Domain-Free Pure Type Systems

Gilles Barthe[1] & Morten Heine Sørensen[2]

[1] Centrum voor Wiskunde en Informatica (CWI)
PO Box 94079, 1090 GB Amsterdam, The Netherlands, gilles@cwi.nl

[2] Department of Computer Science University of Copenhagen (DIKU)
Universitetsparken 1, DK-2100 Copenhagen Ø, Denmark, rambo@diku.dk

Abstract. Pure type systems feature domain-specified λ-abstractions $\lambda x : A.M$. We present a variant of pure type systems, which we call domain-free pure type systems, with domain-free λ-abstractions $\lambda x.M$. We study the basic properties of domain-free pure type systems and establish their formal relationship with pure type systems.

1 Introduction

Typed versions of the λ-calculus were introduced independently by Church [6] and Curry [7]. In Church's system abstractions have *domains*, i.e. are of the form $\lambda x{:}A.t$, whereas in Curry's system abstractions have no domain, i.e. are of the form $\lambda x.t$. Over the years, many type systems have appeared, the majority of which use domain-specified abstractions. Barendregt and others give an abstract, unifying view of type systems with domain-specified abstractions in terms of the notion of *pure type system*—see e.g. [2, 3, 8, 9]. In this paper, we consider for every pure type system a domain-free version in which abstractions have no domain. We call such systems *domain-free pure type systems*.[3] The main technical contribution of the paper—expressed in Theorem 26—states, under suitable hypotheses, a connection between a pure type system and its corresponding domain-free pure type system. Domain-free pure type systems and Theorem 26 have proved useful for defining continuation-passing style translations for pure type systems [4], for proving strong normalisation from weak normalisation of pure type systems [15], and for studying classical pure type systems [5].

Contents of the paper. In Section 2 we introduce abstract type systems. These are used in Section 3 to present the notion of pure type system and domain-free pure type system. In Section 4 we develop some basic properties of domain-free pure type systems, and in Section 5 we relate pure type systems to domain-free pure type systems. In Section 6 we compare domain-free pure type systems with the type assignment systems of [1]. In Section 7 we discuss type checking issues. We conclude in Section 8.

[3] These systems were informally suggested in [10].

2 Abstract type systems

In this section we introduce the notion of an abstract type system, which provides a convenient framework for the presentation and comparison of different notions of type system.

Definition 1. An *abstract type system* (ATS) is a triple $(V, T, \vdash)$ where

1. V is a set of *variables*;
2. T is a set of *terms* with $V \subseteq T$;
3. $C = (V \times T)^*$ is the set of *contexts*.[4] The empty context is denoted by $\langle\rangle$; The domain of a context Γ is $\mathrm{dom}(\Gamma) = \{x \mid \exists t \text{ s.t. } x : t \in \Gamma\}$.
4. $J = C \times T \times T$ is the set of *judgements*;
5. $\vdash \subseteq J$ is the *derivability* relation;

satisfying the following closure property

$$((\Gamma, x : B, \Gamma'), M, A) \in \vdash \quad \Rightarrow \quad ((\Gamma, x : B), x, B) \in \vdash$$

In the sequel, we write $\Gamma \vdash M : A$ for $(\Gamma, M, A) \in \vdash$ and $\vdash M : A$ for $(\langle\rangle, M, A) \in \vdash$.

The derivability relation may be used to define legal terms.

Definition 2. Let $(V, T, \vdash)$ be an ATS.

1. A judgement (Γ, M, A) is *legal* if $\Gamma \vdash M : A$.
2. A context Γ is *legal* if $\Gamma \vdash M : A$ for some M and A.
3. A term M is *legal* if $\Gamma \vdash M : A$ or $\Gamma \vdash A : M$ for some Γ and A.

Morphisms of ATSs provide an important tool to compare type systems. Here we let an ATS-morphism be a map between the underlying sets of terms which preserves derivability.

Definition 3. Let $(V, T, \vdash)$ and $(V', T', \vdash')$ be ATSs. A map $h : T \to T'$ is naturally lifted to maps $h^{\mathcal{C}} : C \to C'$ on contexts and $h^{\mathcal{J}} : J \to J'$ on judgements:

$$h^{\mathcal{C}}(\langle\rangle) = \langle\rangle$$
$$h^{\mathcal{C}}(\Gamma\ ,\ x{:}A) = h^{\mathcal{C}}(\Gamma)\ ,\ h(x){:}h(A)$$

$$h^{\mathcal{J}}(\Gamma, M, A) = (h^{\mathcal{C}}(\Gamma), h(M), h(A))$$

The map $h : T \to T'$ is an ATS-*morphism* if

1. for every $x \in V$, $h(x) \in V'$;
2. for every $j \in J$, $j \in \vdash$ implies $h^{\mathcal{J}}(j) \in \vdash'$.

[4] S^* denotes the set of finite lists over S.

The map $h : T \to T'$ is an ATS-*reflection* if moreover, for every $j' \in \vdash'$, there exists $j \in \vdash$ s.t. $h^{\mathcal{J}}(j) = j'$.

Examples of reflections, apart from the ones considered in this paper, can be found in [1, 3, 8]. Reflections are closed under composition but need not be injective and might not have an inverse. The following observation will be useful in Section 6.

Lemma 4. *Let* $(V_1, T_1, \vdash_1)$, $(V_2, T_2, \vdash_2)$ *and* $(V_3, T_3, \vdash_3)$ *be* ATS*s. Moreover let* $h : T_1 \to T_2$, $h' : T_2 \to T_3$ *and* $h'' : T_1 \to T_3$ *be* ATS*s morphisms s.t.* $h' \circ h(M) = h''(M)$ *for every legal* M. *If* h'' *is an* ATS-*reflection, then* h' *is an* ATS-*reflection.*

3 Pure type systems and domain-free pure type systems

In this section we present pure type systems and domain-free pure type systems. This approach is inspired by [14].

Definition 5.

1. A *specification* is a triple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ where
   (a) $\mathcal{S}$ is a set of *sorts*;
   (b) $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ is a set of *axioms*;
   (c) $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ is a set of *rules*.
2. A specification $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ is *functional* if for every $s_1, s_2, s_2', s_3, s_3' \in \mathcal{S}$,
   (a) $(s_1, s_2) \in \mathcal{A}$, $(s_1, s_2') \in \mathcal{A} \quad \Rightarrow \quad s_2 \equiv s_2'$
   (b) $(s_1, s_2, s_3) \in \mathcal{R}$, $(s_1, s_2, s_3') \in \mathcal{R} \quad \Rightarrow \quad s_3 \equiv s_3'$
3. $s \in \mathcal{S}$ is a *top-sort* if there is no $s' \in \mathcal{S}$ s.t. $(s, s') \in \mathcal{A}$. The set of top-sorts is denoted by $\mathcal{S}^{\mathsf{T}}$.

In the rest of this section, we let $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ denote a fixed specification and V denote a fixed countably infinite set of variables.

Definition 6.

1. The set $\mathcal{T}$ of PTS-*pseudo-terms* is given by the abstract syntax:
$$\mathcal{T} = V \mid \mathcal{S} \mid \mathcal{T}\mathcal{T} \mid \lambda V : \mathcal{T}.\mathcal{T} \mid \Pi V : \mathcal{T}.\mathcal{T}$$
2. β-*reduction* $\to_\beta$ is defined as the compatible closure of the contraction
$$(\lambda x{:}A.M)\ N \to_\beta M[x := N]$$
   where $\bullet[\bullet := \bullet]$ is the standard substitution operator.
3. β-*equality* $=_\beta$ is the reflexive, transitive, symmetric closure of $\to_\beta$.
4. The PTS *derivability* relation $\vdash$ is given by the rules of Table 1.

Every specification $\mathbf{S}$ induces an ATS $\lambda\mathbf{S}$ with $\mathcal{T}$ as the set of terms and $\vdash$ as the derivability relation. Such an ATS is called a *pure type system*, or a PTS.

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

Table 1. PURE TYPE SYSTEMS

Definition 7.

1. The set $\mathcal{L}$ of DFPTS-*pseudo-terms* is given by the abstract syntax:

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

2. $\underline{\beta}$*-reduction* $\to_{\underline{\beta}}$ is defined as the compatible closure of the contraction

$$(\lambda x.M)\ N \to_{\underline{\beta}} M\{x := N\}$$

   where $\bullet\{\bullet := \bullet\}$ is the obvious substitution operator.
3. β*-equality* $=_\beta$ is the reflexive, transitive, symmetric closure of $\to_{\underline{\beta}}$.
4. The DFPTS *derivability* relation $\vdash$ is given by the rules of Table 2.

Every specification $\mathbf{S}$ induces an ATS $\underline{\lambda}\mathbf{S}$ with $\mathcal{L}$ as the set of terms and $\vdash$ as the derivability relation. Such an ATS is called a *domain-free pure type system*, or a DFPTS. The two most significant DFPTSs that appear in the literature are Curry's version of the simply typed λ-calculus $\underline{\lambda}{\to}$ and Martin-Löf's Logical Framework $\underline{\lambda}P$. Further examples of DFPTSs are provided by the remaining specifications of the cube [2, 3], as defined below.

Definition 8. Let $\mathcal{S} = \{*, \Box\}$ and $\mathcal{A} = \{(* : \Box)\}$. The *cube*-specifications are

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

where (s_1, s_2) denotes (s_1, s_2, s_2). The *λ-cube* consists of the eight PTSs $\lambda\mathbf{S}$, where **S** is one of the cube-specifications. Similarly, the *$\underline{\lambda}$-cube* consists the eight DFPTSs $\underline{\lambda}\mathbf{S}$, where **S** is one of the cube-specifications.

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

Table 2. DOMAIN-FREE PURE TYPE SYSTEMS

Definition 9. Let (R, T) be $(\beta, \mathcal{T})$ or $(\underline{\beta}, \mathcal{L})$.

1. $\twoheadrightarrow_R$ is the reflexive transitive closure of $\to_R$;
2. $M \in \mathrm{NF}_R \Leftrightarrow$ there is no $N \in T$ s.t. $M \to_R N$;
3. $M \in \mathrm{SN}_R \Leftrightarrow$ there is no infinite sequence $M_0 \to_R M_1 \to_R M_2 \to_R \ldots$;
4. $M \in \mathrm{WN}_R \Leftrightarrow$ there is $N \in \mathrm{NF}_R$ s.t. $M \twoheadrightarrow_R N$.

Elements of $\mathrm{NF}_R, \mathrm{SN}_R, \mathrm{WN}_R$ are *R-normal forms*, *R-strongly normalizing*, and *R-weakly normalizing*, respectively.

Definition 10. A specification **S** has *normalizing (resp. strongly normalizing)* PTS-*types* if $M \in \mathrm{WN}_\beta$ (resp. $M \in \mathrm{SN}_\beta$) for every legal judgement $\Gamma \vdash M : s$ with $s \in \mathcal{S}$.

4 Properties of domain-free pure type systems

In this section, we state some basic facts about DFPTSs. We follow the structure of [3, Section 5.2]. The proofs are simply inductions, similar to those for the corresponding results for PTSs and are therefore omitted for brevity. Throughout the section, **S** denotes a fixed specification $(\mathcal{S}, \mathcal{A}, \mathcal{R})$.

Proposition 11 (Church-Rosser). *$\to_\beta$ is confluent on $\mathcal{L}$.*

Proof. By the technique of Tait and Martin-Löf. □

Lemma 12 (Substitution). *Assume $\Gamma, x : A, \Delta \vdash B : C$ and $\Gamma \vdash a : A$. Then also $\Gamma, \Delta\{x := a\} \vdash A\{x := a\} : B\{x := a\}$.*

Lemma 13 (Thinning). *If $\Gamma \vdash A : B$ and $\Delta \supseteq \Gamma$ is legal then $\Delta \vdash A : B$.*

Lemma 14 (Generation).

1. $\Gamma \vdash s : C \Rightarrow \exists (s, s') \in \mathcal{A}\ .C =_\beta s'$;
2. $\Gamma \vdash x : C \Rightarrow \exists s \in \mathcal{S}, D \in \mathcal{L}.\ C =_\beta D, x : D \in \Gamma, \Gamma \vdash D : s$;
3. $\Gamma \vdash \lambda x.b : C \Rightarrow \exists s \in \mathcal{S}, A, B \in \mathcal{L}.\ C =_\beta \Pi x{:}A.B, \Gamma, x : A \vdash b : B, \Gamma \vdash \Pi x{:}A.B : s$;
4. $\Gamma \vdash \Pi x{:}A.\ B : C \Rightarrow \exists (s_1, s_2, s_3) \in \mathcal{R}.\ C =_\beta s_3, \Gamma \vdash A : s_1, \Gamma, x : A \vdash B : s_2$;
5. $\Gamma \vdash F\,a : C \Rightarrow \exists x \in V, A, B \in \mathcal{L}.\ C =_\beta B\{x := a\}, \Gamma \vdash F : \Pi x{:}A.B, \Gamma \vdash a : A$.

Lemma 15 (Correctness of types). *If $\Gamma \vdash A : B$ then either $B \in \mathcal{S}^\top$ or $\exists s \in \mathcal{S}.\ \Gamma \vdash B : s$.*

Theorem 16 (Subject Reduction). *If $\Gamma \vdash A : B$ and $A \to_\beta A'$ then $\Gamma \vdash A' : B$.*

One important difference between PTSs and DFPTSs is that even in the simply typed case, the domain-free system does not satisfy Uniqueness of Types.

Lemma 17 (Failure of Uniqueness of Types). *In $\underline{\lambda}\!\to$, there exists a term M and a context Γ s.t. $\Gamma \vdash M : C$ and $\Gamma \vdash M : C'$ with $C \neq_\beta C'$.*

Proof. Take $\Gamma \equiv A : *, B : *$ and $M \equiv \lambda x.x$. Then $\Gamma \vdash M : A \to A$ and $\Gamma \vdash M : B \to B$. □

The failure of Uniqueness of Types is not catastrophic per se but is often accompanied by the loss of Decidability of Type Checking—see Section 7. Interestingly, the Classification Property still holds under some mild condition.

Definition 18. **S** is *classifiable* if it is functional and for all $s_1, s_3, s, s' \in \mathcal{S}$,

1. $\Gamma \vdash A : s \quad \wedge \quad \Gamma \vdash A' : s' \quad \wedge \quad A =_{\underline{\beta}} A' \quad \Rightarrow \quad s \equiv s'$
2. $(s, s_1) \in \mathcal{A} \quad \wedge \quad (s', s_1) \in \mathcal{A} \quad \Rightarrow \quad s \equiv s'$
3. $(s_1, s, s_3) \in \mathcal{R} \quad \wedge \quad (s_1, s', s_3) \in \mathcal{R} \quad \Rightarrow \quad s \equiv s'$

For example, the cube-specifications are classifiable.

The Classification Lemma is proved for a variant of DFPTSs with sorted variables. See [8] for a variant of PTSs based on sorted variables.

Proposition 19. *Assume that* **S** *is classifiable. For every sorts* $s \not\equiv s'$,

$$\mathsf{Term}^s \cap \mathsf{Term}^{s'} = \emptyset$$
$$\mathsf{Type}^s \cap \mathsf{Type}^{s'} = \emptyset$$

where

$$\mathsf{Type}^s = \{M \in \mathcal{L} \mid \Gamma \vdash M : s \textit{ for some context } \Gamma\}$$
$$\mathsf{Term}^s = \{M \in \mathcal{L} \mid \Gamma \vdash M : A \textit{ for some context } \Gamma \textit{ and } A \in \mathsf{Type}^s\}$$

5 Reflection and applications

In this section we study the relation between PTSs and DFPTSs. Throughout the section, **S** denotes a fixed specification.

Every PTS-pseudo-term induces a DFPTS-pseudo-term by erasing the domains of abstractions. This erasing function is used by Geuvers [8] to study PTSs with $\beta\eta$-conversion.

Definition 20. The *erasure* map $|.| : \mathcal{T} \to \mathcal{L}$ is defined as follows:

$$|x| = x$$
$$|s| = s$$
$$|t\ u| = |t|\ |u|$$
$$|\lambda x : A.t| = \lambda x.|t|$$
$$|\Pi x : A.B| = \Pi x : |A|.|B|$$

Erasure preserves reduction, equality and typing:

Proposition 21.

1. *If* $M \to_\beta N$ *then* $|M| \twoheadrightarrow_{\underline{\beta}} |N|$;
2. *If* $M =_\beta N$ *then* $|M| =_{\underline{\beta}} |N|$;
3. *If* $\Gamma \vdash M : A$ *then* $|\Gamma| \vdash |M| : |A|$.

Proof. First prove by induction on M that

$$|M[x := N]| \equiv |M|\{x := |N|\} \qquad (*)$$

Then prove (1) using $(*)$ by induction on the structure of M, (2) by induction on the derivation of $M =_\beta N$ and (3) by induction on the derivation of $\Gamma \vdash M : A$, using $(*)$ and 2. □

Corollary 22. $|.|$ *is an* ATS-*morphism from* $\lambda\mathbf{S}$ *to* $\underline{\lambda}\mathbf{S}$.

The main result of this section is that, under suitable conditions, $|.|$ is an ATS-reflection. This generalizes [3, Proposition 3.2.15] where a similar result is proved for simply typed λ-calculus—see also Section 6. Before proving themain result, we start with three preliminary lemmas.

The first lemma is about the relation between $\to_\beta$ and $\to_{\underline{\beta}}$.

Lemma 23 [8].

1. *If* $|A| \to_{\underline{\beta}} F$ *then there is* B *such that* $A \to_\beta B$ *and* $|B| \equiv F$;
2. *If* $|A| =_{\underline{\beta}} s$ *then* $A =_\beta s$.

The second lemma establishes the fundamental property of erasure.

Lemma 24. *Assume* $M, M' \in \mathrm{NF}_\beta$, $|M| \equiv |M'|$, $\Gamma \vdash M : A$, *and* $\Gamma \vdash M' : A'$.

1. *If* $A \equiv A'$ *then* $M \equiv M'$.
2. *If* $A \equiv s$, $A' \equiv s'$ *then* $M \equiv M'$.

The third lemma provides the necssary machinery to prove the main result.

Lemma 25. *Let* $\mathbf{S}$ *be a specification with normalizing* PTS-*types.*

1. *If* $\Gamma \vdash M : s$, $\Gamma \vdash N : s'$, $|M| =_{\underline{\beta}} |N|$, *then* $M =_\beta N$.
2. *If* Γ_1, Γ_2 *are legal and* $|\Gamma_1| =_{\underline{\beta}} |\Gamma_2|$ *then* $\Gamma_1 =_\beta \Gamma_2$.
3. *If* $\Gamma \vdash P : M$, $\Gamma \vdash N : s'$, $|M| =_{\underline{\beta}} |N|$, *then* $M =_\beta N$.

Theorem 26 (Main result). *Let* $\mathbf{S}$ *be a functional specification with normalizing* PTS-*types. Then* $|.|$ *is an* ATS-*reflection.*

Proof. By induction on the structure of derivations, using the above lemmas. □

We conclude this section with some applications of Theorem 26. These applications are used in [4, 5, 15]. First we examine how normalization is reflected.

Definition 27. Let (ϕ, T) be $(\lambda, \mathcal{T})$ or $(\underline{\lambda}, \mathcal{L})$. Assume $X \subseteq T$. We write $\phi\mathbf{S} \models X$ if every legal $\phi\mathbf{S}$-term t belongs to X.

We have:

Proposition 28. *Let* $\mathbf{S}$ *be a functional specification.*

1. $\lambda\mathbf{S} \models \mathrm{SN}_\beta$ *implies* $\underline{\lambda}\mathbf{S} \models \mathrm{SN}_{\underline{\beta}}$.
2. $\lambda\mathbf{S} \models \mathrm{WN}_\beta$ *implies* $\underline{\lambda}\mathbf{S} \models \mathrm{WN}_{\underline{\beta}}$.
3. *If* $\mathbf{S}$ *has strongly normalizing* PTS*-types, then* $\underline{\lambda}\mathbf{S} \models \mathrm{SN}_{\underline{\beta}}$ *implies* $\lambda\mathbf{S} \models \mathrm{SN}_\beta$.
4. *If* $\mathbf{S}$ *has normalizing* PTS*-types, then* $\underline{\lambda}\mathbf{S} \models \mathrm{WN}_{\underline{\beta}}$ *implies* $\lambda\mathbf{S} \models \mathrm{WN}_\beta$.

Proof. (1): By Thm. 26 and Lem. 23. (2): By Thm. 26, Prop. 21(1), and by noting that the erasure of a β-normal form is a $\underline{\beta}$-normal form. (3) and (4): By Prop. 21(3) and some elementary reasoning on reductions. □

This result is useful in work on the Barendregt-Geuvers-Klop conjecture which states that for every specification $\mathbf{S}$, $\lambda\mathbf{S} \models \mathrm{WN}_\beta$ implies $\lambda\mathbf{S} \models \mathrm{SN}_\beta$—see [15, 4]. Moreover, it implies strong normalisation for the $\underline{\lambda}$-cube:

Proposition 29. $\underline{\lambda}\mathbf{S} \models \mathrm{SN}_{\underline{\beta}}$ *for every cube-specification* $\mathbf{S}$.

Proof. By Proposition 28(1) and strong normalization of the λ-cube. □

As an application of Theorem 26, we can also infer consistency of a DFPTS from the corresponding PTS:

Proposition 30. *For every cube-specification, there is no* $M \in \mathcal{L}$ *s.t.* $x : * \vdash M : x$.

Proof. By Theorem 26 and consistency of the λ-cube. □

6 Comparison with type assignment systems

There are two ways to perceive Curry's version of simply typed λ-calculus:

1. terms in Curry's system are those of the untyped λ-calculus;
2. terms in Curry's system are those of Church's system with domain-free abstractions.

View (2) leads to the notion of domain-free pure type system studied in this paper whereas view (1) leads to the notion of *type assignment system*, or TAS. In recent work [1], van Bakel, Liquori, Ronchi della Roncha and Urzyczyn define for each cube-specification $\mathbf{S}$ a TAS $\overline{\lambda}\mathbf{S}$. These systems, the $\overline{\lambda}$*-cube*, include simple types $\overline{\lambda}{\to}$ introduced by Curry [7], second-order types $\overline{\lambda}2$ introduced by Leivant [12] and higher-order types $\overline{\lambda}\omega$ introduced by Giannini and Ronchi della Rocca [11].

An important aspect of the $\overline{\lambda}$-cube is the separation of the set $\mathcal{U}$ of pseudo-terms into three different syntactic categories: terms, constructors and kinds. Terms use domain-free abstractions whereas constructors use domain-specified abstractions. As a consequence, the erasure map[5] E is an ATS-reflection from $\lambda\mathbf{S}$ to $\overline{\lambda}\mathbf{S}$ only for the specifications which do not combine polymorphism and

[5] E is defined as a map on legal PTS-terms but can of course be extended to a map on PTS-pseudo-terms, e.g. by taking $E(M) = *$ if M is not legal.

dependent types, i.e. $\rightarrow$, 2, ω, P, $\underline{\omega}$ and $P\underline{\omega}$. For the remaining specifications $P2$ and $P\omega$, E is simply an ATS-morphism.

Note that $\underline{\lambda}\mathbf{S}$ and $\overline{\lambda}\mathbf{S}$ are identical for $\mathbf{S} = \rightarrow$ but diverge for the other cube-specifications. Indeed, consider the $\lambda 2$-term $\lambda\alpha{:}*.\lambda x{:}\alpha.x$ of type $\forall\alpha{:}*.\alpha \rightarrow \alpha$. The corresponding term in $\overline{\lambda}2$ is $E(M) \equiv \lambda x.x$ whereas the corresponding domain-free term in $\underline{\lambda}2$ is $|M| \equiv \lambda\alpha.\lambda x.x$.

We now turn to the relation between TASs and DFPTSs. Define the erasure map $E' : \mathcal{U} \rightarrow \mathcal{L}$ which removes the domains of constructor abstractions.

Proposition 31. *If* $\mathbf{S}$ *is a cube-specification without polymorphism (i.e.* $\mathbf{S}$ *is* $\rightarrow$, P, $\underline{\omega}$ *or* $P\underline{\omega}$*), then* E' *is an* ATS*-reflection from* $\overline{\lambda}\mathbf{S}$ *to* $\underline{\lambda}\mathbf{S}$.

Proof. Prove by induction on the structure of derivations that E' is an ATS-morphism and that $|M| = E'(E\ M)$ for every legal PTS-term M. Conclude by Theorem 26 and Lemma 4. □

Note that E' is not an ATS-morphism from $\overline{\lambda}2$ to $\underline{\lambda}2$: consider the term M given above —a similar remark applies to ω, $P2$ and $P\omega$. However, one may define an erasure map $F : \mathcal{L} \rightarrow \mathcal{U}$ which removes type abstractions and type applications.[6]

Proposition 32. F *is an* ATS*-reflection.*

Proof. Prove by induction on the structure of derivations that F is an ATS-morphism and that $E(M) = F(|M|)$ for every legal PTS-term M. Conclude from the fact that E is an ATS-reflection from $\lambda 2$ to $\overline{\lambda}2$ and Lemma 4. □

7 Type checking

The *type checking problem* (TC) for a specification $\mathbf{S}$ consists in deciding whether a given DFPTS-judgement (Γ, M, A) is legal. In this section we briefly discuss some results related to type checking.

Proposition 33. *Given an arbitrary specification* $\mathbf{S}$*, it is decidable whether a* DFPTS*-judgement in* $\underline{\beta}$*-normal is derivable.*

People working on the ALF system claim that this limited form of decidability is sufficient in practice because of the presence of definitions—see e.g. [13].

Proposition 34. *TC is decidable for* $\underline{\lambda} \rightarrow$ *but undecidable for* $\underline{\lambda}P$.

It would be interesting to know whether TC is decidable for $\underline{\lambda}2$ and $\underline{\lambda}\omega$.

[6] It requires that we work with sorted variables, as suggested at the end of Section 4.

8 Conclusion

We have introduced the notion of a domain-free pure type system, developed its basic properties and established its exact relationship with the notion of pure type system. Thus far,DFPTSs have proved useful in several applications [4, 5, 15]. Moreover, —variants of—DFPTSs are used in the implementation of proof-assistant systems based on type theory [13]. It is also possible that DFPTSs will play a role in the design of programming languages; for example, one may envisage a dependently typed extension of ML based on $\underline{\lambda}P$. Finally, we would like to point out that DFPTSs sometimes provide a more natural basis than PTSs for extended type systems. For example, the extension of pure type systems with classical operators [5] yields better results when the abstractions are domain-free.

References

1. S. van Bakel, L. Liquori, S. Ronchi della Roncha, and P. Urzyczyn. Comparing cubes. In A. Nerode and Y.N. Matiyasevich, editors, *Proceedings of LFCS'94*, volume 813 of *Lecture Notes in Computer Science*, pages 353–365. Springer-Verlag, 1994. An extended version is to appear in *Annals of Pure and Applied Logic*.
2. H. Barendregt. Introduction to Generalised Type Systems. *J. Functional Programming*, 1(2):125–154, April 1991.
3. H. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, pages 117–309. Oxford Science Publications, 1992. Volume 2.
4. G. Barthe, J. Hatcliff, and M.H. Sørensen. CPS-translation and applications: the cube and beyond. In O. Danvy, editor, *Proc. of the 2nd ACM SIGPLAN Workshop on Continuations*, number NS-96-13 in BRICS Notes, pages 4:1–31, 1996.
5. G. Barthe, J. Hatcliff, and M.H. Sørensen. Classical pure type systems. In *Proceedings of MFPS'97*, Electronic Notes in Theoretical Computer Science, 1997.
6. A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
7. H. Curry. Functionality in combinatory logic. *Proceedings of the National Academy of Science USA*, 20:584–590, 1934.
8. H. Geuvers. *Logics and type systems*. PhD thesis, University of Nijmegen, 1993.
9. H. Geuvers and M.-J. Nederhof. A modular proof of strong normalisation for the Calculus of Constructions. *Journal of Functional Programming*, 1:155–189, 1991.
10. H. Geuvers and B. Werner. On the Church-Rosser property for expressive type systems and its consequence for their metatheoretic study. In *Proceedings of LICS'94*, pages 320–329. IEEE Computer Society Press, 1994.
11. P. Giannini and S. Ronchi Della Rocca. Characterization of typings in polymorphic type discipline. In *Proceedings of LICS'88*, pages 61–70. IEEE Computer Society Press, 1988.
12. D. Leivant. Polymorphic type inference. In *Proceedings of POPL'83*, pages 88–98. ACM Press, 1983.
13. L. Magnusson. *The implementation of ALF: a proof editor based on Martin-Löf's monomorphic type theory with explicit substitution*. PhD thesis, Department of Computer Science, Chalmers University, 1994.

14. P. Severi. *Normalisation in lambda calculus and its relation to type inference.* PhD thesis, Department of Computer Science, Technical University of Eindhoven, 1996.
15. M.H. Sørensen. Strong normalization from weak normalization in typed λ-calculi. *Information and Computation*, 133(1):35–71, 1997.