Q}}$ normal form. The $\mathcal{Q}$-redex is hidden by the $\mathcal{I}$-redex in t. In contrast, it is not possible for $\mathcal{P}$-redexes to create $\mathcal{Q}$- or $\mathcal{I}$-redexes.

To show that terms typable in $\forall\mathcal{P}.\mathcal{Q}$ have $(\mathcal{Q} \cup \mathcal{I})$-normal forms, we need to lift reductions in the erased domain up to the richer pure type system. The crucial observation is the following:

Lemma 8. *Suppose t is well typed in $\forall\mathcal{P}.\mathcal{Q}$ with a sort in $\mathcal{Q}$ and suppose that $\lfloor t \rfloor \to_\beta v$. Then there exists a term $\tilde{v}$ such that $\lfloor \tilde{v} \rfloor = v$ and*

$$t \to_{\mathcal{I}}^* \to_{\mathcal{Q}} \tilde{v}\,.$$

These lemmas allow us to prove Theorem 2.

Proof of Theorem 2. Suppose that t is well typed in $\forall\mathcal{P}.\mathcal{Q}$. If t has a sort in $\mathcal{P}$, then t is well typed in the PTS $\mathcal{P}$ by Proposition 1 and we are done. Otherwise, t has a sort in $\mathcal{Q}$, and we proceed as follows. We will first find a $\to_{\mathcal{QI}}$ normal form. Since $\lfloor t \rfloor$ is typable in $\mathcal{Q}$ (again by Proposition 1) and $\mathcal{Q}$ is weakly normalizing, there exists a $\mathcal{Q}$-normal form t_1 of $\lfloor t \rfloor$. By Lemma 8 we can lift every step of this reduction chain to $\forall\mathcal{P}.\mathcal{Q}$ by adding $\mathcal{I}$-reductions. This way we obtain a lift $\tilde{t}_1$ of t_1 such that $t \to_{\mathcal{QI}}^* \tilde{t}_1$. Lemma 7 tells us that we can find a $\mathcal{I}$ normal form

t' of $\tilde{t}_1$. Since contracting $\mathcal{I}$-redexes doesn't change the erasure, we know that $\lfloor t' \rfloor = \lfloor \tilde{t}_1 \rfloor = t_1$. By Lemma 6 we conclude that t' is also in $\mathcal{Q}$ normal form. Hence t' is a $\to_{\mathcal{QI}}$ normal form of t.

Now we prove by induction on terms in $\to_{\mathcal{QI}}$ normal form that they have a $\to_\beta$ normal form. The only interesting case is when the term is an application which is part of a redex. Since the term is in $\to_{\mathcal{QI}}$ normal form, this must be a $\mathcal{P}$-redex. By Proposition 1 this means that the term is well-typed in $\mathcal{P}$, so it has a $\to_{\mathcal{P}}$ normal form. Since contracting $\mathcal{P}$-redexes cannot create $\mathcal{I}$- or $\mathcal{Q}$-redexes, this means the normal form is actually an $\to_\beta$ normal form. This completes the induction, which shows that t', and hence t has a $\to_\beta$ normal form. □

Additionally, Proposition 1 gives us the following logical conservativity result.

Corollary 1. *Suppose A is well typed in $\forall\mathcal{P}.\mathcal{Q}$ of type $s \in \mathcal{S}_{\mathcal{Q}}$. Suppose furthermore that A only contains subterms which have a sorts in $\mathcal{Q}$. Then we have*

$$A \text{ is inhabited in } \forall\mathcal{P}.\mathcal{Q} \quad \textit{iff} \quad A \text{ is inhabited in } \mathcal{Q}.$$

5 The PTS $\mathcal{P}_{\mathbf{Poly}}$

In this section, we show that we may extend a PTS $\mathcal{P}$ with quantification of every sort over every other sort, provided the *result* lives in a "fresh" sort. This allows internalizing quantification over free variables: in general if a term t contains a free variable x of sort s, one may instantiate x with any term u of the same type. However it is not in general possible to quantify over x. The following result shows that it is possible to safely form the term $\lambda x : T.\ t$ (if $x : T$) within the theory, by pushing the resulting term into a new sort. This is sometimes referred to as *predicative polymorphism*, *ML-style polymorphism* or *prenex polymorphism*. We feel that it is natural to try to capture such a concept in its most general form.

Additionally, such a practice seems quite useful in general for extensions of type theory with such things as size-types (See Blanqui [5] or Abel [1]) or universes [17] in order to obtain an object theory that naturally allows terms polymorphic in sizes or universes. Note that these particular extensions are impossible in the theory of *pure* type systems, but there is good hope that our approach still applies.

In the following we fix a PTS $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$.

Definition 7. *Let ${s_2}^{s_1} \notin \mathcal{S}$, be a new sort for each pair of sorts $s_1, s_2 \in \mathcal{S}$. We define the PTS $\mathcal{P}_{\text{Poly}} = (\mathcal{S}_{\text{Poly}}, \mathcal{A}_{\text{Poly}}, \mathcal{R}_{\text{Poly}})$ by*

$$\mathcal{S}_{\text{Poly}} = \mathcal{S} \cup \left\{ {s_2}^{s_1} \mid s_1, s_2 \in \mathcal{S} \right\}$$

$$\mathcal{A}_{\text{Poly}} = \mathcal{A}$$

$$\mathcal{R}_{\text{Poly}} = \mathcal{R} \cup \left\{ s_1 \overset{{s_2}^{s_1}}{\rightsquigarrow} s_2 \mid s_1, s_2 \in \mathcal{S} \right\} \cup \left\{ s_1 \overset{{s_2}^{s_1}}{\rightsquigarrow} {s_2}^{s_1} \mid s_1, s_2 \in \mathcal{S} \right\}.$$

This construction also preserves normalization, by a similar argument to above.

Theorem 3. *If $\mathcal{P}$ is WN, then so is $\mathcal{P}_{\text{Poly}}$.*

Proof of Theorem 3. (Sketch)

We identify, in the same manner as in the proof of Theorem 2, two types of redexes: those coming from a rule in $\mathcal{R}$, and those coming from the new rules, which we call $\mathcal{P}$-redexes and $\mathcal{I}$-redexes respectively.

In the same manner as before, we can show

1. Every $\mathcal{P}$-redex belongs to a subterm typable in $\mathcal{P}$.
2. Reducing a $\mathcal{I}$-redex can not create *any* redexes.

Due to this observation, and subject reduction, we may proceed as previously and reduce every $\mathcal{I}$-redex, then normalize each subterm that contains a $\mathcal{P}$-redex. Both operations normalize by the above observation, and the final term is in normal form.

6 Examples

We may verify that the PTS $\mathcal{P}^2$ given in Bernardi and Lasson [4] is a sub-PTS of $\forall\mathcal{P}.\ \mathcal{P}'$ (where $\mathcal{P}'$ is a renaming of $\mathcal{P}$ to make it disjoint from the latter), and as such, is normalizing if $\mathcal{P}$ is. The only-if direction does not follow immediately, as a sub-PTS of a non-normalizing PTS may be normalizing. In the case of that paper however, it is trivial to verify that it in fact does.

It is also easy to use the predicative polymorphism transformation to turn the simply-typed λ-calculus (STLC) into a calculus with ML-style polymorphism: define STLC to be the PTS defined by

$$\mathcal{S} = \{*, \Box\} \qquad \mathcal{A} = \{* : \Box\} \qquad \mathcal{R} = \left\{* \overset{*}{\rightsquigarrow} *\right\}.$$

In the PTS $\text{STLC}_{\text{Poly}}$ we can for example form the polymorphic term

$$\text{id} = \lambda X{:}*.\ \lambda x{:}X.\ x \ :\ \Pi X{:}*.\ X \to X\,.$$

By use of the rule $\Box \overset{*}{\rightsquigarrow}{}^{\Box} *$.

It is amusing to note that there is the rule $* \overset{\Box}{\rightsquigarrow}{}^{*} \Box$ in $\text{STLC}_{\text{Poly}}$, which seems to allow for the construction of dependent types. However this ability is quite restricted, in opposition to "true" dependent types as those in the $\lambda\Pi$-calculus or Martin-Löf Type Theory.

Unfortunately, we do not quite have the predicative system described by Peyton-Jones and Meijer [12], as they have the additional rules

$$*^{\Box} \overset{*}{\rightsquigarrow}{}^{\Box} * \qquad *^{\Box} \overset{*}{\rightsquigarrow}{}^{\Box} *^{\Box}\,.$$

We do not know whether such rules can be added in a general way to every PTS.

Now let us give a slightly more elaborate example in which we construct a system of interest out of more elementary systems. We wish to build a predicate logic over terms. We therefore consider an elementary PTS with a single sort $*^s$ which represents the universe of basic sets of terms, and a sort $\Box^s$ with $*^s : \Box^s$ which allows declaring type variables such as

$$\mathrm{Nat} : *^s, \quad \mathrm{Bool} : *^s .$$

We call TERM this PTS with two sorts, one axiom and no rules. It is easy to see that there are no possible λ-abstractions in this PTS, and so every term is trivially normalizing.

As this is a quite poor framework in which to define even first order terms, we add function spaces to be able to declare variables of a function type, e.g. $S : \mathrm{Nat} \to \mathrm{Nat}$. To do this, we add a third sort, $*^f$, to represent function spaces, along with the rules

$$*^s \overset{*^f}{\rightsquigarrow} *^s, \; *^s \overset{*^f}{\rightsquigarrow} *^f .$$

This allows us to declare variables of functional type such as $S : \mathrm{Nat} \to \mathrm{Nat}$ in a well-formed context. But this new PTS, which we call $\mathrm{TERM}_{\mathrm{ext}}$, is just a sub-PTS of $\mathrm{TERM}_{\mathrm{Poly}}$, where $*^f = *^{s*^s}$! Without any additional work, we can therefore conclude, using Theorem 3 that this PTS admits normalization.

Now we wish to reason about such terms using a propositional framework. If we choose that framework to be STLC, then we can simply form the sub-PTS of $\forall\mathrm{TERM}_{\mathrm{ext}}$. STLC obtained by adding the rules

$$*^s \overset{\Box}{\rightsquigarrow} \Box \quad *^s \overset{*}{\rightsquigarrow} *$$

to obtain a dependently-typed system which captures the $\forall, \Rightarrow$ fragment of (minimal, intuitionistic) first-order logic. This system admits cut elimination by Theorem 2 and normalization of STLC. Such a system was in fact described by Berardi (see Barendregt [3]) by a direct construction, and in the above reference cut elimination is derived by translation into a system with dependent types, rather than our modular approach.

We can also apply Theorem 3 a second time to obtain a system with the additional rule $\Box \overset{*^\Box}{\rightsquigarrow} *$. In this system, we are now able to express axiom schemas: in the context

$$\Gamma = \mathrm{Nat} : *^s, \; 0 : \mathrm{Nat}, \; S : \mathrm{Nat} \to \mathrm{Nat}$$

we have

$$\Gamma \vdash \Pi P : \mathrm{Nat} \to *. \; P\ 0 \to (\Pi n : \mathrm{Nat} \,.\, P\ n \to P\ (S\ n)) \to \Pi m : \mathrm{Nat} \,.\, P\ m \;:\; *^\Box$$

which allows us to build a well-formed context with a variable *ind* of that type. This is possible without fear of losing normalization under the β-rule. Note that this fact does not help much when trying to prove meta-theoretical properties about arithmetic like consistency, which requires more elaborate cut-elimination rules.

One can iterate this construction to get an arbitrary number of "universes". However, since there is no rule $*^{\Box} \overset{*}{\rightsquigarrow}_{\Box} *^{\Box}$, the resulting system is weaker than the usual Martin-Löf type theory with universes.

7 Conclusions

We have presented various operations allowing one to combine or extend arbitrary PTSs, and shown certain of these combinations to preserve normalization.

On the technical side, it is clear that sort-labeling and erasure are powerful techniques for proving properties about reduction in pure type systems, and more investigation is warranted to understand the extent of these techniques. Ideally we would like a general syntactic theorem (which depends only on combinatorial properties of $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$) which captures the extensions to a system (or a set of systems) that can be proven sound with this method.

Natural applications of this approach include the analysis of dependently-typed programming languages. Such languages aim to model programs and proofs using a single framework. However, the construction of a proof language and of a programming language are often at odds, as there are many features of an environment for proofs (impredicativity, normalization, irrelevance) which are not desirable for a programming environment. One approach is to compartmentalize the system into two (or more) universes, along with sometimes complex rules to guide their interaction. In particular the *Trellys* project (see e.g. Sjöberg et al. [18]) aims at exploring the consequences of such distinctions. We believe that our approach may allow a systematic study of these interactions, lightening the burden of meta-theoretical study.

There has been some effort concerning the use of dependently-typed languages to serve as a framework in which to recast, or replay proofs done in different, more complex systems. The language `Dedukti` [8], for instance, has been used to embed proofs coming from Coq [6], and HOL [2] using a suitable encoding. It is a natural question to ask whether the combination of these encodings is still coherent, or more generally under which conditions one can combine such encodings. While in general this question is quite difficult, our Theorem 1, and to a lesser extent Theorem 3 can be seen as a first step in that direction.

All the theorems in this paper can be generalized to hold with strong normalization as well, by adapting the proof to use well-known modularity results in the theory of rewrite systems. We concentrate on weak normalization, as it is sufficient to imply consistency of logical systems based on PTS.

Acknowledgments. We would like to thank Jean-Philippe Bernardy and Marc Lasson for many useful comments and corrections concerning a first draft of this paper, as well as the numerous helpful comments from the anonymous reviewers.

References

1. Abel, A. *A Polymorphic Lambda-Calculus with Sized Higher-Order Types*. PhD thesis, Ludwig-Maximilians-Universität München, 2006.
2. Assaf, A., and Burel, G. The Holide home page. https://www.rocq.inria.fr/deducteam/Holide/index.html.
3. Barendregt, H. Lambda calculi with types. In *Handbook of logic in computer science*, S. Abramsky, D. Gabbay, and T. Maibaum, Eds., vol. 2. Oxford University Press, 1992.
4. Bernardy, J.-P., and Lasson, M. Realizability and parametricity in pure type systems. In *Foundations of Software Science and Computational Structures*, M. Hofmann, Ed., vol. 6604 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2011, pp. 108–122.
5. Blanqui, F. A type-based termination criterion for dependently-typed higher-order rewrite systems. In *Proc. of the 15th International Conference on Rewriting Techniques and Applications* (2004), vol. 3091 of *Lecture Notes in Computer Science*.
6. Boespflug, M., and Burel, G. CoqInE: Translating the calculus of inductive constructions into the lambda pi-calculus modulo. In *Proceedings of the Second International Workshop on Proof Exchange for Theorem Proving* (2012), D. Pichardie and T. Weber, Eds., vol. 878 of *CEUR Workshop Proceedings*, CEUR-WS.org.
7. Coquand, T. An analysis of Girard's paradox. In *Proceedings of LICS'86* (1986), IEEE Computer Society Press, pp. 227–236.
8. Deducteam. *The Dedukti Reference Manual, version 1.0*, 2012.
9. Geuvers, H., and Nederhof, M. A modular proof of strong normalisation for the Calculus of Constructions. *Journal of Functional Programming 1*, 2 (1991), 155–189.
10. Girard, J.-Y. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris VII, 1972.
11. Hurkens, A. A Simplification of Girard's Paradox. In *Proceedings of TLCA'95* (1995), M. Dezani-Ciancaglini and G. Plotkin, Eds., vol. 902, pp. 266–278.
12. Jones, S., and Meijer, E. Henk: a typed intermediate language, 1997.
13. Luo, Z. ECC: An Extended Calculus of Constructions. In *Proc. of LICS* (1990), pp. 385–395.
14. Martin-Löf, P. An intuitionistic theory of types. Tech. Rep. TR 72-?, University of Stockholm, 1972.
15. Melliès, P.-A., and Werner, B. A generic proof of strong normalisation for pure type systems. In *Proceedings of TYPES'96* (1998), E. Giménez and C. Paulin-Mohring, Eds., vol. 1512.
16. Miquel, A. *Le Calcul des Constructions implicite: syntaxe et sémantique*. PhD thesis, Université Paris 11, 2001.
17. Palmgren, E. On universes in type theory. In *Twenty-five years of constructive type theory (Venice, 1995)*, vol. 36 of *Oxford Logic Guides*. Oxford Univ. Press, New York, 1998, pp. 191–204.
18. Sjöberg, V., Casinghino, C., Ahn, K. Y., Collins, N., III, H. D. E., Fu, P., Kimmell, G., Sheard, T., Stump, A., and Weirich, S. Irrelevance, heterogeneous equality, and call-by-value dependent type systems. In *MSFP* (2012), J. Chapman and P. B. Levy, Eds., vol. 76 of *EPTCS*, pp. 112–162.