J. Functional Programming **10** (5): 417–452, September 2000. Printed in the United Kingdom

Domain-free pure type systems

GILLES BARTHE

INRIA Sophia-Antipolis, 2004 Route des Lucioles, BP 93,
06902 Sophia-Antipolis Cedex, France
(*e-mail:* Gilles.Barthe@sophia.inria.fr)

MORTEN HEINE SØRENSEN

Department of Computer Science, University of Copenhagen (DIKU),
Universitetsparken 1, DK-2100 Copenhagen Ø, Denmark
(*e-mail:* rambo@diku.dk)

Abstract

Pure type systems make use of domain-full λ-abstractions $\lambda x : D \,.\, M$. We present a variant of pure type systems, which we call *domain-free pure type systems,* with domain-free λ-abstractions $\lambda x \,.\, M$. Domain-free pure type systems have a number of advantages over both pure type systems and so-called type assignment systems (they also have some disadvantages), and have been used in theoretical developments as well as in implementations of proof-assistants. We study the basic properties of domain-free pure type systems, establish their formal relationship with pure type systems and type assignment systems, and give a number of applications of these correspondences.

Capsule Review

The distinction between Curry style and Church style of presentation of type systems, introduced by Barendregt, is well known. The authors point out an interesting third way of presenting type systems, which is intermediate between Curry and Church, and which is called here "domain-free" type systems. This presentation is actually the one used in Martin-Löf's logical framework, and it presents various interesting aspects, both theoretical and practical, that are carefully described in this paper.

1 Introduction

Typed versions of the λ-calculus were introduced independently by Church (1940) and Curry (1934). More precisely, Curry (1934) introduced types into the theory of *combinators,* and Curry and Feys (1958) modified the system in a natural way to λ-calculus. In Church's system abstractions have *domains*, i.e. are of the form $\lambda x : D \,.\, t$, whereas in Curry's system abstractions have no domain, i.e. are of the form $\lambda x \,.\, t$. Thus, in Church's system one writes

$$\lambda x : \alpha \,.\, x : \alpha \to \alpha$$

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

whereas in Curry's system one writes

$$\lambda x\,.\,x : \alpha \to \alpha$$

There are two ways to perceive the terms that have types in Curry's system:

1. As a subset of the untyped λ-terms;
2. As those that have types in Church's system, but with domains omitted.

For the pair of systems introduced by Church and Curry the two views are equivalent, but for some other systems the two views diverge. Consider, for instance, the second-order typed λ-calculus *à la* Church, which was invented independently by Girard (1970) and Reynolds (1974). An example term and type is

$$\Lambda\alpha : *\,.\,\lambda x : \alpha\,.\,x \;:\; \forall\alpha : *\,.\,\alpha \to \alpha$$

In Leivant's (1983) formulation of second-order typed λ-calculus *à la* Curry, the similar term and type is

$$\lambda x\,.\,x \;:\; \forall\alpha : *\,.\,\alpha \to \alpha$$

This clearly fits view (1). The similar term and type in view (2) is

$$\Lambda\alpha\,.\,\lambda x\,.\,x \;:\; \forall\alpha : *\,.\,\alpha \to \alpha$$

Thus, we may distinguish *three* approaches to type systems: Church's approach, which we call the *domain-full* approach, and the variants (1) and (2) of Curry's approach. View (1), traditionally known as the *type assignment* approach, has been extensively studied in the literature – see (Barendregt, 1992) and (van Bakel *et al.*, 1994). In contrast, view (2), which we call the *domain-free* approach, has received little attention.[1]

This is surprising, since:

1. Domain-free type systems provide a framework for the description of logics.[2]
2. Domain-free type systems are in use in Martin-Löf's Logical Frameworks and in some versions of Martin-Löf's Intuitionistic Type Theory, e.g. see Tasistro (1997).
3. A number of proof-development systems, most notably Alf (Magnusson, 1994), rely on domain-free type systems, while others, most notably Elf (Pfenning, 1994), allow for domain-free λ-abstractions.

[1] Incidentally, the difference between (1) and (2) should not be confused with another difference between Curry's and Church's original systems. Curry considered as a starting point the *whole* set of untyped λ-terms, thus including, for example, $\lambda x\,.\,x\,x$, and defined the *legal* terms as those having a type in his system, including, for instance, $\lambda x\,.\,x$ (since it has type $\alpha \to \alpha$) but excluding, for example, $\lambda x\,.\,x\,x$ (since it has no type). In contrast, Church built the type system directly into the term formation rules. For Church there were no other terms than the legal ones, e.g. $\lambda x : \alpha\,.\,x$. Objects like $\lambda x : \alpha\,.\,x\,x$ did not exist anywhere in his approach. The distinction between (1) and (2) does not concern the question whether the legal terms are constructed directly, or selected from a broader set of terms, but rather whether there is a single kind of abstraction $\lambda x\,.\,t$ (as in untyped λ-calculus) or several kinds of abstraction like $\lambda x\,.\,t$ and $\Lambda\alpha\,.\,t$ (as in the corresponding Church system). All the systems presented in this paper – with or without domains – start out from some set of terms and use a typing relation to select among these the legal ones.

[2] A major difference between domain-full type systems and domain-free ones is that the latter may not have decidable type-checking. However domain-free type systems may describe logics in the same way as domain-full type systems do.

4. Domain-free type systems offer some advantages over both domain-full type systems and type assignment systems.

Advantages over domain-full type systems include the following:

- *Extensibility*. Domain-free type systems are sometimes easier to extend than domain-full type systems. For example, extending Girard's (1972) domain-full higher-order typed λ-calculus with a catch/throw mechanism leads to non-left linear rules, which are notoriously difficult to handle. An example of such a rule is given by

$$\texttt{catch } \alpha : A \texttt{ in } (\texttt{throw } M \texttt{ to } \alpha : A) \quad \to \quad M \qquad \text{if } \alpha \notin \mathrm{FV}(M)$$

  The similar domain-free rule

$$\texttt{catch } \alpha \texttt{ in } (\texttt{throw } M \texttt{ to } \alpha) \quad \to \quad M \qquad \text{if } \alpha \notin \mathrm{FV}(M)$$

  is much simpler.
- *Efficiency*. Domain-free reduction is sometimes more efficient than domain-full reduction. For example, consider for the extension mentioned above the rule

$$(\texttt{catch } \alpha : (\Pi x{:}\, C.\, D) \texttt{ in } M)\, N \quad \to \quad \texttt{catch } \beta : D\{x := N\} \texttt{ in } M'\, N$$

  where M' arises from M by replacing all occurrences of $\texttt{throw } K \texttt{ to } \alpha : A$ by $\texttt{throw } K\, N \texttt{ to } \beta : D\{x := N\}$ (ignoring the question as to whether we should insist that $A \equiv \Pi x{:}\, C.\, D$). In order to reduce $(\texttt{catch } \alpha : B \texttt{ in } M)\, N$ with $B \equiv (\lambda y : *.\, y)\, \Pi z{:}\, *.\, z$, we need to reduce B first. The domain-free rule

$$(\texttt{catch } \alpha \texttt{ in } M)\, N \quad \to \quad \texttt{catch } \beta \texttt{ in } M'\, N$$

  where M' arises from M by replacing all occurrences of $\texttt{throw } K \texttt{ to } \alpha$ by $\texttt{throw } K\, N \texttt{ to } \beta$ does not require this (and sidesteps the complications regarding the form of A).
- *Simplicity*. Domain-free type systems are sometimes easier to study than domain-full type systems. For example, continuation-passing style (CPS) translations are easier to define for domain-free type systems (see Barthe *et al.*, 1999). This observation is especially relevant as continuation-passing style translations are, apart from their theoretical interest, a fundamental tool in compilation (Appel, 1992). In addition, strong normalization is often easier to prove for domain-free type systems than for their (domain-full) counterpart.[3]

Advantages of domain-free type systems over type assignment systems include the following:

- *Uniformity*. Domain-free type systems are more uniform than type assignment systems. For example, domain-free type systems use a single rule for abstraction, whereas some type assignment systems, e.g. the higher-order type

[3] It is, for example, straightforward to prove strong normalization of a domain-free classical Calculus of Constructions, i.e. a Calculus of Constructions enhanced with a control/double negation operator. Proving a similar result for a domain-full Calculus of Constructions is immensely more complicated. See Barthe *et al.* (1997).

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

assignment of Giannini and Ronchi Della Rocca (1988), use λ-abstractions both with and without domains and two rules for abstraction.

- *Clarity*. Domain-free type systems are easier to formulate for complex type disciplines. For example, providing a type assignment system for the Calculus of Constructions is a non-trivial task (see van Bakel *et al*., 1994). In contrast, providing a domain-free type system for the Calculus of Constructions is easy.

We do not claim that domain-free type systems are better than domain-full systems or type assignment systems. Indeed, they have their own drawbacks; for instance, a domain-free type system may have undecidable type checking problem even when type-checking for the domain-full counterpart is decidable. Nevertheless, the above discussion suggests that domain-free type systems are interesting in their own right.

The purpose of this paper is to present a framework in which to study domain-free type systems. The framework, which we call *domain-free pure type systems*, is inspired by *pure type systems* (Barendregt, 1992; Berardi, 1990; Geuvers, 1993; Terlouw, 1989), which give an abstract, unifying view of type systems with domain-full abstractions. Just as pure type systems contain as a special case the *λ-cube,* the domain-free pure type systems contain as a special case the *domain-free λ-cube.*

Our contribution is three-fold. First, we introduce domain-free pure type systems (section 2) and develop their basic theory (section 3); it turns out that they satisfy almost the same properties as pure type systems. Secondly, we study the relationship between pure type systems and domain-free pure type systems (section 4); the two formalisms are equivalent – in a precise sense – for many of the systems appearing in the literature. Thirdly, we demonstrate that this equivalence is very useful in that it allows to transfer several results from one formalism to another (section 5).

There is no general formalism available which is to the type assignment approach what pure type systems and domain-free pure type systems are to the domain-full and domain-free approach, respectively. However, there is a *type assignment λ-cube* (van Bakel *et al*., 1994) which is to the type assignment approach what the λ-cube and the domain-free λ-cube are to the domain-full and domain-free approach, respectively (see Figure 1). For the sake of completeness, we also study the relationship between the type assignment cube and the two other cubes (section 6).

The paper is an extended and updated version of Barthe and Sørensen (1997).

2 Domain-free pure type systems

Domain-free pure type systems are generated from specifications, just like pure type systems are (see Barendregt, 1992). Specifications are triples expressing certain abstract dependencies.

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

1. $\mathscr{S}$ is a set of *sorts*;
2. $\mathscr{A} \subseteq \mathscr{S} \times \mathscr{S}$ is a set of *axioms*;
3. $\mathscr{R} \subseteq \mathscr{S} \times \mathscr{S} \times \mathscr{S}$ is a set of *rules.*

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

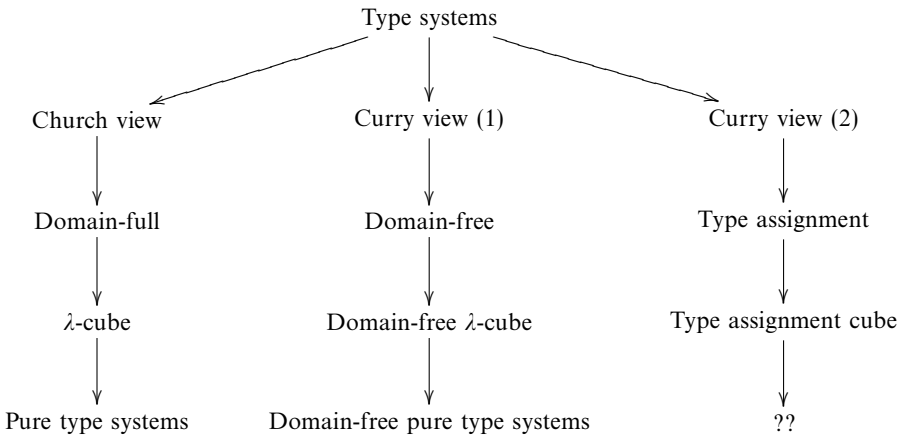

Fig. 1. Approaches to type systems.

As usual, a rule of form (s_1, s_2, s_2) is also written (s_1, s_2). A sort $s \in \mathscr{S}$ is a *top-sort* if $(s, s') \notin \mathscr{A}$ for all $s' \in \mathscr{S}$. The set of top-sorts is denoted by $\mathscr{S}_\top$.

In the rest of the paper V denotes a fixed, countably infinite set of variables.

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

1. The set $\underline{\mathscr{E}}$ of *(domain-free) expressions* (over $\mathbf{S}$) is given by the abstract syntax:

$$\underline{\mathscr{E}} = V \mid \mathscr{S} \mid \underline{\mathscr{E}}\,\underline{\mathscr{E}} \mid \lambda V.\underline{\mathscr{E}} \mid \Pi V : \underline{\mathscr{E}}.\underline{\mathscr{E}}$$

   We use $a, b, c, d, A, B, C, D, K, L, M, N$, etc. to denote elements of $\underline{\mathscr{E}}$; x, y, z, etc. to denote elements of V; and s, s', etc. to denote elements of $\mathscr{S}$. We assume the reader is familiar with the notions of free and bound variables and related conventions; $\mathrm{FV}(M)$ denotes the set of variables occurring free in M, and $\equiv$ denotes syntactic equality (see Barendregt, 1992).
2. A *(domain-free) context* is a finite sequence of form $x_1 : A_1, \ldots, x_n : A_n$; the empty sequence is written $\langle\rangle$. The set of all contexts is called $\underline{\mathscr{G}}$. We write $\mathrm{dom}(x_1 : A_1, \ldots, x_n : A_n) = \{x_1, \ldots, x_m\}$ and use Γ, Δ, etc. to denote elements of $\underline{\mathscr{G}}$. If Γ is $x_1 : A_1, \ldots, x_n : A_n$ we also write $x_i : A_i \in \Gamma$ for each $i \in \{1, \ldots, n\}$.
3. *$\underline{\beta}$-reduction* $\rightarrow_{\underline{\beta}}$ on $\underline{\mathscr{E}}$ is defined as the compatible closure of the contraction

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

   where $\bullet\{\bullet := \bullet\}$ is the obvious substitution operator. *Multi-step $\underline{\beta}$-reduction* $\twoheadrightarrow_{\underline{\beta}}$ and *$\underline{\beta}$-equality* $=_{\underline{\beta}}$ are the reflexive, transitive closure and reflexive, transitive, symmetric closure, respectively, of $\rightarrow_{\underline{\beta}}$.
   The relation $\rightarrow_{\underline{\beta}}$ is extended to $\underline{\mathscr{G}}$ by:

$$A \rightarrow_{\underline{\beta}} B \Rightarrow \Gamma, x : A, \Delta \rightarrow_{\underline{\beta}} \Gamma, x : B, \Delta$$

   The relations $\twoheadrightarrow_{\underline{\beta}}$ and $=_{\underline{\beta}}$ on $\underline{\mathscr{G}}$ are the obvious closures of $\rightarrow_{\underline{\beta}}$ on $\underline{\mathscr{G}}$.

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

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

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

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

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

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

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

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

Fig. 2. Domain-Free Pure Type Systems (DFPTS).

4. The *derivability* relation $\vdash$ is given by the rules of Figure 2. We occasionally write $\vdash_{\mathbf{S}}$ to explicate the dependence of the derivability relation on $\mathbf{S}$. If $\Gamma \vdash A : B$ then Γ, A, and B are *legal*. We write $\Gamma \vdash A : B : C$ if $\Gamma \vdash A : B$ and $\Gamma \vdash B : C$.
5. The tuple $\underline{\lambda}\mathbf{S} = (\underline{\mathscr{E}}, \underline{\mathscr{G}}, =_{\underline{\beta}}, \vdash)$ is the *Domain-Free Pure Type System (*DFPTS*) induced by* $\mathbf{S}$.

The most significant DFPTSs that appear in the literature (in variant form) are generated by specifications that belong to the Barendregt's cube of specifications. These DFPTSs roughly correspond to Curry's version of the simply typed λ-calculus (in our setting, this system is generated by the specification $\rightarrow$), Martin-Löf's Logical Frameworks (generated by the specification P of Logical Frameworks) and Martin-Löf's type theory with one universe (generated by the specification $P\underline{\omega}$). Other specifications that correspond to well-known domain-full type systems are 2 (System F or polymorphic λ-calculus), ω (System F^ω or higher-order polymorphic λ-calculus) and C (Calculus of Constructions).

Definition 3
Let $\mathscr{S} = \{*, \Box\}$ and $\mathscr{A} = \{(* : \Box)\}$. The *cube*-specifications are

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

We use the standard abbreviations $\omega = \underline{\omega}2$ and $C = P\omega = P\underline{\omega}2$.

The DFPTSs generated by the cube-specifications form the $\underline{\lambda}$*-cube* (see Figure 3), which we also call the *domain-free* λ-cube.

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

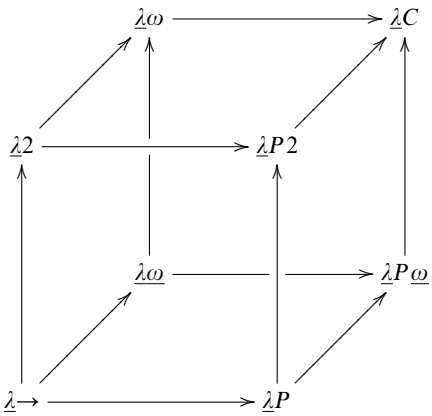

Fig. 3. The $\underline{\lambda}$-cube.

Remark 4
In the rest of this paper we often consider a specification $\mathbf{S}$ and speak about, for example, a sort s or a domain-free expression M. In such cases it must be understood that $s \in \mathscr{S}$ where $\mathbf{S} = (\mathscr{S}, \mathscr{A}, \mathscr{R})$ and $M \in \underline{\mathscr{E}}$ where $\underline{\lambda}\mathbf{S} = (\underline{\mathscr{E}}, \underline{\mathscr{G}}, =_{\underline{\beta}}, \vdash)$.

We close this section with some definitions required in the following sections.

Definition 5
Let $\mathbf{S}$ be a specification.

$$\begin{aligned} \underline{\mathsf{Type}}^s &= \{M \in \underline{\mathscr{E}} \mid \Gamma \vdash M : s \text{ for some } \Gamma\} \\ \underline{\mathsf{Term}}^s &= \{M \in \underline{\mathscr{E}} \mid \Gamma \vdash M : A : s \text{ for some } \Gamma \text{ and } A\} \end{aligned}$$

Also, $\underline{\mathsf{Type}} = \cup_{s \in \mathscr{S}} \underline{\mathsf{Type}}^s$ and $\underline{\mathsf{Term}} = \cup_{s \in \mathscr{S}} \underline{\mathsf{Term}}^s$.

Definition 6
Let $\mathbf{S}$ be a specification. A *$\underline{\beta}$-reduction path from* $M_0 \in \underline{\mathscr{E}}$ is a finite or infinite sequence

$$M_0 \to_{\underline{\beta}} M_1 \to_{\underline{\beta}} M_2 \to_{\underline{\beta}} \ldots$$

If the sequence is finite it *ends* in the last term M_n and has *length* n.

Definition 7
Let $\mathbf{S}$ be a specification and $M \in \underline{\mathscr{E}}$.

1. $M \in \mathrm{NF}_{\underline{\beta}}$ iff there is no $\underline{\beta}$-reduction path from M of length 1 or more.
2. $M \in \mathrm{WN}_{\underline{\beta}}$ iff there is a $\underline{\beta}$-reduction path from M ending in an $N \in \mathrm{NF}_{\underline{\beta}}$.
3. $M \in \mathrm{SN}_{\underline{\beta}}$ iff all $\underline{\beta}$-reduction paths from M are finite.

Elements of $\mathrm{NF}_{\underline{\beta}}$, $\mathrm{WN}_{\underline{\beta}}$, and $\mathrm{SN}_{\underline{\beta}}$ are called *$\underline{\beta}$-normal forms*, *$\underline{\beta}$-(weakly) normalizing*, and *$\underline{\beta}$-strongly normalizing*, respectively.

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

3 Properties of domain-free pure type systems

In this section, we study properties of DFPTSs. In the first subsection we develop the basic properties such as *correctness of types* and *subject reduction,* following the structure of Barendregt (1992, section 5.2); proof details are omitted when the proof proceeds by induction and is similar to the proof for the corresponding result for pure type systems. In the second subsection we present the classification lemma, which is useful in several applications, e.g. to define CPS-translations for the domain-free λ-cube (Barthe *et al.*, 1999). In the third subsection we consider type-checking issues. We show that although type-checking may be undecidable, even for systems of the $\underline{\lambda}$-cube, one can prove a weaker result which allows for DFPTSs to be used in practice.

3.1 Basic properties

Throughout this subsection, **S** denotes a fixed specification.

Lemma 8 (*Properties of Substitution*)

1. $A\{x := B\}\{y := C\} \equiv A\{y := C\}\{x := B\{y := C\}\}$, if $y \notin \mathrm{FV}(B)$;
2. $B =_{\underline{\beta}} C \Rightarrow A\{x := B\} =_{\underline{\beta}} A\{x := C\}$;
3. $A =_{\underline{\beta}} B \;\&\; C =_{\underline{\beta}} D \Rightarrow A\{x := C\} =_{\underline{\beta}} B\{x := D\}$.

Proof
(1)–(2): induction on A. (3): induction on $A =_{\underline{\beta}} B$, using (1)-(2). □

Proposition 9 (*Church–Rosser*)
The relation $\rightarrow_{\underline{\beta}}$ on $\underline{\mathscr{E}}$ is confluent.

Proof
By the technique of Tait and Martin-Löf (e.g. see Barendregt, 1992). □

Alternatively, note that $(\underline{\mathscr{E}}, \rightarrow_{\underline{\beta}})$ is an orthogonal rewriting system and invoke Klop *et al.* (1993).

Lemma 10 (*Free Variables*)
If $x_1 : A_1, \ldots, x_n : A_n \Vdash B : C$ then:

1. $x_1, \ldots, x_n$ are distinct;
2. $\mathrm{FV}(B) \cup \mathrm{FV}(C) \subseteq \{x_1, \ldots, x_n\}$;
3. $\mathrm{FV}(A_i) \subseteq \{x_1, \ldots, x_{i-1}\}$ for $1 \leqslant i \leqslant n$.

Proof
By induction on the derivation of $x_1 : A_1, \ldots, x_n : A_n \Vdash B : C$. □

Lemma 11 (*Start*)
If Γ is legal then:

1. $(s_1, s_2) \in \mathscr{A} \Rightarrow \Gamma \Vdash s_1 : s_2$;
2. $x : A \in \Gamma \Rightarrow \Gamma \Vdash x : A$.

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

Proof
Since Γ is legal $\Gamma \Vdash B : C$ for some B, C. Proceed by induction on the derivation of $\Gamma \Vdash B : C$. □

Lemma 12 (*Transitivity*)
Let Δ be legal. If $x_1 : A_1 \ldots, x_n : A_n \Vdash A : B$ and $\Delta \Vdash x_i : A_i$ for $i = 1, \ldots, n$ then $\Delta \Vdash A : B$.

Proof
By induction on the derivation of $x_1 : A_1, \ldots, x_n : A_n \Vdash A : B$ making use of the start lemma. □

Lemma 13 (*Substitution*)
If $\Gamma, x : A, \Delta \Vdash B : C$ and $\Gamma \Vdash a : A$, then[4] $\Gamma, \Delta\{x := a\} \Vdash A\{x := a\} : B\{x := a\}$.

Proof
By induction on the derivation of $\Gamma, x : A, \Delta \Vdash B : C$ using the free variables lemma and properties of substitution. □

Lemma 14 (*Thinning*)
If $\Gamma \Vdash A : B$, Δ is legal, and every $x : A$ in Γ is also in Δ, then $\Delta \Vdash A : B$.

Proof
This follows from the start lemma and the transitivity lemma. □

Lemma 15 (*Generation*)
Suppose that $\Gamma \Vdash M : C$.

1. $M \equiv s \Rightarrow \exists (s, s') \in \mathscr{A}.\ C =_{\underline{\beta}} s'$
2. $M \equiv x \Rightarrow \exists D \in \underline{\mathscr{E}}.\ C =_{\underline{\beta}} D\ \&\ x : D \in \Gamma$.
3. $M \equiv \lambda x.\, b \Rightarrow \exists s \in \mathscr{S}, A, B \in \underline{\mathscr{E}}.\ C =_{\underline{\beta}} \Pi x : A.\, B\ \&\ \Gamma, x : A \Vdash b : B\ \&\ \Gamma \Vdash \Pi x : A.\, B : s$.
4. $M \equiv \Pi x : A.\, B \Rightarrow \exists (s_1, s_2, s_3) \in \mathscr{R}.\ C =_{\underline{\beta}} s_3\ \&\ \Gamma \Vdash A : s_1\ \&\ \Gamma, x : A \Vdash B : s_2$.
5. $M \equiv F\, a \Rightarrow \exists x \in V, A, B \in \underline{\mathscr{E}}.\ C =_{\underline{\beta}} B\{x := a\}\ \&\ \Gamma \Vdash F : \Pi x : A.\, B\ \&\ \Gamma \Vdash a : A$.

Proof
By induction on the derivation of $\Gamma \Vdash M : C$. □

Lemma 16 (*Correctness of types*)
If $\Gamma \Vdash A : B$ then either $B \in \mathscr{S}$ or $\exists s \in \mathscr{S}.\ \Gamma \Vdash B : s$.

Proof
By induction on $\Gamma \Vdash A : B$, using the generation and substitution lemmas. □

Theorem 17 (*Subject Reduction*)
If $\Gamma \Vdash A : B$ and $A \to_{\underline{\beta}} A'$ then $\Gamma \Vdash A' : B$.

Proof
Prove by simultaneous induction on the derivation of $\Gamma \Vdash A : B$:

1. if $\Gamma \Vdash A : B$ and $A \to_{\underline{\beta}} A'$ then $\Gamma \Vdash A' : B$;
2. if $\Gamma \Vdash A : B$ and $\Gamma \to_{\underline{\beta}} \Gamma'$ then $\Gamma' \Vdash A : B$.

The proof uses the substitution lemma. □

[4] Substitution (and any other map) is extended from expressions to contexts in the usual way.

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

$$\text{(start-s)} \qquad \frac{\Gamma \Vdash A : s}{\Gamma, x{:}\, A \Vdash x : A} \qquad \text{if } x \in V^s \setminus \mathrm{dom}(\Gamma)$$

$$\text{(weakening-s)} \qquad \frac{\Gamma \Vdash A : B \quad \Gamma \Vdash C : s}{\Gamma, x{:}\, C \Vdash A : B} \qquad \text{if } x \in V^s \setminus \mathrm{dom}(\Gamma)$$

Fig. 4. Domain-Free Pure Type Systems with sorted variables.

We conclude this subsection by pointing out that we have not been able to prove the strengthening lemma, i.e.

$$\Gamma_1, x : A, \Gamma_2 \Vdash M : B \quad \& \quad x \notin \mathrm{FV}(\Gamma_2) \cup \mathrm{FV}(M) \cup \mathrm{FV}(B) \quad \Rightarrow \quad \Gamma_1, \Gamma_2 \Vdash M : B$$

Indeed the standard proof relies on the following implication, which does not holds for DFPTSs:

$$\Gamma_1, x : A, \Gamma_2 \Vdash M : B \quad \& \quad x \notin \mathrm{FV}(\Gamma_2) \cup \mathrm{FV}(M)$$
$$\Rightarrow \quad \exists B' \in \underline{\mathscr{E}}.\ \Gamma_1, \Gamma_2 \Vdash M : B \quad \& \quad B =_{\underline{\beta}} B'$$

3.2 The classification lemma

Traditional formulations of type theories distinguish between different syntactic categories: for instance, in the system F_ω (Girard, 1972) one distinguishes between *objects, constructors,* and *kinds.* Such a distinction is used in several practical and theoretical applications, but is not enforced in the syntax of DFPTSs. However it can be recovered *a posteriori* using the *classification lemma.*

As usual, the classification lemma will be proved for a variant of DFPTSs with sorted variables. In this variant, it is assumed that V is partitioned into $\bigcup_{s \in \mathscr{S}} V^s$, with each V^s being countably infinite, and that variables are manipulated according to the rules in Figure 4.

The classification lemma does not hold for all DFPTSs; therefore, we consider the following standard classes of specifications.

Definition 18
Let $\mathbf{S}$ be a specification.

1. $\mathbf{S}$ is *functional* if for every $s_1, s_2, s_2', s_3, s_3' \in \mathscr{S}$,
   (a) $(s_1, s_2) \in \mathscr{A}$ & $(s_1, s_2') \in \mathscr{A}$ $\Rightarrow$ $s_2 \equiv s_2'$
   (b) $(s_1, s_2, s_3) \in \mathscr{R}$ & $(s_1, s_2, s_3') \in \mathscr{R}$ $\Rightarrow$ $s_3 \equiv s_3'$
2. $\mathbf{S}$ is *injective* if $\mathbf{S}$ is functional and for every $s_1, s_1', s_2, s_2', s_3 \in \mathscr{S}$,
   (a) $(s_1, s_2) \in \mathscr{A}$ & $(s_1', s_2) \in \mathscr{A}$ $\Rightarrow$ $s_1 \equiv s_1'$
   (b) $(s_1, s_2, s_3) \in \mathscr{R}$ & $(s_1, s_2', s_3) \in \mathscr{R}$ $\Rightarrow$ $s_2 \equiv s_2'$

For pure type systems, the classification lemma is proved using *uniqueness of types*, a property which fails in even very simple domain-free pure type systems.

Lemma 19 (*Failure of Uniqueness of Types*)
In $\underline{\lambda}\!\!\rightarrow$, there exist M and Γ with $\Gamma \Vdash M : C$ and $\Gamma \Vdash M : C'$ but $C \neq_{\underline{\beta}} C'$.

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

Proof
Take $\Gamma \equiv y{:}*, z{:}*$ and $M \equiv \lambda x\,.\,x$. Then $\Gamma \Vdash M : y \to y$ and $\Gamma \Vdash M : z \to z$. □

The key to the proof of the classification lemma for DFPTSs is to observe that one can replace, throughout the proof, uses of the uniqueness of types property with uses of the weaker property of *preservation of sorts*, inspired from Geuvers (1993).

Definition 20
Let $\mathbf{S}$ be a specification.

1. $\mathbf{S}$ satisfies *uniqueness of types* if
$$\Gamma \Vdash M : A \;\&\; \Gamma \Vdash M : A' \quad \Rightarrow \quad A =_{\underline{\beta}} A'$$
2. $\mathbf{S}$ *preserves sorts* if for every $s, s' \in \mathscr{S}$,
$$\Gamma \Vdash A : s \;\&\; \Gamma \Vdash A' : s' \;\&\; A =_{\underline{\beta}} A' \quad \Rightarrow \quad s \equiv s'$$

Remark 21
Note that the uniqueness of types property is equivalent to
$$\Gamma \Vdash M : A \;\&\; \Gamma \Vdash M' : A' \;\&\; M =_{\underline{\beta}} M' \quad \Rightarrow \quad A =_{\underline{\beta}} A' \qquad (*)$$
Indeed, the implication from (*) to the uniqueness of types property is trivial – the latter is a special case of the former. For the opposite direction, assume
$$\Gamma \Vdash M : A \;\&\; \Gamma \Vdash M' : A' \;\&\; M =_{\underline{\beta}} M'$$
By Church-Rosser and subject reduction $\Gamma \Vdash M'' : A \;\&\; \Gamma \Vdash M'' : A'$ for some M'' so $A =_{\underline{\beta}} A'$ by uniqueness of types.
The preservation of sorts property is the special case of (*) where we only consider A's and A's that are sorts.

We now set out to prove the classification lemma for injective systems that in addition preserve sorts – this takes up the following two lemmas and two corollaries. After this we show that all functional specifications with normalizing types enjoy preservation of sorts.

Lemma 22
Let $\mathbf{S}$ preserve sorts and $M \in \underline{\mathsf{Type}}^s$. Then

1. $M \equiv x \Rightarrow x \in V^{s'}$ for some $(s, s') \in \mathscr{A}$;
2. $M \equiv s' \Rightarrow (s', s) \in \mathscr{A}$;
3. $M \not\equiv \lambda x\,.\,B$;
4. $M \equiv B\,A \Rightarrow B \in \underline{\mathsf{Term}}^{s_3} \;\&\; A \in \underline{\mathsf{Term}}^{s_1}$ for some $(s_1, s_2, s_3) \in \mathscr{R}, (s, s_2) \in \mathscr{A}$;
5. $M \equiv \Pi x{:}A\,.\,B \Rightarrow A \in \underline{\mathsf{Type}}^{s_1} \;\&\; B \in \underline{\mathsf{Type}}^{s_2}$ for some $(s_1, s_2, s) \in \mathscr{R}$.

Proof
Let $\Gamma \Vdash M : s$ and use in each case generation, making use of Church–Rosser, subject reduction, and preservation of sorts where necessary. □

Lemma 23
Let $\mathbf{S}$ preserve sorts and $M \in \underline{\mathsf{Term}}^s$. Then

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

1. $M \equiv x \Rightarrow x \in V^s$;
2. $M \equiv s' \Rightarrow (s', s''), (s'', s) \in \mathscr{A}$ for some s'';
3. $M \equiv \lambda x.B \Rightarrow x \in V^{s_1}$ & $B \in \underline{\mathsf{Term}}^{s_2}$ for some $(s_1, s_2, s) \in \mathscr{R}$;
4. $M \equiv B\,A \Rightarrow B \in \underline{\mathsf{Term}}^{s_3}$ & $A \in \underline{\mathsf{Term}}^{s_1}$ for some $(s_1, s, s_3) \in \mathscr{R}$;
5. $M \equiv \Pi x{:}A.B \Rightarrow A \in \underline{\mathsf{Type}}^{s_1}$ & $B \in \underline{\mathsf{Type}}^{s_2}$ for some $(s_1, s_2, s_3) \in \mathscr{R}, (s_3, s) \in \mathscr{A}$.

Proof
Let $\Gamma \Vdash M : D : s$ and use in each case generation, making use of Church–Rosser, subject reduction, and preservation of sorts where necessary. □

The following fundamental property is similar to results proved by Geuvers (1993), Berardi (1990), and Geuvers and Nederhof (1991).

Corollary 24 (*Classification Lemma*)
Let $\mathbf{S}$ be injective and preserve sorts. Then for all sorts $s \not\equiv s'$,

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

Proof
Use Lemmas 22 and 23 to prove

$$M \in \underline{\mathsf{Term}}^s \cap \underline{\mathsf{Term}}^{s'} \quad \Rightarrow \quad s \equiv s'$$
$$M \in \underline{\mathsf{Type}}^s \cap \underline{\mathsf{Type}}^{s'} \quad \Rightarrow \quad s \equiv s'$$

simultaneously by induction on M. □

Corollary 25
Let $\mathbf{S}$ be injective and preserve sorts, and M be legal. Then

1. $M \in \underline{\mathsf{Term}}^s$ for some $s \in \mathscr{S}$; or
2. $M \in \underline{\mathsf{Type}}^s$ for some $s \in \mathscr{S}_\top$; or
3. $M \equiv s$ for some $s \in \mathscr{S}_\top$.

Moreover, (1)–(3) are mutually exclusive and s is unique in (1)–(3).

Proof
We first show that one of (1)–(3) hold. Since M is legal either $\Gamma \Vdash M : A$ or $\Gamma \Vdash A : M$ for some A.

1. $\Gamma \Vdash M : A$. By correctness of types either $\Gamma \Vdash A : s$ or $A \in \mathscr{S}$.
   (a) $\Gamma \Vdash A : s$. Then $M \in \underline{\mathsf{Term}}^s$.
   (b) $A \in \mathscr{S}$. If A is a top-sort then $M \in \underline{\mathsf{Type}}^s$ for a top-sort s. If A is not a top-sort there is a sort s' with $(A, s') \in \mathscr{A}$, so $M \in \underline{\mathsf{Term}}^{s'}$.
2. $\Gamma \Vdash A : M$. By correctness of types, either $\Gamma \Vdash M : s$ or $M \in \mathscr{S}$. In the former case proceed as in Case 1. In the latter case, either M is a top-sort, or M is not a top-sort in which case proceed again as in Case 1.

By the preceding corollary it follows that s is unique in (1)–(3). It remains to show that the clauses (1)–(3) are mutually exclusive. This is proved by induction on M using Lemmas 22 and 23. □

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

We next show that all functional systems with $\underline{\mathsf{Type}} \subseteq \mathrm{WN}_{\underline{\beta}}$ preserve sorts; although this includes the cube specifications by Proposition 52, we also show directly (i.e. without using $\underline{\mathsf{Type}} \subseteq \mathrm{WN}_{\underline{\beta}}$) that these preserve sorts. Thus, the classification lemma holds for all injective systems with $\underline{\mathsf{Type}} \subseteq \mathrm{WN}_{\underline{\beta}}$, and for all cube specifications.

Lemma 26
The following specifications preserve sorts:

1. All cube-specifications;
2. All functional specifications with $\underline{\mathsf{Type}} \subseteq \mathrm{WN}_{\underline{\beta}}$.

Proof
We consider the two cases separately.

1. Let $\mathbf{S}$ be a cube-specification. First, show that $\Gamma \not\Vdash \Box : A$ for all Γ and A by induction on derivations. Using this then show, again by induction on derivations, that $\Gamma \Vdash M : \Box$ implies $M \in \underline{\mathscr{K}^+}$, where $\underline{\mathscr{K}^+}$ is defined by the abstract syntax
$$\underline{\mathscr{K}^+} = * \mid \Pi x{:}\underline{\mathscr{E}}.\, \underline{\mathscr{K}^+}$$
Now show by induction on M using generation that $\Gamma \Vdash M : *$ implies $M \notin \underline{\mathscr{K}^+}$.
Now, if $\Gamma \Vdash A : s$, $\Gamma \Vdash A' : s'$ and $A =_{\underline{\beta}} A'$, then by Church–Rosser and subject reduction $\Gamma \Vdash A'' : s$ and $\Gamma \Vdash A'' : s'$ for some A''. If $s \not\equiv s'$, e.g. $s \equiv *$ and $s' \equiv \Box$, then $A'' \notin \underline{\mathscr{K}^+}$ and $A'' \in \underline{\mathscr{K}^+}$, a contradiction.
2. Let $\mathbf{S}$ be a functional specification $\mathbf{S}$ with $\underline{\mathsf{Type}} \subseteq \mathrm{WN}_{\underline{\beta}}$. Define the class $\mathscr{U}$ by the abstract syntax
$$\begin{aligned} \mathscr{B} &= V \mid \mathscr{B}\,\underline{\mathscr{E}} \\ \mathscr{U} &= \mathscr{S} \mid \mathscr{B} \mid \Pi V{:}\mathscr{U}.\, \mathscr{U} \end{aligned}$$
Now show that for every $M \in \mathscr{U}$,
$$\Gamma \Vdash M : B \;\&\; \Gamma \Vdash M : C \quad \Rightarrow \quad B =_{\underline{\beta}} C$$
by induction on the definition of $\mathscr{U}$. Consider also the class $\mathscr{N}$ defined by the abstract syntax
$$\mathscr{N} = x \mid \lambda x.\, \underline{\mathscr{E}} \mid \mathscr{N}\,\underline{\mathscr{E}}$$
Now show by induction on M, using generation, that
$$\Gamma \Vdash M : \Pi x{:}A.\, B \Rightarrow M \in \mathscr{N}$$
Using this property show by induction on A that
$$A \in \mathrm{NF}_{\underline{\beta}} \;\&\; \Gamma \Vdash A : s \Rightarrow A \in \mathscr{U}$$
Thus,
$$A \in \mathrm{NF}_{\underline{\beta}} \;\&\; \Gamma \Vdash A : s \;\&\; \Gamma \Vdash A : s' \quad \Rightarrow \quad s \equiv s'$$
Since $\underline{\mathsf{Type}} \subseteq \mathrm{WN}_{\underline{\beta}}$ the claim now follows using Church–Rosser and subject reduction.

This concludes the proof. □

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

For some purposes it is desirable that the classification of expressions into categories be decidable. We conclude this subsection with such a decidable classification; for the sake of brevity, we consider the $\underline{\lambda}$-cube only.

Definition 27
Let $\underline{\mathscr{E}}' = \underline{\mathscr{O}} \cup \underline{\mathscr{C}} \cup \underline{\mathscr{K}} \cup \{\Box\}$ where the sets $\underline{\mathscr{O}}$, $\underline{\mathscr{C}}$, $\underline{\mathscr{K}}$ of domain-free expressions are given by the abstract syntax:

$$\begin{array}{rcl} \underline{\mathscr{O}} & = & V^* \mid \lambda V^*.\underline{\mathscr{O}} \mid \underline{\mathscr{O}}\,\underline{\mathscr{O}} \mid \lambda V^\Box.\underline{\mathscr{O}} \mid \underline{\mathscr{O}}\,\underline{\mathscr{C}} \\ \underline{\mathscr{C}} & = & V^\Box \mid \Pi V^*{:}\underline{\mathscr{C}}.\underline{\mathscr{C}} \mid \Pi V^\Box{:}\underline{\mathscr{K}}.\underline{\mathscr{C}} \mid \lambda V^*.\underline{\mathscr{C}} \mid \lambda V^\Box.\underline{\mathscr{C}} \mid \underline{\mathscr{C}}\,\underline{\mathscr{C}} \mid \underline{\mathscr{C}}\,\underline{\mathscr{O}} \\ \underline{\mathscr{K}} & = & * \mid \Pi V^*{:}\underline{\mathscr{C}}.\underline{\mathscr{K}} \mid \Pi V^\Box{:}\underline{\mathscr{K}}.\underline{\mathscr{K}} \end{array}$$

The following is a decidable variant of Corollary 25 for the $\underline{\lambda}$-cube.

Lemma 28
Let $\mathbf{S} = C$ and $M \in \underline{\mathscr{E}}$ be legal. Then

1. $M \in \underline{\mathscr{O}}$; or
2. $M \in \underline{\mathscr{C}}$; or
3. $M \in \underline{\mathscr{K}}$; or
4. $M \equiv \Box$.

Moreover, the sets $\underline{\mathscr{O}}, \underline{\mathscr{C}}, \underline{\mathscr{K}}, \{\Box\}$ are pairwise disjoint.

Remark 29
It is a routine matter to derive corresponding classifications for the remaining cube specifications. For $\rightarrow$, one takes

$$\underline{\mathscr{O}} = V^* \mid \lambda V^*.\underline{\mathscr{O}} \mid \underline{\mathscr{O}}\,\underline{\mathscr{O}} \qquad \underline{\mathscr{C}} = V^\Box \mid \Pi V^*{:}\underline{\mathscr{C}}.\underline{\mathscr{C}} \qquad \underline{\mathscr{K}} = *$$

Moreover,

1. For 2 one takes in addition to the classes for $\rightarrow$
$$\underline{\mathscr{O}} = \ldots \mid \lambda V^\Box.\underline{\mathscr{O}} \mid \underline{\mathscr{O}}\,\underline{\mathscr{C}} \qquad \mathscr{C} = \ldots \mid \Pi V^\Box{:}\underline{\mathscr{K}}.\underline{\mathscr{C}}$$
2. For P one takes in addition to the classes for $\rightarrow$:
$$\underline{\mathscr{C}} = \ldots \mid \lambda V^*.\underline{\mathscr{C}} \mid \underline{\mathscr{C}}\,\underline{\mathscr{O}} \qquad \underline{\mathscr{K}} = \ldots \mid \Pi V^*{:}\underline{\mathscr{C}}.\underline{\mathscr{K}}$$
3. For $\underline{\omega}$ one takes in addition to the classes for $\rightarrow$
$$\underline{\mathscr{C}} = \ldots \mid \lambda V^\Box.\underline{\mathscr{C}} \mid \underline{\mathscr{C}}\,\underline{\mathscr{C}} \qquad \underline{\mathscr{K}} = \ldots \mid \Pi V^\Box{:}\underline{\mathscr{K}}.\underline{\mathscr{K}}$$

For $C = 2P\underline{\omega}$, $2P$, $\omega = 2\underline{\omega}$, and $P\underline{\omega}$ one combines the above sets.

As one might guess, the relation between the characterizations in Corollary 25 and Lemma 28 is as follows.

Lemma 30
Let $\mathbf{S} = C$. If $M \in \underline{\mathscr{E}}$ is legal then

$$\begin{array}{rcl} M \in \underline{\mathsf{Term}}^* & \Leftrightarrow & M \in \underline{\mathscr{O}} \\ M \in \underline{\mathsf{Term}}^\Box & \Leftrightarrow & M \in \underline{\mathscr{C}} \\ M \in \underline{\mathsf{Type}}^\Box & \Leftrightarrow & M \in \underline{\mathscr{K}} \end{array}$$

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

3.3 Type-checking

Decidability of type-checking and related problems is a central issue in the design of programming languages and proof-assistants. Many modern programming languages (e.g. ML) are designed so as to accommodate an implicit programming style in which typing information is inferred automatically by the compiler. Proof assistants are usually designed in such a way that proof-checking can be done mechanically.

Depending upon the style of programming language or proof-assistant considered, decidability questions may take several forms. Here we shall concern ourselves with the following three questions.

Definition 31
Let $\mathbf{S}$ be a specification.

1. The *type-checking* problem (TC) consists in deciding, given Γ, M, A, whether $\Gamma \vdash M : A$; symbolically: $\Gamma \vdash M : A$?.
2. The *type-synthesis* problem (TS) consists in deciding, given Γ, M, whether there exists A such that $\Gamma \vdash M : A$; symbolically: $\Gamma \vdash M :?$.
3. The *strong typability* problem (TY) consists in deciding, given Γ_0 and M, whether there exists Γ, A such that $\Gamma_0, \Gamma \vdash M : A$; symbolically: $\Gamma_0, ? \vdash M :?$.[5]

First, we focus on the systems $\underline{\lambda}\!\!\rightarrow$, $\underline{\lambda}\,\underline{\omega}$ $\underline{\lambda}2$ and $\underline{\lambda}P$. We show that polymorphism, i.e. the rule $(\Box, *)$ of $\underline{\lambda}2$, and dependencies, i.e. the rule $(*, \Box)$ of $\underline{\lambda}P$, lead to undecidable type-checking. Then, we show that normalizing DFPTSs have a weak form of decidable type-checking. Note that undecidability of type-checking for $\underline{\lambda}2$ was proved independently by Fujita (1999) in a recent manuscript.

Theorem 32 (*Decidability results for the $\underline{\lambda}$-cube*)
1. For $\underline{\lambda}\!\!\rightarrow$: TC, TS and TY are decidable.
2. For $\underline{\lambda}2$: TC, TS and TY are undecidable.
3. For $\underline{\lambda\omega}$: TC, TS and TY are decidable.
4. For $\underline{\lambda}P$: TC and TS are undecidable.

Proof
We use the variant of DFPTSs with sorted variables (see Figure 4), and use the classification of Lemma 28. Besides we let $\mathsf{K} \equiv \lambda x.\lambda y.x$, $\mathsf{triv} \equiv \lambda\beta.\lambda z.z$ and $\bot = \Pi\alpha{:}*.\,\alpha$.

1. See, for example, Barendregt (1992).
2. This is the most interesting result. We prove the undecidability of TY, then deduce undecidability of TS from that of TY, and finally deduce undecidability of TC from that of TS.
   - *TY is undecidable for $\underline{\lambda}2$.* Define a translation[6] $\lceil\bullet\rceil : \mathscr{O} \to \underline{\mathscr{O}}$ as follows,

[5] A specific instance of the strong typability problem is the typability problem, where Γ_0 is taken to be the empty context.
[6] The categories $\mathscr{O}, \mathscr{C}, \mathscr{K}$ of domain-full expressions are defined in section 4.1 where a corresponding classification lemma is proved.

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

where δ is a fresh variable:

$$\begin{array}{rcl} \lceil x \rceil & = & x \\ \lceil t\ u \rceil & = & \lceil t \rceil\ \lceil u \rceil \\ \lceil t\ A \rceil & = & \lceil t \rceil\ A \\ \lceil \lambda x : A.t \rceil & = & \lambda x.\mathsf{K}\ \lceil t \rceil\ (\delta\ (A \to A)\ x) \\ \lceil \lambda x : *.t \rceil & = & \lambda x.\ \lceil t \rceil \end{array}$$

where $t, u \in \mathscr{O}$ and $A \in \mathscr{C}$. Then one can prove by induction on the structure of M that

$$\Gamma \vdash M : A \quad \Leftrightarrow \quad \delta : \bot, \Gamma \Vdash \lceil M \rceil : A$$

Hence we have for every $M \in \mathscr{O}$,

$$\exists \Gamma \in \mathscr{G}.\ \exists A \in \mathscr{C}.\ \Gamma \vdash M : A \Leftrightarrow \exists \Gamma \in \underline{\mathscr{G}}.\ \exists A \in \underline{\mathscr{C}}.\ \delta : \bot, \Gamma \Vdash \lceil M \rceil : A$$

Schubert (1998) has shown that the left-hand side is undecidable. It follows that the right-hand side is also undecidable. The right-hand side is an instance of (TY) with $\Gamma_0 = \delta : \bot$, therefore TY for $\lambda 2$ is undecidable.

- *TS is undecidable for* $\underline{\lambda}2$. Let $M \in \underline{\mathscr{O}}$, $\{\alpha_1, \ldots, \alpha_p\} = \mathrm{FV}(M) \cap V^{\Box}$ and $\{x_1, \ldots, x_n\} = \mathrm{FV}(M) \cap V^{*}$. Then

$$\begin{array}{ll} & \exists C \in \underline{\mathscr{C}}.\ \ \delta : \bot \Vdash (\lambda \alpha_1, \ \ldots\ , \alpha_p, x_1, \ldots, x_n\,.\, M) : C \\ \Leftrightarrow & \exists \vec{B}, C \in \underline{\mathscr{C}}.\ \ \delta : \bot, \alpha_1 : *, \ \ldots\ , \alpha_p : *, x_1 : B_1, \ldots, x_n : B_n \Vdash M : C \\ \Leftrightarrow & \exists \Gamma \in \underline{\mathscr{G}}, C \in \underline{\mathscr{C}}.\ \ \delta : \bot, \Gamma \Vdash M : C \end{array}$$

In the first step, we use the generation lemma. In the second step, only the reverse direction requires some justification. For it, observe that (1) the permutation lemma holds, so the context may be reorganized in such a way that constructor variables are shifted to the left and object variables are shifted to the right (2) some weak form of strengthening holds *in presence of the assumption* $\delta : \bot$. More precisely, we have from the substitution lemma that if $\delta : \bot, \Gamma_1, x : A, \Gamma_2 \Vdash M : C$ with $x \notin \mathrm{FV}(M)$ then

$$\begin{array}{rcl} x \in V^{*} & \Rightarrow & \delta : \bot, \Gamma_1, \Gamma_2 \Vdash M : C \\ x \in V^{\Box} & \Rightarrow & \delta : \bot, \Gamma_1, \Gamma_2\{x := \bot\} \Vdash M : C\{x := \bot\} \end{array}$$

where in the first case one applies the substitution $\{x := \delta\ A\}$. The last formula is undecidable so the first formula must also be undecidable. Thus TS is undecidable.

- *TC is undecidable for* $\underline{\lambda}2$. Let $\Gamma \in \underline{\mathscr{G}}$ and $M \in \underline{\mathscr{O}}$. By generation

$$\exists A \in \underline{\mathscr{C}}.\ \Gamma \Vdash M : A \quad \Leftrightarrow \quad \Gamma \Vdash (\lambda u\,.\,\mathsf{triv})\ M : \Pi \beta{:}*.\ \beta \to \beta$$

The left formula is undecidable so the right formula must also be undecidable. Thus TC is undecidable.

3. TC, TS, and TY for $M \in \underline{\mathscr{C}}$ are decidable; it is not hard to see that these problems are variants of TC, TS, and TY for $M \in \underline{\mathscr{O}}$ in $\underline{\lambda}\!\to$, which are decidable. Hence, we only need to treat the case where $M \in \underline{\mathscr{O}}$. For brevity, we only consider TY. Assume we are given a pseudo-context Γ_0. First, note that

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

M is not typable unless $\mathrm{FV}(M) \subseteq V^*$ since legal objects of $\underline{\lambda\omega}$ do not contain constructors as subterms. Secondly, assume that $\mathrm{FV}(M) \setminus \Gamma_0 = \{x_1, \ldots, x_n\} \subseteq V^*$. Then

$$\exists C \in \underline{\mathscr{C}}.\ \Gamma_0, \gamma : * \vdash \lambda \vec{x}.\, M : C \quad (\&)$$

is decidable by the same algorithm as in Barendregt (1992). By generation, the formula (&) is equivalent to

$$\exists \vec{B}, C \in \underline{\mathscr{C}}.\ \Gamma_0, \gamma : *, x_1 : B_1, \ \ldots\ , x_n : B_n \vdash M : C \quad (\&\&)$$

which is thus also decidable. Now we claim (&&) is equivalent to

$$\exists \Gamma \in \underline{\mathscr{G}}.\ \exists C \in \underline{\mathscr{C}}.\ \Gamma_0, \Gamma \vdash M : C$$

and this equivalence implies decidability of TY. To justify the last equivalence, we only need to consider the reverse implication. We proceed almost exactly as in 2. So assume $\Gamma_0, \Gamma \vdash M : C$. By Thinning, $\Gamma_0, \gamma : *, \Gamma \vdash M : C$ for γ fresh. Now one can define for every kind K an element can_K such that $\Gamma_0, \gamma : * \vdash \mathsf{can}_K : K$. By repeated applications of the substitution lemma,

$$\Gamma_0, \gamma : *, \Gamma' \vdash M : C$$

where all the variables in Γ' are object variables. We conclude, by applying strengthening on object variables, that

$$\Gamma_0, \gamma : *, x_1 : B_1, \ \ldots\ , x_n : B_n \vdash M : C$$

for suitable B_is.

4. See Dowek (1993) – the results are proved in a somewhat disguised form.

This concludes the proof. □

Note that Theorem 32 does not solve the problem of decidability of typability for $\underline{\lambda 2}$.

Theorem 32 shows that type-checking may be undecidable in even rather weak DFPTSs. Still, one can establish a weak decidability result, which fortunately is sufficient in practice for DFPTSs to be used as the basis of proof-assistants (e.g. see Betarte and Tasistro (1998) and Magnusson (1994)).

Proposition 33
Let **S** be a specification such that:

1. $\mathscr{S}$ is finite;
2. $\mathscr{A}$ and $\mathscr{R}$ are decidable;
3. $\underline{\mathsf{Type}} \subseteq \mathrm{WN}_{\underline{\beta}}$;

Assume $\Gamma \vdash A : s$ for some s with $A \in \mathrm{NF}_{\underline{\beta}}$ and $B \in \mathrm{NF}_{\underline{\beta}}$ for every $y : B \in \Gamma$. Given $M \in \mathrm{NF}_{\underline{\beta}}$, it is decidable whether $\Gamma \vdash M : A$.

Proof
By induction on the structure of the term M. We treat two cases:

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

1. $M \equiv \lambda x\,.\,P$
   First, if A does not have form $\Pi x : D'.\, E'$ then $\Gamma \not\Vdash M : A$. Indeed, if $\Gamma \Vdash M : A$ then by generation $\Gamma, x : D \Vdash P : E$ for some $\Pi x{:}\, D.\, E =_{\underline{\beta}} A$. Since $A \in \mathrm{NF}_{\underline{\beta}}$, $A \equiv \Pi x{:}\, D'.\, E'$ for some D', E', a contradiction. Secondly, suppose $A \equiv \Pi x{:}\, D'.\, E'$. Then, by generation and subject reduction, $\Gamma \Vdash M : A$ iff $\Gamma, x : D' \Vdash P : E'$, which is decidable by induction hypothesis.
2. $M \equiv x\ P_1\ \ldots\ P_n$
   First, if $x \notin \mathrm{dom}(\Gamma)$, then $\Gamma \not\Vdash M : A$ by the free variables lemma. Secondly, suppose $x : B \in \Gamma$. If B does not have the form $\Pi z_1{:}\, C_1.D_1$ then by generation, $\Gamma \not\Vdash M : A$. Thirdly, suppose $x : B \in \Gamma$ with $B \equiv \Pi z_1{:}\, C_1.D_1$. Then by generation and subject reduction, $\Gamma \Vdash M : A$ iff

$$\begin{cases} \Gamma \Vdash P_1 : C_1 \\ \Gamma \Vdash P_2 : C_2 \\ \quad\vdots \\ \Gamma \Vdash P_n : C_n \end{cases}$$

   with (we use $\mathrm{nf}(T)$ to denote the $\underline{\beta}$-normal form of T)

$$\begin{aligned} \mathrm{nf}(D_1\{z_1 := P_1\}) &= \Pi z_2{:}\, C_2.\, D_2 \\ \vdots &\quad \vdots \\ \mathrm{nf}(D_{n-1}\{z_{n-1} := P_{n-1}\}) &= \Pi z_n{:}\, C_n.\, D_n \\ \mathrm{nf}(D_n\{z_n := P_n\}) &= A \end{aligned}$$

   Each of the C_i and D_i is totally determined by the C_j's and D_j's such that $j < i$ (in particular, they are defined otherwise the judgement would not be derivable).[7] Hence the conjuncts above are completely determined. Each of the conjunct is decidable by induction hypothesis so we are done.

The decidability of $\mathscr{A}$ is used in the case where M is a sort. The finiteness of $\mathscr{S}$ and the decidability of $\mathscr{R}$ are used in the case where M is a product. □

For domain-free pure type systems with a cumulative hierarchy of universes, such as a domain-free variant of ECC (Luo, 1994), one may define in a similar way an algorithm that computes a principal type of a term in $\underline{\beta}$-normal form in a context in $\underline{\beta}$-normal form.

4 Domain-free pure type systems versus pure type systems

In this section we compare domain-free pure type systems with conventional pure type systems. The terminology for the latter is that of Barendregt (1992), to which the reader is referred for further details.

In the first subsection we introduce pure type systems. In the second subsection we show that every derivation in a pure type system can be projected to a derivation

[7] By a theorem due to Curry and Feys (1958), the normal form $\mathrm{nf}(Q)$ of Q can be computed from Q by repeatedly contracting the left-most redex.

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

in the corresponding domain-free pure type system. In the third subsection we show that the converse also holds for a large class of specifications; this is the main result of the section.

4.1 Pure type systems

A specification $\mathbf{S}$ induces a pure type system $\lambda\mathbf{S}$ as follows.

Definition 34
Let $\mathbf{S} = (\mathscr{S}, \mathscr{A}, \mathscr{R})$ be a specification.

1. The set $\mathscr{E}$ of *(domain-full) expressions* (over $\mathscr{S}$) is given by the abstract syntax:
$$\mathscr{E} = V \mid \mathscr{S} \mid \mathscr{E}\mathscr{E} \mid \lambda V : \mathscr{E}.\mathscr{E} \mid \Pi V : \mathscr{E}.\mathscr{E}$$
We use the same naming conventions as for domain-free pure type systems and assume, again, that the reader is familiar with the notions of free and bound variables, and related conventions; $\mathrm{FV}(M)$ denotes the set of variables occurring free in M, and $\equiv$ denotes syntactic equality.
2. A *(domain-full) context* is a finite sequence of form $x_1 : A_1, \ldots, x_n : A_n$; the empty sequence is written $\langle\rangle$. The set of all contexts is called $\mathscr{G}$. We write $\mathrm{dom}(x_1 : A_1, \ldots, x_n : A_n) = \{x_1, \ldots, x_{nm}\}$ and use the same naming conventions as for domain-free contexts.
3. *β-reduction* $\to_\beta$ on $\mathscr{E}$ 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 obvious substitution operator. *Multi-step β-reduction* $\twoheadrightarrow_\beta$ and *β-equality* $=_\beta$ are the reflexive, transitive closure and reflexive, transitive, symmetric closure, respectively, of $\to_\beta$. The relation $\to_\beta$ is extended to $\mathscr{G}$ by:
$$A \to_\beta B \Rightarrow \Gamma, x : A, \Delta \to_\beta \Gamma, x : B, \Delta$$
The relations $\twoheadrightarrow_\beta$ and $=_\beta$ on $\mathscr{G}$ are the obvious closures of $\to_\beta$ on $\mathscr{G}$.
4. The *derivability* relation $\vdash$ is given by the rules of Figure 5. We occasionally write $\vdash_{\mathbf{S}}$ to explicate the dependence of the derivability relation on $\mathbf{S}$. If $\Gamma \vdash A : B$ then Γ, A, and B are *legal*. We write $\Gamma \vdash A : B : C$ if $\Gamma \vdash A : B$ and $\Gamma \vdash B : C$.
5. The tuple $\lambda\mathbf{S} = (\mathscr{E}, \mathscr{G}, =_\beta, \vdash)$ is the *pure type system (*PTS*) induced by* $\mathbf{S}$.

The following notions are analogous to those for DFPTSs.

Definition 35
Let $\mathbf{S}$ be a specification and $s \in \mathscr{S}$.

$$\begin{aligned} \mathsf{Type}^s &= \{M \in \mathscr{E} \mid \Gamma \vdash M : s \text{ for some } \Gamma\} \\ \mathsf{Term}^s &= \{M \in \mathscr{E} \mid \Gamma \vdash M : A : s \text{ for some } \Gamma \text{ and } A\} \end{aligned}$$

Also, $\mathsf{Type} = \cup_{s \in \mathscr{S}} \mathsf{Type}^s$ and $\mathsf{Term} = \cup_{s \in \mathscr{S}} \mathsf{Term}^s$.

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

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

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

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

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

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

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

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

Fig. 5. Pure type systems.

The notion of a *β-reduction path* and the associated sets NF_β, WN_β, and SN_β of *β-normal forms, β(-weakly) normalizing* expressions, and *β-strongly normalizing* expressions, respectively, are defined analogously to Definitions 6 and 7.

Finally, we present a classification lemma for the λ-cube, which is used in section 6 and in section 3.3.

Definition 36
Let $\mathscr{E}' = \mathscr{O} \cup \mathscr{C} \cup \mathscr{K} \cup \{\Box\}$ be the subset of $\mathscr{E}$ where the sets $\mathscr{O}$, $\mathscr{C}$, $\mathscr{K}$ of domain-full expressions are given by the abstract syntax:

$$\begin{aligned} \mathscr{O} &= V^* \mid \lambda V^* : \mathscr{C}\,.\,\mathscr{O} \mid \mathscr{O}\,\mathscr{O} \mid \lambda V^\Box : \mathscr{K}\,.\,\mathscr{O} \mid \mathscr{O}\,\mathscr{C} \\ \mathscr{C} &= V^\Box \mid \Pi V^* : \mathscr{C}.\,\mathscr{C} \mid \Pi V^\Box : \mathscr{K}.\,\mathscr{C} \mid \lambda V^* : \mathscr{C}\,.\,\mathscr{C} \mid \lambda V^\Box : \mathscr{K}\,.\,\mathscr{C} \mid \mathscr{C}\,\mathscr{C} \mid \mathscr{C}\,\mathscr{O} \\ \mathscr{K} &= * \mid \Pi V^* : \mathscr{C}.\,\mathscr{K} \mid \Pi V^\Box : \mathscr{K}.\,\mathscr{K} \end{aligned}$$

The following result is similar to Lemma 28.

Lemma 37
Let $\mathbf{S} = C$ and $M \in \mathscr{E}$ be legal. Then

1. $M \in \mathscr{O}$; or
2. $M \in \mathscr{C}$; or
3. $M \in \mathscr{K}$; or
4. $M \equiv \Box$.

Moreover, the sets $\mathscr{O}, \mathscr{C}, \mathscr{K}, \{\Box\}$ are pairwise disjoint.

4.2 Erasing

Every domain-full expression induces a domain-free expression by erasing the domains of abstractions. This erasing function is used by Geuvers (1993) to study PTSs

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

with $\beta\eta$-conversion; it also appears in the context of various specifications in the literature.

Definition 38
The *erasure* map $|\bullet| : \mathscr{E} \to \underline{\mathscr{E}}$ is defined as follows:

$$\begin{aligned} |x| &= x \\ |s| &= s \\ |t\ u| &= |t|\ |u| \\ |\lambda x{:}A.t| &= \lambda x.|t| \\ |\Pi x{:}A.B| &= \Pi x : |A|.|B| \end{aligned}$$

Erasure preserves reduction, equality and typing:

Proposition 39 (*Erasing*)
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 derivation of $M \to_\beta N$. Then prove (2) by induction on the derivation of $M =_\beta N$ using (1). Finally, prove (3) by induction on the derivation of $\Gamma \vdash M : A$, using (*) and (2). □

4.3 Lifting

The main result of this section is that, under suitable conditions, derivations may be lifted along $|\bullet|$. This generalizes (Barendregt, 1992) Proposition 3.2.15 where a similar result is proved for simply typed λ-calculus (see also section 6). Unless explicitly stated, properties such as Church-Rosser, subject reduction, and context conversion, used below, refer to PTSs.

Lemma 40 (*Context conversion*)
Let **S** be a specification. Assume that $\Gamma \in \mathscr{G}$ is legal, $\Delta \vdash M : A$ and $\Gamma =_\beta \Delta$. Then $\Gamma \vdash M : A$.

Proof
By Church–Rosser, there exists Ξ s.t. $\Gamma \twoheadrightarrow_\beta \Xi$ and $\Delta \twoheadrightarrow_\beta \Xi$. By subject reduction $\Xi \vdash M : A$ and $\Gamma \vdash x{:}A$ for each $x{:}A$ in Ξ. By transitivity, $\Gamma \vdash M : A$. □

Before proving the main result we prove three preliminary lemmas. The first lemma gives some useful information on the relation between $\to_\beta$ and $\to_{\underline{\beta}}$.

Lemma 41 (*Geuvers (1993)*)
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 $A =_\beta s$.

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

Proof

1. The redex contracted in $|A| \to_{\underline{\beta}} F$ corresponds to a unique redex in A; contracting this redex in A leads to a B with $|B| \equiv F$.
2. By Church-Rosser of $\to_{\underline{\beta}}$ on $\underline{\mathscr{E}}$, $|A| \twoheadrightarrow_{\underline{\beta}} s$. Now use (1).

This concludes the proof. □

The second lemma establishes a fundamental property of erasure.

Lemma 42

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 $A \equiv s$ and $A' \equiv s'$ then $M \equiv M'$.

Proof

We prove (1)–(2) simultaneously by induction on M.

1. $M \equiv s''$. Then $|M'| \equiv |M| \equiv |s''| \equiv s''$, so $M' \equiv s'' \equiv M$.
2. $M \equiv x$. Similar to Case 1.
3. $M \equiv \lambda x : D\,.\,b$. Then $M' \equiv \lambda x : D'\,.\,b'$, where $|b| \equiv |b'|$. By generation we have $A =_\beta \Pi x{:}\,D.B$ where $\Gamma \vdash \Pi x{:}\,D.B : s$ and $\Gamma, x{:}\,D \vdash b : B$ and $A' =_\beta \Pi x{:}\,D'.B'$ where $\Gamma \vdash \Pi x{:}\,D'.\,B' : s'$ and $\Gamma, x{:}\,D' \vdash b' : B'$.
   (1) By Church–Rosser, $D =_\beta D'$ and $B =_\beta B'$. Since D, D' are normal forms, $D \equiv D'$. By Church-Rosser and subject reduction there is B'' such that $\Gamma, x : D \vdash b : B''$ and $\Gamma, x : D \vdash b' : B''$. By part (1) of the induction hypothesis $b \equiv b'$, so $M \equiv M'$.
   (2) Impossible since $s \equiv A =_\beta \Pi x{:}\,D.\,B$ contradicts Church–Rosser.
4. $M \equiv \Pi x{:}\,B.\,C$. Then $M' \equiv \Pi x{:}\,B'.\,C'$, where $|B| \equiv |B'|$ and $|C| \equiv |C'|$. By generation, $\Gamma \vdash B : s_1$, $\Gamma, x{:}\,B \vdash C : s_2$, and $A =_\beta s_3$, and $\Gamma \vdash B' : s_1'$, $\Gamma, x{:}\,B' \vdash C' : s_2'$, and $A' =_\beta s_3'$. By part (2) of the induction hypothesis $B \equiv B'$ and then $C \equiv C'$. Hence $M \equiv M'$.
5. $M \equiv N\,P$. Since M is normal and legal, in fact $M \equiv x\,M_1 \ldots M_n$ and $M' \equiv x\,M_1' \ldots M_n'$, where $|M_1| \equiv |M_1'|, \ldots, |M_n| \equiv |M_n'|$.
   We now show by induction on n that
   (a) $x\,M_1 \ldots M_n \equiv x\,M_1' \ldots M_n'$;
   (b) $A =_\beta A'$.
   where (a) implies (1)–(2). If $n = 0$ then (a) is trivial. Moreover, by generation we have $x{:}\,E \in \Gamma$ where $A =_\beta E =_\beta A'$, proving (b). If $n = m + 1$, then by generation $\Gamma \vdash x\,M_1 \ldots M_m : \Pi y{:}\,B.\,C$, $\Gamma \vdash M_{m+1} : B$, $A =_\beta C[y := M_{m+1}]$ and $\Gamma \vdash x\,M_1' \ldots M_m' : \Pi y{:}\,B'.\,C'$, $\Gamma \vdash M_{m+1}' : B'$, $A' =_\beta C'[y := M_{m+1}']$ By part (b) of the induction hypothesis $\Pi y{:}\,B.C =_\beta \Pi y{:}\,B'.C'$. Hence by Church–Rosser $B =_\beta B'$ and $C =_\beta C'$. By Church–Rosser and subject reduction there is a B'' such that $\Gamma \vdash M_{m+1} : B''$ and $\Gamma \vdash M_{m+1}' : B''$. Hence by part (1) of the induction hypothesis $M_{m+1} \equiv M_{m+1}'$. By induction hypothesis (a) $x\,M_1 \ldots M_m \equiv x\,M_1' \ldots M_m'$. Hence $x\,M_1 \ldots M_{m+1} \equiv x\,M_1' \ldots M_{m+1}'$, and also $A =_\beta C[y := M_{m+1}] =_\beta C'[y := M_{m+1}'] =_\beta A'$.

This concludes the proof. □

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

Remark 43
Lemma 42 (1) is not true if we drop the assumption $M, M' \in \mathrm{NF}_\beta$. Take

$$\begin{aligned} \Gamma &\equiv A : *, A' : *, B : *, b : B \\ M &\equiv (\lambda x{:}\, A \to A.\ b)\ (\lambda y : A.\ y) \\ M' &\equiv (\lambda x{:}\, A' \to A'.\ b)\ (\lambda y : A'.\ y) \end{aligned}$$

Lemma 44
Let $\mathbf{S}$ be a specification with $\mathsf{Type} \subseteq \mathrm{WN}_\beta$.

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 $M =_\beta N$.

Proof
1. Let M' and N' be normal forms of M and N, respectively. By subject reduction $\Gamma \vdash M' : s$, $\Gamma \vdash N' : s'$. By projection $|M'| =_{\underline{\beta}} |M| =_{\underline{\beta}} |N| =_{\underline{\beta}} |N'|$. By Church–Rosser on $\underline{\mathscr{E}}$, $|M'| \equiv |N'|$. Hence by Lemma 42(2) $M' \equiv N'$. Hence $M =_\beta M' \equiv N' =_\beta N$.
2. By induction on the length of Γ_1. If $\Gamma_1 \equiv \langle\rangle$, then $\Gamma_1 \equiv \langle\rangle \equiv \Gamma_2$.
   If $\Gamma_1 \equiv \Delta_1, x{:}\, A_1$ then $\Gamma_2 \equiv \Delta_2, x{:}\, A_2$ where $|\Delta_1| =_{|\beta|} |\Delta_2|$ and $|A_1| =_{|\beta|} |A_2|$. By induction hypothesis $\Delta_1 =_\beta \Delta_2$. By the start lemma $\Delta_1 \vdash A_1 : s_1$ and $\Delta_2 \vdash A_2 : s_2$. By Church–Rosser and subject reduction there is Δ such that $\Delta \vdash A_1 : s_1$ and $\Delta \vdash A_2 : s_2$. Hence by (1), $A_1 =_\beta A_2$.
3. By correctness of types, either M is a top-sort or $\Gamma \vdash M : s$ for some $s \in \mathscr{S}$. If M is a top-sort, then $N \twoheadrightarrow_\beta M$ by Lemma 41 and $\Gamma \vdash M : s'$ by subject reduction, a contradiction. Hence $\Gamma \vdash M : s$ for some $s \in \mathscr{S}$. Apply (1).

This concludes the proof. □

We can now proceed with the proof of the main result.

Theorem 45 (*Lifting*)
Let $\mathbf{S}$ be a functional specification with $\mathsf{Type} \subseteq \mathrm{WN}_\beta$. If $\Delta \Vdash E : F$ then $\Gamma \vdash A : B$ for some Γ, A, B with $|\Gamma| \equiv \Delta$, $|A| \equiv E$, and $|B| \equiv F$.

Proof
By induction on the derivation of $\Delta \Vdash E : F$.

1. The derivation ends in
$$\boxed{\langle\rangle \Vdash s_1 : s_2}$$
   Choose $\Gamma \equiv \langle\rangle$, $A \equiv s_1$, $B \equiv s_2$.
2. The derivation ends in
$$\boxed{\frac{\Delta \Vdash E : s}{\Delta, x{:}\, E \Vdash x : E} \qquad x \notin \mathrm{dom}(\Gamma)}$$
   By induction hypothesis $\Gamma \vdash A : s$ with $|\Gamma| \equiv \Delta$ and $|A| \equiv E$. Therefore $\Gamma, x{:}\, A \vdash x : A$ with $|\Gamma, x{:}\, A| \equiv \Delta, x{:}\, E$, $|x| \equiv x$ and $|A| \equiv E$.

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

3. The derivation ends in

$$\frac{\Delta \vDash E : F \qquad \Delta \vDash G : s}{\Delta, x{:}\, G \vDash E : F} \quad x \notin \mathrm{dom}(\Gamma)$$

By induction hypothesis $\Gamma_1 \vdash A : B$ and $\Gamma_2 \vdash C : s$, where $|\Gamma_1| \equiv \Delta \equiv |\Gamma_2|$, $|A| \equiv E$, $|B| \equiv F$, and $C \equiv G$. By Lemma 44 (2), $\Gamma_1 =_\beta \Gamma_2$. By Context Conversion, $\Gamma_1 \vdash C : s$. Hence by weakening $\Gamma_1, x{:}\, C \vdash A : B$.

4. The derivation ends in

$$\frac{\Delta \vDash E : s_1 \qquad \Delta, x{:}\, E \vDash F : s_2}{\Delta \vDash (\Pi x{:}\, E.\, F) : s_3}$$

By induction hypothesis $\Gamma_1 \vdash A_1 : s_1$ and $\Gamma_2, x{:}\, A_2 \vdash B : s_2$ where $|\Gamma_1| \equiv \Delta \equiv |\Gamma_2|$, $|A_1| \equiv E \equiv |A_2|$, and $|B| \equiv F$. By Lemma 44 (1-2), $\Gamma_1 =_\beta \Gamma_2$ and $A_1 =_\beta A_2$. By context conversion, $\Gamma_2 \vdash A_1 : s_1$. By the start lemma $\Gamma_2 \vdash A_2 : s'$. By uniqueness of types, $s_1 \equiv s'$. Hence by product $\Gamma \vdash \Pi x{:}\, A_2.\, B_2 : s_3$.

5. The derivation ends in

$$\frac{\Delta \vDash E : (\Pi x{:}\, F.\, G) \qquad \Gamma \vDash f : F}{\Gamma \vDash E\ f : G\{x := f\}}$$

By induction hypothesis $\Gamma_1 \vdash A : \Pi x : B_1.\, C$ and $\Gamma_2 \vdash b : B_2$ where $|\Gamma_1| \equiv \Delta \equiv |\Gamma_2|$, $|A| \equiv E$, $|B_1| \equiv F \equiv |B_2|$, $|C| \equiv G$, and $|b| \equiv f$. By Lemma 44 (2), $\Gamma_1 =_\beta \Gamma_2$. By context conversion, $\Gamma_2 \vdash A : \Pi x{:}\, B_1.\, C$. By generation, $\Gamma_2 \vdash B_1 : s_1$. By Lemma 44 (3), $B_1 =_\beta B_2$. By conversion, $\Gamma_2 \vdash b : B_1$. Hence by application $\Gamma_2 \vdash A\, b : C[x := b]$.

6. The derivation ends in

$$\frac{\Delta, x{:}\, E \vDash f : F \qquad \Delta \vDash (\Pi x{:}\, E.\, F) : s}{\Delta \vDash \lambda x.\, f : \Pi x{:}\, E.\, F}$$

By induction hypothesis $\Gamma_1, x{:}\, A_1 \vdash b : B_1$ and $\Gamma_2 \vdash \Pi x{:}\, A_2.\, B_2 : s$ where $|\Gamma_1| \equiv \Delta \equiv |\Gamma_2|$, $|A_1| \equiv E \equiv |A_2|$, $|B_1| \equiv F \equiv |B_2|$, $|b| \equiv f$. By generation, $\Gamma_2, x{:}\, A_2$ is a legal context. By Lemma 44 (2), $\Gamma_1, x{:}\, A_1 =_\beta \Gamma_2, x{:}\, A_2$. By context conversion, $\Gamma_2, x{:}\, A_2 \vdash b : B_1$. By generation, $\Gamma_2, x{:}\, A_2 \vdash B_2 : s'$. By Lemma 44 (3), $B_1 =_\beta B_2$. By conversion, $\Gamma_2, x{:}\, A_2 \vdash b : B_2$. Hence by abstraction $\Gamma_2 \vdash \lambda x{:}\, A_2.\, b : \Pi x{:}\, A_2.\, B_2$.

7. The derivation ends in

$$\frac{\Delta \vDash E : F \qquad \Delta \vDash F' : s' \qquad F =_\beta F'}{\Delta \vDash E : F'}$$

By induction hypothesis $\Gamma_1 \vdash A : B$ and $\Gamma_2 \vdash B' : s'$ where $|\Gamma_1| \equiv \Delta \equiv |\Gamma_2|$, $|A| \equiv E$, $|B| \equiv F$, and $|B'| \equiv F'$. By Lemma 44 (2), $\Gamma_1 =_\beta \Gamma_2$. By context conversion, $\Gamma_2 \vdash A : B$. By Lemma 44(3), $B =_\beta B'$. Hence by conversion $\Gamma_2 \vdash A : B'$.

This concludes the proof. □

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

Note that functionality is used only in Case 4 of the proof. Other properties (e.g. *fullness*) may be assumed instead of functionality. It is also possible to prove a similar result for non-functional PTSs by replacing $\equiv$ by $=_{\underline{\beta}}$ in the statement of the theorem.

We conclude this section by noting that the equational theory of functional, normalizing PTSs is not affected by erasure.

Proposition 46
Let $\mathbf{S}$ be functional with $\mathsf{Type} \subseteq \mathrm{WN}_\beta$. Assume $M, N \in \mathrm{WN}_\beta$, $\Gamma \vdash M : A$, and $\Gamma \vdash N : A$. Then

$$M =_\beta N \quad \Leftrightarrow \quad |M| =_{\underline{\beta}} |N|$$

Proof
The left-to-right implication follows by erasing.

As for the right-to-left implication, assume that $|M| =_{\underline{\beta}} |N|$. Since $M, N \in \mathrm{WN}_\beta$, $|M|, |N| \in \mathrm{WN}_{\underline{\beta}}$ (see the proof of Proposition 49(2)), so let $P \in \mathrm{NF}_{\underline{\beta}}$ be such that $|M| \twoheadrightarrow_{\underline{\beta}}$ and $|N| \twoheadrightarrow_{\underline{\beta}} P$. By Lemma 41, there exists Q and R such that $|Q| \equiv |R| \equiv P$, $M \twoheadrightarrow_\beta Q$ and $N \twoheadrightarrow_\beta R$. By assumption $Q \twoheadrightarrow_\beta Q'$ and $R \twoheadrightarrow_\beta R'$ for some $Q', R' \in \mathrm{NF}_\beta$. Then $|Q'| \equiv |Q| \equiv |R| \equiv |R'|$. By subject reduction $\Gamma \vdash Q' : A$ and $\Gamma \vdash R' : A$. By Lemma 42(1), $Q \equiv R$. Hence $M =_\beta N$. □

The result does not hold for non-normalizing systems.

Proposition 47
Let $\mathbf{S} = *$. There exists $M, N \in \mathscr{E}$ such that $\Gamma \vdash M : A$, and $\Gamma \vdash N : A$, and $|M| =_{\underline{\beta}} |N|$ but not $M =_\beta N$.

Proof
See Hurkens (1995). □

5 Applications

Proposition 39 and Theorem 45 accommodate transfer of results from PTSs to DFPTSs and back. In this section, we study three applications of this idea. In the first subsection we investigate the relationship between normalization in DFPTSs and PTSs. In the second subsection we study the relationship between conservative extensions of DFPTSs and PTSs. In the last subsection we are concerned with looping combinators.

5.1 Normalization

Recall that an expression is weakly normalizing if it has a reduction sequence ending in a normal form, whereas an expression is strongly normalizing if all reduction sequences from the expression eventually end in normal forms; that is, if the expression has no infinite reductions. A (domain-free) Pure Type System is weakly normalizing (resp. strongly normalizing) if all its legal expressions are.

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

The Barendregt–Geuvers–Klop conjecture states that every β-weakly normalizing PTS is also β-strongly normalizing. In order to solve the conjecture, it is natural to develop techniques to infer the latter from the former. Indeed, a variety of such techniques have been invented, most recently by Sørensen and Xi. The main ingredient in Sørensen's (1997) technique is a so-called *Continuation-Passing Style* (CPS) translation. The technique involves certain technical difficulties which render domain-free expressions easier to work with than domain-full ones (see Sørensen (1997) and Barthe *et al.* (1999)). It is therefore natural to investigate under which conditions the implication from weak normalization to strong normalization of a DFPTS can be lifted to the corresponding PTS. This is the purpose of the present subsection.

Definition 48
Let $\mathbf{S}$ be a specification and let (ϕ, T) be $(\lambda, \mathscr{E})$ or $(\underline{\lambda}, \underline{\mathscr{E}})$. Assume $X \subseteq T$. We write $\phi\mathbf{S} \models X$ if every legal $\phi\mathbf{S}$-expression belongs to X.

Proposition 49
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 $\mathsf{Type} \subseteq \mathrm{SN}_\beta$, then $\underline{\lambda}\mathbf{S} \models \mathrm{SN}_{\underline{\beta}}$ implies $\lambda\mathbf{S} \models \mathrm{SN}_\beta$;
4. If $\mathsf{Type} \subseteq \mathrm{WN}_\beta$, then $\underline{\lambda}\mathbf{S} \models \mathrm{WN}_{\underline{\beta}}$ implies $\lambda\mathbf{S} \models \mathrm{WN}_\beta$.

Proof

1. Assume $\lambda\mathbf{S} \models \mathrm{SN}_\beta$ (in particular $\lambda\mathbf{S} \models \mathrm{WN}_\beta$ and hence $\mathsf{Type} \subseteq \mathrm{WN}_\beta$), and let $M \in \underline{\mathscr{E}}$ be legal. By Theorem 45 there is a legal $E \in \mathscr{E}$ such that $|E| \equiv M$. If M had an infinite reduction
$$M \equiv M_0 \to_{\underline{\beta}} M_1 \to_{\underline{\beta}} \ldots$$
then by Lemma 41(1) there would also be an infinite reduction from E:
$$E \equiv E_0 \to_\beta E_1 \to_\beta \ldots$$
where $|E_i| \equiv M_i$. This contradicts the assumptions.
2. Assume $\lambda\mathbf{S} \models \mathrm{WN}_\beta$ (again, in particular, $\mathsf{Type} \subseteq \mathrm{WN}_\beta$), and let $M \in \underline{\mathscr{E}}$ be legal. By Theorem 45 there is a legal $E \in \mathscr{E}$ such that $|E| \equiv M$. By assumption there is a reduction
$$E \equiv E_0 \to_\beta E_1 \to_\beta \ldots \to_\beta E_n \in \mathrm{NF}_\beta$$
Then, by Proposition 39(1),
$$M \equiv |E_0| \twoheadrightarrow_{\underline{\beta}} |E_1| \twoheadrightarrow_{\underline{\beta}} \ldots \twoheadrightarrow_{\underline{\beta}} |E_n|$$
Finally, erasure preserves normal forms, hence $E_n \in \mathrm{NF}_\beta$ implies $|E_n| \in \mathrm{NF}_{\underline{\beta}}$. Thus, $M \in \mathrm{WN}_{\underline{\beta}}$.
3. Assume $\underline{\lambda}\mathbf{S} \models \mathrm{SN}_{\underline{\beta}}$ and that $\mathsf{Type} \subseteq \mathrm{SN}_\beta$, and let $E \in \mathscr{E}$ be legal. If E had an infinite reduction
$$E \equiv E_0 \to_\beta E_1 \to_\beta \ldots$$

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

then by Proposition 39(1) we would also have

$$M \equiv M_0 \twoheadrightarrow_{\underline{\beta}} M_1 \twoheadrightarrow_{\underline{\beta}} \ldots$$

with $|E| \equiv M$ and $|E_i| \equiv M_i$. More specifically, for every reduction $E_i \to_\beta E_{i+1}$ we have $M_i \equiv M_{i+1}$ if the former reduction is inside a domain, and $M_i \to_{\underline{\beta}} M_{i+1}$ otherwise. By the assumptions there cannot be infinitely many consecutive domain-reductions.[8] Thus, we also have an infinite reduction from M.

4. Assume $\underline{\lambda}\mathbf{S} \models \mathrm{WN}_{\underline{\beta}}$ and that $\mathsf{Type} \subseteq \mathrm{WN}_\beta$, and let $E \in \mathscr{E}$ be legal. By Proposition 39(3) $|E| \in \underline{\mathscr{E}}$ is legal, so $|E| \twoheadrightarrow_{\underline{\beta}} N \in \mathrm{NF}_{\underline{\beta}}$ for some N. By Lemma 41(1), there is an $F \in \mathscr{E}$ such that $|F| \equiv N$. By assumption, $F \twoheadrightarrow_\beta G \in \mathrm{NF}_\beta$ by domain-reductions alone. Thus, $E \in \mathrm{WN}_\beta$.

This completes the proof. □

Corollary 50
Let $\mathbf{S}$ be a functional specification with $\mathsf{Type} \subseteq \mathrm{SN}_\beta$. If

$$\underline{\lambda}\mathbf{S} \models \mathrm{WN}_{\underline{\beta}} \Rightarrow \underline{\lambda}\mathbf{S} \models \mathrm{SN}_{\underline{\beta}}$$

then

$$\lambda\mathbf{S} \models \mathrm{WN}_\beta \Rightarrow \lambda\mathbf{S} \models \mathrm{SN}_\beta$$

Remark 51
This shows that if weak normalization implies strong normalization of all legal expressions in $\underline{\lambda}2$ then the same holds for $\lambda 2$. This is also true for the specification C. However, in the latter case the assumption $\mathsf{Type} \subseteq \mathrm{SN}_\beta$ is rather strong since the types in λC may contain terms.

The strong assumption comes from Proposition 49(3) which may be strengthened by considering an extension of DFPTSs with a K-combinator (Barthe, 1995) and reducing strong normalisation of a PTS to that of its corresponding DFPTS with the K-combinator. However the Proposition is already useful as stated, for example, in work on the Barendregt–Geuvers–Klop conjecture (Sørensen, 1997).

Proposition 49 implies strong normalization for the $\underline{\lambda}$-cube:

Proposition 52
$\underline{\lambda}\mathbf{S} \models \mathrm{SN}_{\underline{\beta}}$ for every cube-specification $\mathbf{S}$.

Proof
By Proposition 49(1) and strong normalization of the λ-cube. □

5.2 *Consistency and conservativity*

Next we examine how consistency and conservativity results are reflected. Throughout this subsection $\mathbf{S}$ denotes a functional specification with $\mathsf{Type} \subseteq \mathrm{WN}_\beta$.

[8] Strictly speaking, this conclusion requires an argument. Although $\mathsf{Type} \subseteq \mathrm{WN}_\beta$ it might be that there were an infinite reduction path of domain-reductions inside *different* domains. However, it is easy to prove by induction on M that if $\mathsf{Type} \subseteq \mathrm{WN}_\beta$ then M has no infinite reduction in which all steps are inside domains.

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

Definition 53
Let $s \in \mathscr{S}$. $\underline{\lambda}\mathbf{S}$ is *s-consistent* if there is no $M \in \underline{\mathscr{E}}$ such that $\alpha : s \Vdash M : \alpha$.

The definition for PTSs is analogous.

The following shows that consistency of a PTS and the corresponding DFPTS are equivalent if the specification is functional and $\mathsf{Type} \subseteq \mathrm{WN}_\beta$.

Proposition 54 (*Consistency*)
Let $s \in \mathscr{S}$. $\underline{\lambda}\mathbf{S}$ is s-consistent iff $\lambda\mathbf{S}$ is.

Proof
By Theorem 45 and Proposition 39

$$\exists M \in \mathscr{E}.\ \alpha : s \vdash M : \alpha \quad \Leftrightarrow \quad \exists N \in \underline{\mathscr{E}}.\ \alpha : s \Vdash N : \alpha$$

as required. □

Definition 55
Let $\mathbf{S}' = (\mathscr{S}', \mathscr{A}', \mathscr{R}') \subseteq (\mathscr{S}, \mathscr{A}, \mathscr{R}) = \mathbf{S}$ (component-wise inclusion) and let $s \in \mathscr{S}'$. $\underline{\lambda}\mathbf{S}'$ is *s-conservative over* $\underline{\lambda}\mathbf{S}$ if

$$\Gamma \Vdash_{\mathbf{S}} M : A \quad \& \quad \Gamma \Vdash_{\mathbf{S}'} A : s \quad \Rightarrow \quad \exists N \in \underline{\mathscr{E}}.\ \Gamma \Vdash_{\mathbf{S}'} N : A$$

Informally, the definition states that if some proposition A has a proof M in a DFPTS $\underline{\lambda}\mathbf{S}$, and the proposition is also meaningful (but not necessarily provable) in the smaller DFPTS $\underline{\lambda}\mathbf{S}'$, then in fact, A has a proof N in $\underline{\lambda}\mathbf{S}'$.

The definition for PTSs is analogous.

Proposition 56 (*Conservativity*)
Let $s \in \mathscr{S}'$. Then $\underline{\lambda}\mathbf{S}'$ is s-conservative over $\underline{\lambda}\mathbf{S}$ iff $\lambda\mathbf{S}'$ is s-conservative over $\lambda\mathbf{S}$.

Proof
For the left-to-right implication, assume $\Gamma \vdash_{\mathbf{S}'} A : s$ and $\Gamma \vdash_{\mathbf{S}} M : A$. By Proposition 39, $|\Gamma| \Vdash_{\mathbf{S}'} |A| : s$ and $|\Gamma| \Vdash_{\mathbf{S}} |M| : |A|$. By hypothesis, there exists N such that $|\Gamma| \Vdash_{\mathbf{S}'} N : |A|$. By Theorem 45, there exists Δ, P, B with $\Delta \vdash_{\mathbf{S}'} P : B$, $|\Delta| \equiv |\Gamma|$, and $|B| \equiv |A|$. By Lemma 44, $\Delta =_\beta \Gamma$ and $B =_\beta A$. By Conversion and context conversion, $\Gamma \vdash_{\mathbf{S}'} P : A$.

To prove the reverse implication, assume $\Gamma \Vdash_{\mathbf{S}'} A : s$ and $\Gamma \Vdash_{\mathbf{S}} M : A$. By Theorem 45, $\Delta \vdash_{\mathbf{S}'} B : s$ and $\Xi \vdash_{\mathbf{S}} N : C$ where $|\Delta| \equiv |\Xi| \equiv \Gamma$ and $|B| \equiv |C| \equiv A$. Then also $\Delta \vdash_{\mathbf{S}} B : s$, so by Lemma 44, $\Delta =_\beta \Xi$ and $B =_\beta C$. By subject reduction, $\Phi \vdash_{\mathbf{S}'} D : s$ and $\Phi \vdash_{\mathbf{S}} N : D$ for some D such that $B, C \twoheadrightarrow_\beta D$ and Φ such that $\Delta, \Xi \twoheadrightarrow_\beta \Phi$. By hypothesis, there exists P such that $\Phi \vdash_{\mathbf{S}'} P : D$. By Proposition 39, $|\Phi| \Vdash_{\mathbf{S}'} |P| : |D|$. By Conversion and context conversion, $\Gamma \Vdash_{\mathbf{S}'} |P| : A$. □

The latter result allows us to derive (non-)conservativity results for the $\underline{\lambda}$-cube from Geuvers (1993). For example, one has:

Corollary 57
Let $\mathbf{S}' \subseteq \mathbf{S}$ be cube-specifications. Then $\underline{\lambda}\mathbf{S}'$ is $*$-conservative over $\underline{\lambda}\mathbf{S}$ iff $\mathbf{S}' \neq P2$ and $\mathbf{S} \neq P\omega$.

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

5.3 Looping combinators and fixpoints

Finally, we consider fixpoints and looping combinators. Throughout this section, **S** denotes a fixed specification.

Definition 58

1. A *fixpoint combinator* of sort s is an expression Y such that

$$\vdash Y : (\Pi A{:}s.\,(A \to A) \to A)$$

   satisfying for every A and f,

$$Y\ A\ f =_{\underline{\beta}} f\ (Y\ A\ f)$$

2. A *looping combinator* of sort s is an expression Y for which there exists $Y_0 \equiv Y,\ Y_1, \ldots$ such that for every i

$$\vdash Y_i : (\Pi A{:}s.\,(A \to A) \to A)$$

   and for every A and f,

$$Y_i\ A\ f =_{\underline{\beta}} f\ (Y_{i+1}\ A\ f)$$

The definition for PTSs is analogous.

Proposition 59

1. If $\lambda\mathbf{S}$ has a looping combinator, then so has $\underline{\lambda}\mathbf{S}$.
2. If $\lambda\mathbf{S}$ has a fixpoint combinator, then so has $\underline{\lambda}\mathbf{S}$.

Proof
By Proposition 39. □

Coquand and Herbelin (1994) study looping combinators for pure type systems (see also Geuvers and Werner (1994) for some results about fixpoints in Pure Type Systems with $\beta\eta$-conversion). They show that every so-called inconsistent logical non-dependent pure type systems has a looping combinator.

Corollary 60
$\underline{\lambda}U^-$, $\underline{\lambda}U$ and $\underline{\lambda}*$ have a looping combinator of sort $*$ where

1. $U^- = (\mathscr{S}, \mathscr{A}, \mathscr{R})$ and
   - $\mathscr{S} = \{*, \Box, \triangle\}$;
   - $\mathscr{A} = \{(*, \Box), (\Box, \triangle)\}$;
   - $\mathscr{R} = \{(*, *), (\Box, \Box), (\Box, *), (\triangle, \Box)\}$.
2. U arises from U^- by addition of the rule $(\triangle, *)$;
3. $* = (\{*\}, \{(*, *)\}, \{(*, *, *)\})$.

Proof
In Coquand and Herbelin (1994), it is shown that λU^-, λU and $\lambda *$ have a looping combinator. We apply Proposition 59. □

It would be interesting to investigate whether the above result can be strengthened for $\underline{\lambda}U^-$ and $\underline{\lambda}*$.

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

Open problem 61
Do $\underline{\lambda}U^-$ and $\underline{\lambda}*$ have a fixpoint combinator?

One possible approach to solve this problem is to construct with the techniques (Coquand and Herbelin, 1994) a domain-full looping combinator Y_n from the domain-full term proving inconsistency of λU^- in Hurkens (1995) and to check whether the erasure of the looping combinator is indeed a fixpoint combinator for $\underline{\lambda}U^-$.

Note that a positive answer to the above question would leave open the related question of a fixpoint combinator in $\lambda*$.

6 Domain-free pure type systems vs type assignment systems

In recent work, van Bakel, Liquori, Ronchi della Roncha and Urzyczyn (1994) define for each cube-specification $\mathbf{S}$ a *type assignment system* $\overline{\lambda}\mathbf{S}$. These systems, which form what we call the $\overline{\lambda}$*-cube*, include simple types $\overline{\lambda}\!\rightarrow$ introduced by Curry (1934), second-order types $\overline{\lambda}2$ introduced by Leivant (1983) and higher-order types $\overline{\lambda}\omega$ introduced by Giannini and Ronchi della Rocca (1988). In this section, we study the relationship between between the $\overline{\lambda}$-cube, the $\underline{\lambda}$-cube, and the λ-cube.

The first subsection introduces the $\overline{\lambda}$-cube and the second subsection investigates its relationship with the two other cubes. To ease the comparison between cubes, we use sorted variables and the syntaxes of the classification lemmas (Definitions 27 and 36).

6.1 The $\overline{\lambda}$-cube

An important aspect of the $\overline{\lambda}$-cube is the distinction between three different syntactic categories: objects, constructors and kinds. Each category uses its own form of λ-abstraction: objects use domain-free λ-abstractions, whereas constructors use domain-full λ-abstractions (there is no notion of λ-abstraction for kinds).

Definition 62
Let $\mathbf{S} = (\mathscr{S}, \mathscr{A}, \mathscr{R})$ be a cube specification.

1. Let V^* and $V^\Box$ be denumerable, disjoint sets of variables. Define the set $\overline{\mathscr{E}}$ of *(type assignment) expressions* by $\overline{\mathscr{E}} = \overline{\mathscr{O}} \cup \overline{\mathscr{C}} \cup \overline{\mathscr{K}} \cup \{\Box\}$, where *objects* $\overline{\mathscr{O}}$, *constructors* $\overline{\mathscr{C}}$, and *kinds* $\overline{\mathscr{K}}$ are defined by:

$$\begin{array}{rcl} \overline{\mathscr{O}} & = & V^* \mid \lambda V^* . \overline{\mathscr{O}} \mid \overline{\mathscr{O}}\,\overline{\mathscr{O}} \\ \overline{\mathscr{C}} & = & V^\Box \mid \Pi V^* : \overline{\mathscr{C}} . \overline{\mathscr{C}} \mid \Pi V^\Box : \overline{\mathscr{K}} . \overline{\mathscr{C}} \mid \lambda V^* : \overline{\mathscr{C}} . \overline{\mathscr{C}} \mid \lambda V^\Box : \overline{\mathscr{K}} . \overline{\mathscr{C}} \mid \overline{\mathscr{C}}\,\overline{\mathscr{C}} \mid \overline{\mathscr{C}}\,\overline{\mathscr{O}} \\ \overline{\mathscr{K}} & = & * \mid \Pi V^* : \overline{\mathscr{C}} . \overline{\mathscr{K}} \mid \Pi V^\Box : \overline{\mathscr{K}} . \overline{\mathscr{K}} \end{array}$$

We use M, N to denote elements of $\overline{\mathscr{E}}$; ϕ, ψ to denote elements of $\overline{\mathscr{C}}$; K, K' to denote elements of $\overline{\mathscr{K}}$; x, y to denote elements of V^*; and α, β to denote elements of $V^\Box$. We use A, B, C to range over arbitrary elements of $\overline{\mathscr{E}}$ and u, v, w to range over $V = V^* \cup V^\Box$. As usual, s, s' denote elements of $\mathscr{S}$. We assume the reader is familiar with the notions of free and bound variables and

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

$$\frac{\Gamma \Vdash A : s}{\Gamma, v{:}\, A \Vdash v : A} \text{ if } v \notin \mathrm{dom}(\Gamma) \qquad \frac{\Gamma \Vdash A : B \quad \Gamma \Vdash C : s}{\Gamma, v{:}\, C \Vdash A : B} \text{ if } v \notin \mathrm{dom}(\Gamma)$$

$$\frac{\Gamma \Vdash A : B \quad \Gamma \Vdash B' : s}{\Gamma \Vdash A : B'} \text{ if } B =_{\overline{\beta}} B' \qquad \langle\rangle \Vdash * : \Box$$

$$\frac{\Gamma, x{:}\, \phi \Vdash M : \psi}{\Gamma \Vdash \lambda x . M : \Pi x{:}\, \phi . \psi} \text{ if } (*,*) \in \mathscr{R} \qquad \frac{\Gamma \Vdash M : (\Pi x{:}\, \phi . \psi) \quad \Gamma \Vdash N : \phi}{\Gamma \Vdash M\, N : \psi \langle x := N \rangle} \text{ if } (*,*) \in \mathscr{R}$$

$$\frac{\Gamma, \alpha{:}\, K \Vdash M : \psi}{\Gamma \Vdash M : \Pi \alpha{:}\, K . \psi} \text{ if } (\Box,*) \in \mathscr{R} \qquad \frac{\Gamma \Vdash M : (\Pi \alpha{:}\, K . \psi) \quad \Gamma \Vdash \phi : K}{\Gamma \Vdash M : \psi \langle \alpha := \phi \rangle} \text{ if } (\Box,*) \in \mathscr{R}$$

$$\frac{\Gamma, x{:}\, \phi \Vdash \psi : K}{\Gamma \Vdash \lambda x : \phi . \psi : \Pi x{:}\, \phi . K} \text{ if } (*,\Box) \in \mathscr{R} \qquad \frac{\Gamma \Vdash \psi : (\Pi x{:}\, \phi . K) \quad \Gamma \Vdash N : \phi}{\Gamma \Vdash \psi\, N : K \langle x := N \rangle} \text{ if } (*,\Box) \in \mathscr{R}$$

$$\frac{\Gamma, \alpha{:}\, K \Vdash \psi : K'}{\Gamma \Vdash \lambda \alpha : K . \psi : \Pi \alpha{:}\, K . K'} \text{if } (\Box,\Box) \in \mathscr{R} \qquad \frac{\Gamma \Vdash \phi : (\Pi \alpha{:}\, K . K') \quad \Gamma \Vdash \psi : K}{\Gamma \Vdash \phi\, \psi : K' \langle \alpha := \psi \rangle} \text{if } (\Box,\Box) \in \mathscr{R}$$

$$\frac{\Gamma, x{:}\, \phi \Vdash \psi : *}{\Gamma \Vdash (\Pi x{:}\, \phi . \psi) : *} \text{ if } (*,*) \in \mathscr{R} \qquad \frac{\Gamma, \alpha{:}\, K \Vdash \psi : *}{\Gamma \Vdash (\Pi \alpha{:}\, K . \psi) : *} \text{ if } (\Box,*) \in \mathscr{R}$$

$$\frac{\Gamma, x{:}\, \phi \Vdash K : \Box}{\Gamma \Vdash (\Pi x{:}\, \phi . K) : \Box} \text{ if } (*,\Box) \in \mathscr{R} \qquad \frac{\Gamma, \alpha{:}\, K \Vdash K' : \Box}{\Gamma \Vdash (\Pi \alpha{:}\, K . K') : \Box} \text{ if } (\Box,\Box) \in \mathscr{R}$$

Fig. 6. Type assignment systems.

related conventions; $\mathrm{FV}(M)$ denotes the set of variables occurring free in M, and $\equiv$ denotes syntactic equality (see Barendregt, 1992).

2. A *(type assignment) context* is a finite sequence of form $v_1{:}\, A_1, \ldots, v_n{:}\, A_n$; the empty sequence is written $\langle\rangle$. The set of all contexts is called $\overline{\mathscr{G}}$. We write $\mathrm{dom}(v_1{:}\, A_1, \ldots, v_n{:}\, A_n) = \{v_1, \ldots, v_n\}$ and use the same naming conventions as for domain-free contexts.
3. *$\overline{\beta}$-reduction* $\to_{\overline{\beta}}$ on $\overline{\mathscr{E}}$ is defined as the compatible closure of the contractions

$$\begin{aligned} (\lambda x : A.M)\ N &\to_{\overline{\beta}} M\langle x := N\rangle \\ (\lambda x.M)\ N &\to_{\overline{\beta}} M\langle x := N\rangle \end{aligned}$$

   where $\bullet\langle \bullet := \bullet \rangle$ is the obvious substitution operator. *$\overline{\beta}$-equality* $=_{\overline{\beta}}$ is the reflexive, transitive, symmetric closure of $\to_{\overline{\beta}}$.
4. The *derivability* relation $\Vdash$ is given by the rules of Figure 6. If $\Gamma \Vdash A : B$ then Γ, A and B are legal.
5. The tuple $\overline{\lambda}\mathbf{S} = (\overline{\mathscr{E}}, \overline{\mathscr{G}}, =_{\overline{\beta}}, \Vdash)$ is the *Type Assignment System (*TAS*) induced by* $\mathbf{S}$.

6.2 The $\overline{\lambda}$-cube versus the λ-cube and the $\underline{\lambda}$-cube

We have already considered the erasing function $|\bullet|$ from domain-full expressions to domain-free expressions. Incidentally, $|\bullet|$ maps elements of $\mathscr{E}'$ to elements of $\underline{\mathscr{E}}'$; in fact, $|\bullet|$ maps elements of $\mathscr{O}, \mathscr{C}, \mathscr{K}, \{\Box\}$ to elements of $\underline{\mathscr{O}}, \underline{\mathscr{C}}, \underline{\mathscr{K}}, \{\Box\}$, respectively.

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

$$\text{Objects } G : \overline{\mathscr{O}} \to \underline{\mathscr{O}}$$

$$\begin{array}{rcl} G(x) & = & x \\ G(\lambda x\,.\,M) & = & \lambda x\,.\,G(M) \\ G(M\ N) & = & G(M)\ G(N) \end{array}$$

$$\text{Constructors } G : \overline{\mathscr{C}} \to \underline{\mathscr{C}}$$

$$\begin{array}{rcl} G(\alpha) & = & \alpha \\ G(\lambda v : A\,.\,\phi) & = & \lambda v\,.\,G(\phi) \\ G(\phi\ A) & = & G(\phi)\ G(A) \\ G(\Pi v\colon A.\ \phi) & = & \Pi v\colon G(A).\ G(\phi) \end{array}$$

$$\text{Kinds } G : \overline{\mathscr{K}} \to \underline{\mathscr{K}}$$

$$\begin{array}{rcl} G(*) & = & * \\ G(\Pi x\colon A.\ K) & = & \Pi x\colon G(A).\ G(K) \end{array}$$

$$\text{Box } G : \{\Box\} \to \{\Box\}$$

$$G(\Box) \quad = \quad \Box$$

Fig. 7. Erasure from type-assignment to domain-free expressions.

$$\text{Objects } E : \mathscr{O} \to \overline{\mathscr{O}}$$

$$\begin{array}{rcl} E(x) & = & x \\ E(\lambda x : \phi\,.\,M) & = & \lambda x\,.\,E(M) \\ E(\lambda \alpha : K\,.\,M) & = & E(M) \\ E(M\ N) & = & E(M)\ E(N) \\ E(M\ \phi) & = & E(M) \end{array}$$

$$\text{Constructors } E : \mathscr{C} \to \overline{\mathscr{C}}$$

$$\begin{array}{rcl} E(\alpha) & = & \alpha \\ E(\lambda v : A\,.\,\phi) & = & \lambda v : E(A)\,.\,E(\phi) \\ E(\phi\ A) & = & E(\phi)\ E(A) \\ E(\Pi v\colon A.\ \phi) & = & \Pi v\colon E(A).\ E(\phi) \end{array}$$

$$\text{Kinds } E : \mathscr{K} \to \overline{\mathscr{K}}$$

$$\begin{array}{rcl} E(*) & = & * \\ E(\Pi x\colon A.\ K) & = & \Pi x\colon E(A).\ E(K) \end{array}$$

$$\text{Box } E : \{\Box\} \to \{\Box\}$$

$$E(\Box) \quad = \quad \Box$$

Fig. 8. Erasure from domain-full to type-assignment expressions.

In this subsection we introduce two more erasing functions, E from domain-full expressions to type assignment expressions and G from type assignment expressions to domain-free expressions.

Definition 63

1. The erasure map $G : \overline{\mathscr{E}} \to \underline{\mathscr{E}'}$ is defined in Figure 7.
2. The erasure map $E : \mathscr{E}' \to \overline{\mathscr{E}}$ is defined in Figure 8.

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

The first erasure map preserves derivability for specifications **S** without polymorphism, i.e. for $\rightarrow$, P, $\underline{\omega}$, and $P\underline{\omega}$.

Theorem 64
Let **S** be a specification without polymorphism.

1. If $\Gamma \Vdash M : A$, then $G(\Gamma) \vdash G(M) : G(A)$.
2. If $\Gamma \vdash M : A$, then $\Delta \Vdash N : B$ for some Δ, N, B with $G(\Delta) \equiv \Gamma$, $G(N) \equiv M$ and $G(B) \equiv A$.

Proof
1. By induction on the structure of derivations.
2. Using the classification lemma, prove by induction on expressions that $|Q| \equiv G(E(Q))$ for every $Q \in \mathscr{O} \cup \mathscr{C} \cup \mathscr{K} \cup \{\Box\}$. By Theorem 45, there exists a derivable PTS-judgment $\Delta_0 \vdash N_0 : B_0$ such that $|\Delta_0| \equiv \Gamma$, $|N_0| \equiv M$ and $|B_0| \equiv A$. Let $\Delta \equiv E(\Delta_0)$, $N \equiv E(N_0)$ and $B \equiv E(B_0)$. It follows from (van Bakel *et al.*, 1994) that $\Delta \Vdash N : B$. Moreover, $G(\Delta) \equiv \Gamma$, $G(N) \equiv M$ and $G(B) \equiv A$ since $|Q| \equiv G(E(Q))$.

This concludes the proof. $\square$

Corollary 65
TC, TS and TY are decidable for $\bar{\lambda}\underline{\omega}$.

Proof
We only treat TC since TS and TY are treated in a similar way. The interesting case is when the judgment is of the form $\Gamma \Vdash M : A$ with $M \in \overline{\mathscr{O}}$ (if $M \in \overline{\mathscr{C}}$, then the judgment is trivially decidable). We have $\Gamma \Vdash M : A$ iff $\Gamma \Vdash A : *$ and $G(\Gamma) \Vdash M : G(A)$. Each conjunct is decidable (the first one trivially, the second one by Theorem 32(3)), so TC is decidable. $\square$

For specifications with polymorphism, G does not preserve typing. Indeed, the judgment $\Vdash \lambda x.x : \Pi\alpha{:}*.\,\alpha \rightarrow \alpha$, which is derivable in $\bar{\lambda}2$, is not derivable in $\underline{\lambda}2$. On the other hand, one can define for the systems of the domain-free λ-cube an erasure map from domain-free objects to type assignment objects.

Definition 66
The erasure map $F : \underline{\mathscr{O}} \rightarrow \overline{\mathscr{O}}$ is defined as follows:

$$\begin{aligned} F(x) &= x \\ F(\lambda x.M) &= \lambda x.F(M) \\ F(\lambda \alpha.M) &= F(M) \\ F(M\,N) &= F(M)\,F(N) \\ F(M\,\phi) &= F(M) \end{aligned}$$

The erasure map F preserves and reflects typing for the specification 2.

Theorem 67
Let $\mathbf{S} = 2$ and let $M \in \underline{\mathscr{O}}$.

1. If $\Gamma \vdash M : A$, then $\Gamma \Vdash F(M) : A$.

https://doi.org/10.1017/S0956796800003750 Published online by Cambridge University Press

2. If $\Gamma \Vdash M : A$, then there exists a derivable judgment $\Gamma \vdash N : A$ such that $F(N) \equiv M$.

Proof
Note that legal constructors in $\underline{\lambda}2$ are legal constructors in $\overline{\lambda}2$, and vice versa, and that the only legal kind in $\underline{\lambda}2$ and $\overline{\lambda}2$ is $*$.

1. By induction on the structure of derivations.
2. Prove by induction on the definition of $\mathcal{O}$ that $E(Q) \equiv F(|Q|)$ for every $Q \in \mathcal{O}$. Then assume that $\Gamma \Vdash M : A$. By van Bakel *et al.* (1994), there exists a derivable PTS-judgment $\Gamma \vdash N : A$ such that $E(N) \equiv M$. Let $|N| \equiv P$. It follows that $\Gamma \vdash P : A$ with $F(P) \equiv M$.

This concludes the proof. □

It is not clear how to define an erasure map between $\underline{\lambda}\omega$ and $\overline{\lambda}\omega$ or between $\underline{\lambda}C$ and $\overline{\lambda}C$.

7 Conclusion

We have introduced the notion of a domain-free pure type system, developed its basic properties, established its exact relationship with the notion of pure type system and with the notion of type assignment system, and used the former correspondence to study a number of applications. Despite failing to have decidable type-checking, domain-free pure type systems provide an attractive alternative to pure type systems and have been used in several theoretical studies (Barthe *et al.*, 1999; Barthe *et al.*, 1997; Sørensen, 1997) and implementations (Magnusson, 1994; Pfenning, 1994).

Acknowledgements

The authors are grateful to J. Hatcliff and T. Coquand for numerous discussions on the domain-free pure type systems, to R. Pollack for discussions on type-checking for domain-free pure type systems, and to J. Wells for suggesting to use Schubert's result to prove the undecidability of type-checking for $\underline{\lambda}2$. Finally, we are indebted to B. Werner and the anonymous referees for their constructive suggestions.

Part of this work was performed while the first author was working at CWI (Amsterdam, The Netherlands) and at Chalmers University (Göteborg, Sweden). The first author was partially supported by a European TMR Fellowship.

References

Appel, A. (1992) *Compiling with Continuations.* Cambridge University Press.

Bakel, S. van, Liquori, L., Ronchi della Roncha, S. and Urzyczyn, P. (1994) Comparing cubes. In: Nerode, A. and Matiyasevich, Y. N. (eds.), *Proceedings of LFCS'94: Lecture Notes in Computer Science*, **813**, pp. 353–365. Springer-Verlag.

Barendregt, H. (1992) Lambda calculi with types. In: Abramsky, S., Gabbay, D. M. and Maibaum, T.S.E. (eds.), *Handbook of Logic in Computer Science, Vol. 2*, pp. 117–309. Oxford Science Publications.

Barthe, G. (1995) Extensions of pure type systems. In: Dezani-Ciancaglini, M. and Plotkin, G. (eds.), *Proceedings of TLCA'95: Lecture Notes in Computer Science*, **902**. Springer-Verlag.

Barthe, G., Hatcliff, J. and Sørensen, M. H. (1999) CPS-translation and applications: the cube and beyond. *Higher-Order and Symbolic Computation*, **12**(2), 125–170.

Barthe, G., Hatcliff, J. and Sørensen, M.H. (1997) A notion of classical pure type system. In: Brookes, S. and Mislove, M. (eds.), *Proceedings of MFPS'97. Electronic Notes in Theoretical Computer Science*, **6**.

Barthe, G. and Sørensen, M. H. (1997) Domain-free Pure Type Systems. In: Adian, S. and Nerode, A. (eds.), *Logical Foundations of Computer Science: Lecture Notes in Computer Science*, **1234**, pp. 9–20. Springer-Verlag.

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

Betarte, G. and Tasistro, A. (1998) Extension of Martin-Löf's theory of types with record types and subtyping: motivation, rules and type checking. In: Sambin, G. and Smith, J. (eds.), *Twenty-five Years of Constructive Type Theory*. Oxford University Press.

Church, A. (1940) A formulation of the simple theory of types. *J. Symbolic Logic*, **5**, 56–68.

Coquand, T. and Herbelin, H. (1994) A-translation and looping combinators in pure type systems. *J. Functional Programming*, **4**(1), 77–88.

Curry, H. (1934) Functionality in combinatory logic. *Proceedings of the National Academy of Science USA*, **20**, 584–590.

Curry, H. and Feys, R. (1958) *Combinatory Logic*. North-Holland.

Dezani-Ciancaglini, M. and Plotkin, G. (eds). (1995) *Proceedings of TLCA'95: Lecture Notes in Computer Science*, **902**. Springer-Verlag.

Dowek, G. (1993) The undecidability of typability in the Lambda-Pi-Calculus. In: Bezem, M. and Groote, J. F. (eds.), *Proceedings of TLCA'93: Lecture Notes in Computer Science*, **664**, pp. 139–145. Springer-Verlag.

Fujita, K.-E. (1999) Type Inference for Domain-Free $\lambda 2$. Technical Report, Department of Computer Science and Systems Engineering CSSE-5, Kyushu Institute of Technology, 1999.

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

Geuvers, J. H. and Nederhof, M. J. (1991) A modular proof of strong normalization for the calculus of constructions. *J. Functional Programming*, **1**(2), 155–189.

Geuvers, H. and Werner, B. (1994) On the Church-Rosser property for expressive type systems and its consequence for their metatheoretic study *Proceedings of LICS'94*, pp. 320–329. IEEE Press.

Giannini, P. and Ronchi Della Rocca, S. (1988) Characterization of typings in polymorphic type discipline. *Proceedings of LICS'88*, pp. 61–70. IEEE Press.

Girard, J.-Y. (1970) Une extension du système de fonctionelles recursives de Gödel et son application aux fondements de l'analyse. In: Fenstad, J. E. (ed.), *The 2nd Scandinavian Logical Symposium*, pp. 63–92. North-Holland.

Girard, J.-Y. (1972) *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris VII.

Hurkens, A. (1995) A simplification of Girard's Paradox. In: Dezani-Ciancaglini, M. and Plotkin, G. (eds.), *Proceedings of TLCA'95: Lecture Notes in Computer Science*, **902**. Springer-Verlag.

Klop, J.W., van Oostrom, V. and van Raamsdonk, F. (1993) Combinatory reduction systems: Introduction and survey. *Theoretical Computer Science*, **121**(1–2), 279–308.

Leivant, D. (1983) Polymorphic type inference. *Proceedings of POPL'83*, pp. 88–98. ACM Press.

Luo, Z. (1994) *Computation and Reasoning: A Type Theory for Computer Science*. International Series of Monographs on Computer Science, Oxford University Press.

Magnusson, L. (1994) *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.

Pfenning, F. (1994) Elf: a meta-language for deductive systems. In: Bundy, A. (ed.), *Proceedings of CADE-12: Lecture Notes in Artificial Intelligence*, **814**, pp. 811–815. Springer-Verlag.

Reynolds, J. (1974) Towards a theory of type structure. In: Robinet, B. (ed.), *Proceedings of the programming symposium: Lecture Notes in Computer Science*, **19**, pp. 408–425. Springer-Verlag.

Schubert, A. (1998) Second-order unification and type inference for Church-style polymorphism. *Proceedings of POPL'98*, pp. 279–288. ACM Press.

Sørensen, M. H. (1997) Strong normalization from weak normalization in typed λ-calculi. *Information & Computation*, **133**(1), 35–71.

Tasistro, A. (1997) *Substitution, Record Types and Subtyping in Type Theory, with applications to the theory of programming.* PhD thesis, Department of Computer Science, Chalmers Tekniska Högskola.

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