A Notion of Classical Pure Type System (Preliminary version)

Gilles Barthe

CWI, PO Box 94079, 1090 GB Amsterdam, The Netherlands, gilles@cwi.nl

John Hatcliff

Oklahoma State University, Department of Computer Science, 219 Math Sciences, Stillwater, OK, USA, 74078, hatcliff@a.cs.okstate.edu

Morten Heine Sørensen

DIKU, Universitetsparken 1, DK-2100 Copenhagen, Denmark, rambo@diku.dk

Abstract

We present a notion of classical pure type system, which extends the formalism of pure type system with a double negation operator.

1 Introduction

It is an old idea that proofs in formal logics are certain functions and objects. The Brower-Heyting-Kolmogorov (BHK) interpretation [15,51,40], in the form stated by Heyting [40], states that a proof of an implication $P \to Q$ is a "construction" which transforms any proof of P into a proof of Q. This idea was formalized independently by Kleene's realizability interpretation [46,47] in which proofs of intuitionistic number theory are interpreted as numbers, by the *Curry-Howard (CH) isomorphism* [21,43] in which proofs of intuitionistic implicational propositional logic are interpreted as simply typed λ-terms, and by the Lambek-Lawvere (LL) isomorphism [52,55] in which proofs of intuitionistic positive propositional logic are interpreted as morphisms in a cartesian closed category. In the latter cases, the interpretations have an inverse, in that every simply typed λ-term or morphism in a cartesian closed category may be interpreted as a proof; hence the name "isomorphism". Moreover, both interpretations preserve the semantics of the deductive systems, in that proof normalization in logic corresponds exactly to β-reduction in λ-calculus

[78,98] and proof equivalence in logic corresponds exactly to equality between morphisms in a cartesian closed category [53,59]. [1]

The Curry-Howard and Lambek-Lawvere isomorphisms have come to play an important role in the area of logic and computation. Both have been generalized to systems of an increasing complexity –see e.g. [6,25,30,60,67,94] for the Curry-Howard isomorphism and [44,74,75,92] for the Lambek-Lawvere isomorphism– and have been used in a large number of applications. For example, the Curry-Howard isomorphism has been exploited in the use of type theory as a framework for reasoning and computation [16,17,56,68] and in the design of proof-development systems [8,19,33,57,58,73,89]. Yet and quite significantly, both isomorphisms have until the late 1980's invariably been studied in relation with intuitionistic logics. [2]

At that time Griffin [34] realized that Felleisen's *control operator* C [28,29] could be meaningfully added to the simply typed λ-calculus by typing C with the double negation rule [81,87,90] –hereafter we refer to Felleisen's calculus as the λC-calculus and to Griffin's system as the simply typed λC-calculus. Moreover, Griffin showed that the reduction rules for C were closely related to classical proof normalization as studied by Prawitz [81], Seldin [86,87], and Stålmarck [90]. Griffin's discoveries were followed by a series of papers on classical logic, control operators and the Curry-Howard isomorphism, see for example [3,4,18,24,42,62–66,69–72,82]. Most of these works introduce one typed *classical λ-calculus*, i.e. a typed λ-calculus enriched with control operators, and study its properties with respect to e.g. normalization, confluence and categorical semantics or its applications to e.g. classical theorem proving and witness extraction. However, none of the typed classical λ-calculi proposed so far seems to have achieved a status of universality similar to that of the ordinary typed λ-calculus and the question of finding "the classical typed λ-calculus" still remains an area of active investigation.

In a different line of work, some works considered generalizing classical λ-calculi to more powerful systems such as polymorphic λ-calculus or higher-order λ-calculus. Remarkably, this question has so far only been addressed in a few specific cases, e.g. for the second-order type assignment system by Parigot [71,72], ML by Duba, Harper and MacQueen [27] and later by Harper and Lillibridge [38] and Girard's higher-order polymorphic λ-calculus by Harper and Lillibridge [37]. [3] Nevertheless, the central claim of this paper is that

---

[1] There is no notion of reduction associated to a cartesian closed category. Hence the LL isomorphism only reflects the notion of proof equivalence. In order to reflect the notion of proof normalization, the notion of 2-category must be considered [85].

[2] The skepticism towards a proof or categorical semantics of classical logic may be attributed to a number of factors, two of which are mentioned below. In category theory, Joyal noticed that every bicartesian closed category in which A is isomorphic to $\neg\neg A$ is degenerate [54]. In classical proof theory, Girard noticed that cut-elimination in classical sequent calculus was not confluent; different strategies for reduction gives different proofs in normal form, i.e. proof normalization involves an element of non-determinism [32].

[3] Other calculi which have been considered in the literature include PCF [70], linear λ-

generalizing existing classical λ-calculi to complex type systems is of definite interest.

(i) From a theoretical point of view, such an endeavour confronts the existing classical λ-calculi to the issue of uniformity. As a result, the analysis of a classical λ-calculus is not hindered by any endemic feature of a specific type structure. One immediate advantage of such an analysis is to discriminate between *endemic* classical λ-calculi, which are only meaningful for a specific type discipline, and *global* classical λ-calculi, which are meaningful for an arbitrary type discipline.

(ii) From a practical point of view, such an endeavour allows to generalize the Curry-Howard isomorphism to powerful logics, such as classical higher-order predicate logic, and lays the foundations for the design of proof-development systems with a *computationally meaningful* classical operator.

This paper presents a uniform framework for classical λ-calculi. The central notion of this paper, *classical pure type system* (CPTS), is based on the notion of pure type system [6,30,31] and offers a uniform formalism to define and study classical λ-calculi. The formalism is minimal –e.g. its only type constructor is the generalized function space Π– and yet allows for many interesting observations. In particular, it may be used to study –for the first time it seems– dependently typed classical λ-calculi such as the Classical Calculus of Constructions.

Overview of the paper

- *Section 2: Preliminaries*

  This section presents computational type systems (CTS), an hybrid notion combining features of higher-order rewriting systems and type theories. Furthermore, we formulate in the framework of CTSs standard definitions stemming from the areas of higher-order rewriting [7,49] and type theory [6,56].

  The notion of computational type system is introduced solely for its convenience; although it covers the type systems considered in this paper, it is not intended as a general framework for type theories.

- *Section 3: A notion of classical pure type system*

  This section introduces the central notion of this paper, namely that of a classical pure type system. Classical pure type systems are introduced in a similar spirit as pure type systems. The first subsection is devoted to the definition of logical specification. Our definition is closely related, although not identical, to that of Coquand and Herbelin [20]. In the second

---

calculus [14], λ-calculus with explicit substitutions [39,84]. Werner also considered –in unpublished work, 1992– a classical variant of non-dependent logical pure type systems; however his notion of reduction is extremely weak.

Framework	Type Theory	Reduction
Pure Type System	λS	β
Classical Pure Type System	$\lambda \Delta S$	$\beta \Delta$
Domain-Free Pure Type System	$\underline{\lambda S}$	$\underline{\beta}$
Domain-Free Classical Pure Type System	$\underline{\lambda \Delta S}$	$\underline{\beta \Delta}$

Fig. 1. BASIC NOTATION FOR TYPE THEORIES

subsection, we specify some important classes of specifications that appear in the literature and are considered further in the paper. In the third subsection, we introduce the λ-cube [6] and the L-cube [30] as examples of specifications. These examples will be studied in some depth and will serve as running examples throughout the paper. In the fourth subsection, we define the notion of a classical pure type system: in a nutshell, a classical pure type system is a pure type system extended with a binding double negation construction $\Delta x{:}A.\, M$. Our notion is inspired by classical natural deduction as studied by Prawitz [80,81], Seldin [87] and Stålmarck [90]. In order to present the Continuation-Passing Style translation –see Section 5– and analyze its behavior in a typed setting, it is convenient to introduce a variant of classical pure type systems, called *domain-free* classical pure type systems (DFCPTS), in which abstractions do not carry domain tags, i.e. are of the form $\lambda x.M$ and $\Delta x.\, M$. This is done in the fifth subsection. The sixth and last subsection reviews the notions of pure type system (PTS) and domain-free pure type system (DFPTS). The purpose of this last subsection is mainly to fix terminology; properties of pure type systems and domain-free pure type systems may be found respectively in [6,30] and [11].

Remark 1.1 *We consider four different type-theoretic frameworks: pure type systems, classical pure type systems, domain-free pure type systems and domain-free classical pure type systems. Every specification S defines one type theory in each framework. The basic notation is given in Figure 1.*

- *Section 4: Basic properties of classical pure type systems*

    The notion of CPTS provides a general framework to define and study classical typed λ-calculi. This section establishes some basic syntactic properties of CPTSs: closure of derivations under substitution, subject reduction, uniqueness of types, decidability of type-checking... Besides we compare CPTSs with DFCPTSs. In particular, we define an erasure map, which removes the domains of λ- and Δ-abstractions, and prove that erasure preserves derivability for functional specifications. Moreover, we also exhibit some anomalies of erasure with respect to Δ-reduction. We isolate one specific anomaly, which we call the unorthodox behavior of Δ-reduction, which

will cause the failure of syntactic techniques to prove strong normalization for CPTSs –see Section 6.

• *Section 5: Continuation-Passing Style Translation*
The Continuation-Passing Style (CPS) translation is a standard compilation technique for functional languages [2,22,83]. CPS translations were studied among others by Plotkin [76], who defined a CPS translation from type-free λ-terms to type-free λ-terms, and by Felleisen *et al.* [29], who extended Plotkin's translation from type-free λC-terms to type-free λ-terms. Generalizing an earlier observation by Meyer and Wand [61] that Plotkin's CPS translation maps simply typed λ-terms to simply typed λ-terms, Griffin [34] noted that Felleisen's extended translation maps simply typed λC-terms to simply typed λ-terms. He further showed that this translation, when viewed as a translation on proofs, becomes the Kolmogorov embedding of classical logic into minimal logic [50]. Murthy [18,62,63] and Griffin himself [35] later systematized these ideas by studying different logical embeddings, control operators, and CPS translations. More recently, CPS translations from classical typed λ-calculi to typed λ-calculi were studied by de Groote for $\lambda\mu$ [23] and $\lambda^{\rightarrow}_{exn}$ [24], by Duba, Harper and MacQueen for the monomorphic fragment of ML (MML) [27] and by Harper and Lillibridge for ML with polymorphism (PML) [38] and Girard's higher-order polymorphic λ-calculus with control operators [37].

In the first subsection, we discuss some of the problems related to CPS translations for proof-relevant systems. Much of the discussion is taken from [10] where the authors develop CPS translations for logical PTSs and DFPTSs.

In the second subsection, we define for every injective logical specification **S** a CPS translation from $\lambda\Delta S$ to λS; the translation is inspired from [10].

In the third subsection, we use the correctness of the CPS translation to derive the consistency of a CPTS from that of its corresponding PTS. Then we generalize our technique to prove consistency of classical arithmetic from consistency of constructive arithmetic. It is possible to go beyond this specific example and develop some general results for establishing the consistency of contexts but, for the sake of brevity, we do not follow this path.

• *Section 6: Strong normalization*
As a further application of the CPS translation, Griffin [34] used the translation to infer weak normalization of simply typed λC-terms from strong normalization of simply typed λ-terms. Analogous results were later obtained using a similar argument by de Groote for Parigot's $\lambda\mu$-calculus [23], by Duba, Harper and MacQueen for MML [27], and by Harper and Lillibridge for PML [38] and higher-order λ-calculus [37]. Other authors were also able to deduce strong normalization of a classical λ-calculus from strong normalization of simply typed λ-calculus: Rehof and Sørensen for λ_Δ [82] and de Groote for $\lambda^{\rightarrow}_{exn}$ [24].

5

In the first part of this section, we use the CPS translation to derive a normalization result for DFCPTSs. More precisely, we prove that a DFCPTS is $\beta\Delta$-strongly normalizing provided its corresponding DFPTS is β-strongly normalizing.

In the second part of this section, we consider the possibility of scaling up the result to CPTSs. It is easy to establish $\beta\Delta$-strong normalization for a proof-irrelevant CPTS as a consequence of β-strong normalization for its corresponding PTS. Unfortunately, it is not possible to apply the technique directly to proof-relevant CPTSs. We analyze this negative result and locate its cause as the unorthodox behavior of Δ-reduction.

In the third part of this section, we turn to standard proof techniques for strong normalization. Specifically, we present a general model construction which is based on saturated sets and which may be used to derive $\beta\Delta$-strong normalization for a large class of CPTSs including e.g. the *Classical Calculus of Constructions*. Along with the construction we present a sufficiency criterion for the model to be well-defined. Both the model and the criterion are inspired from previous work by Z. Luo [56] and J. Terlouw [93] on proving strong normalization for the Extended Calculus of Constructions and $\lambda\Pi$-type systems respectively.

In the fourth part of this section, we consider yet another standard proof technique for strong normalization, namely the technique of reduction-preserving mappings. More precisely, we examine the Harper-Honsell-Plotkin translation [36] from –the PTS core of– Edinburgh's Logical Frameworks to simply typed λ-calculus and the Geuvers-Nederhof translation [31] from Coquand's Calculus of Constructions to Girard's higher-order λ-calculus. It turns out that these translations lift to the framework of DFCPTSs but not to that of CPTSs. In the latter case, this failure is once more due to the unorthodox behavior of Δ-reduction.

- *Section 7: Classical Pure Type Systems as Logics*

  This brief section collects some basic facts about the logical status of classical pure type systems. It is by no means a complete account of the logical properties of CPTSs.

  In the first subsection, we establish for impredicative and proof-irrelevant logical specifications, a correspondence between classical provability and intuitionistic provability by using the well-known encoding of classical logic as an intuionistic context.

  In the second subsection, we study the formulae-as-sets embedding. Once we view, as suggested above, the CPTSs of the L-cube as "the"classical logics, the embedding may be studied from a purely type-theoretical standpoint: as the homomorphic extension of a suitable morphism of specifications from the systems of the L-cube to the systems of the λ-cube. We prove that the embedding is sound –typing is preserved by the translation– but incomplete for the predicate logics– typing is not reflected by the translation.

  In the third subsection, we prove that the Classical Calculus of Construc-

6

tions $\lambda\Delta C$ is proof-relevant, thus generalising a result of [88,91].

- *Section 8: Issues and related work*

The first part of the paper –Sections 3 to 6– is exclusively concerned with properties of classical pure type systems and lacks of any consideration as to the design of classical pure type systems. In this section, we discuss design issues for classical λ-calculi and propose some alternative frameworks for classical pure type systems.

In the first subsection, we discuss the possible formats for classical classical natural deduction and the corresponding choices for the syntax of terms. Moreover, we discuss the relationship between our notion of CPTS and Prawitz's classical natural deduction. In the second subsection, we consider the issue of reduction, which is undoubtedly one of the least understood aspects of classical λ-calculus. Rather than singling out a few set of reduction rules out of the dozens of existing ones, we suggest syntactical criteria upon which to classify the rules and isolate some of these criteria as fundamental for the theory of classical pure type systems. These criteia lead us to distinguish between two forms of reduction rules: principal rules which generate a notion of proof equivalence –or in more general terms definitional equality– and minor rules which do not –typically, such rules involve an element of non-deterministic choice which causes the failure of confluence. As it will appear, this separation combines the advantages of 'well-behaved' and 'powerful' classical λ-calculi by having on the one hand a well-behaved principal reduction relation and on the other hand a powerful minor reduction relation. The distinction turns out to be especially handy in specific applications, such as witness extraction.

In the third subsection, we briefly discuss related work.

2 Preliminaries

The first subsection introduces some basic terminology for binary relations and is mostly taken from [48]. The second subsection is devoted to computational type systems.

2.1 Relations

Throughout this subsection, we let X denote an arbitrary set and $\mathcal{R}$, $\mathcal{S}$ denote binary relations over X. Elements of X are called objects. We use the following notation.

Definition 2.1

- $\mathcal{R}.\mathcal{S}$ denotes the composition of $\mathcal{R}$ and $\mathcal{S}$.
- $\mathcal{R}^{op}$ denotes the inverse of $\mathcal{R}$.
- $\downarrow_{\mathcal{R}}$ denotes the relation $\mathcal{R}^{\omega} \cdot (\mathcal{R}^{op})^{\omega}$.

- *The closures of $\mathcal{R}$ are denoted as follows —where R stands for reflexive, S for symmetric and T for transitive, C for closure:*

Notion	TC	RTC	RSTC
Notation	$\mathcal{R}^+$	$\mathcal{R}^\omega$	$=_\mathcal{R}$

Some of the relations will written as $\to_i$, in which case we use a standard notation. In particular, we use $\to_{ij}$ to denote the union of two relations $\to_i$ and $\to_j$ and write $\twoheadrightarrow_i$ instead of $\to_i^\omega$, $=_i$ instead of $=_{\to_i}$, and $\downarrow_i$ instead of $\downarrow_{\to_i}$.

We also introduce some standard properties of relations.

Definition 2.2

- *A relation $\mathcal{R}$ is* locally confluent *if $\mathcal{R}^{op} \cdot \mathcal{R} \subseteq \downarrow_\mathcal{R}$.*

- *A relation $\mathcal{R}$ is* confluent *if $=_\mathcal{R} \subseteq \downarrow_\mathcal{R}$.*

- *An object a is* normal *with respect to a relation $\mathcal{R}$ if there is no b s.t. $a\mathcal{R}b$.*

- *An object a is* weakly normalizing *with respect to a relation $\mathcal{R}$ if there is a normal object b s.t. $a\mathcal{R}^\omega b$.*

- *An object a is* strongly normalizing *with respect to a relation $\mathcal{R}$ if there is no infinite reduction sequence $a\mathcal{R}a'\mathcal{R}\ldots$*

The sets of normal, weakly normalizing and strongly normalizing objects (w.r.t. $\mathcal{R}$) are denoted by $NF_\mathcal{R}$, $WN_\mathcal{R}$, $SN_\mathcal{R}$.

2.2 Computational type systems

In order to deal uniformly with various λ-calculi, we introduce the notion of computational type system. The notion combines features of higher-order rewriting systems [49] and of abstract type systems [11].

Definition 2.3 *A* computational type system *is a 7-tuple*

$$\mathbb{T} = (V, S, C, \mathcal{D}, \mathcal{F}, \gamma, \vdash)$$

such that:

- *V, S, C, $\mathcal{D}$ and $\mathcal{F}$ are disjoint sets. Elements of V, S, C, $\mathcal{D}$ and $\mathcal{F}$ are respectively called* variables, sorts, constants, domain-specified quantifiers *and* domain-free quantifiers;

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

$$\mathcal{U} = V \mid S \mid C \mid \mathcal{U}\mathcal{U} \mid \mathsf{d}V : \mathcal{U}.\,\mathcal{U} \mid \mathsf{f}V.\,\mathcal{U}$$

- *$\gamma \subseteq \mathcal{U} \times \mathcal{U}$ is a* notion of reduction;

- *$\vdash \subseteq \mathsf{List}(V \times \mathcal{U}) \times \mathcal{U} \times \mathcal{U}$ is the* derivability relation.

Elements of $\mathsf{List}(V \times \mathcal{U})$ are called contexts; *elements of $\mathsf{List}(V \times \mathcal{U}) \times \mathcal{U} \times \mathcal{U}$ are called* judgements.

Throughout the paper, we adopt the following conventions.

- We write $\Gamma \vdash \Delta$ with $\Delta \equiv x_1 : A_1, \ldots, x_n : A_n$ if $\Gamma \Vdash^J x_i : A_i$ for $i = 1, \ldots, n$.

- We write $\Gamma \vdash M : A : B$ for $\Gamma \vdash M : A$ and $\Gamma \vdash A : B$.

Computational type systems allow us to introduce in a generic way standard notions from higher-order rewrite systems and type theories.

2.2.1 Notions related to terms and reduction

We first define the notion of strict subterm.

Definition 2.4 *The subterm relation $\lhd$ is defined inductively as follows:*

- $M \lhd M\ N$ *and* $N \lhd M\ N$;

- $M \lhd dx : A.M$ *and* $A \lhd dx : A.M$;

- $M \lhd fx.M$;

- *if* $M \lhd N$ *and* $N \lhd P$, *then* $M \lhd P$.

If $M \lhd N$, *then* M *is a* subterm *of* N.

Next we define the notion of free and bound variables.

Definition 2.5 *The sets $FV(M)$ and $BV(M)$ of* free variables *and* bound variables *of a pseudo-term M are defined inductively as follows:*

$$FV(x) = \{x\} \qquad\qquad BV(x) = \emptyset$$
$$FV(s) = \emptyset \qquad\qquad BV(s) = \emptyset$$
$$FV(c) = \emptyset \qquad\qquad BV(c) = \emptyset$$
$$FV(M\ N) = FV(M) \cup FV(N) \qquad\qquad BV(M\ N) = BV(M) \cup BV(N)$$
$$FV(dx : A.\ M) = FV(A) \cup (FV(M) \setminus \{x\}) \qquad BV(dx : A.\ M) = BV(A) \cup BV(M) \cup \{x\}$$
$$FV(fx.\ M) = FV(M) \setminus \{x\} \qquad\qquad BV(fx.\ M) = BV(M) \cup \{x\}$$

For reasons of hygiene, we adopt *Barendregt's convention* and assume that $FV(M) \cap BV(M) = \emptyset$ for every pseudo-term M. Substitution is defined in the usual, capture-avoiding, way.

Definition 2.6 *Let $M, N \in \mathcal{U}$ and let $x \in V \setminus BV(M)$. The pseudo-term*

9

$M\{x := N\}$ *is defined inductively as follows*

$$x\{x := N\} \equiv N$$

$$y\{x := N\} \equiv y \qquad\qquad provided\ y \in V\ and\ y \not\equiv x$$

$$s\{x := N\} \equiv s \qquad\qquad for\ s \in S \cup C$$

$$(P\,Q)\{x := N\} \equiv (P\{x := N\})\,(Q\{x := N\})$$

$$(\mathsf{d}y : A.\ P)\{x := N\} \equiv \mathsf{d}y : (A\{x := N\}).\ (P\{x := N\})$$

$$(\mathsf{f}y.\ P)\{x := N\} \equiv \mathsf{f}y.\ (P\{x := N\})$$

Each computational type system has a reduction relation, which is obtained from its notion of reduction in the usual way.

Definition 2.7 *The reduction relation* $\to_\gamma$ *is defined as the compatible closure of* γ*, i.e. as the smallest relation s.t. for every* $M, M', N \in \mathcal{U}$

$$(M, M') \in \gamma \Rightarrow M \to_\gamma M'$$

$$M \to_\gamma M' \Rightarrow M\ N \to_\gamma M'\ N$$

$$M \to_\gamma M' \Rightarrow N\ M \to_\gamma N\ M'$$

$$M \to_\gamma M' \Rightarrow \mathsf{d}x : N.\ M \to_\gamma \mathsf{d}x : N.\ M'$$

$$M \to_\gamma M' \Rightarrow \mathsf{d}x : M.\ N \to_\gamma \mathsf{d}x : M'.\ N$$

$$M \to_\gamma M' \Rightarrow \mathsf{f}x.M \to_\gamma \mathsf{f}x.M'$$

2.2.2 Notions related to typing

We turn to type-theoretic notions, in particular to the notion of legality.

Definition 2.8

- *A judgement* (Γ, M, A) *is* legal *if* $\Gamma \vdash M : A$.

- *A context* Γ *is* legal *if* $\Gamma \vdash M : A$ *for some* M *and* A.

- *A pseudo-term* M *is* legal *if* $\Gamma \vdash M : A$ *or* $\Gamma \vdash A : M$ *for some* Γ *and* A.

One can also define specific classes of pseudo-terms.

Definition 2.9

- *A pseudo-term* M *is a* s-type *in context* Γ *with* $s \in S$ *if* $\Gamma \vdash M : s$.

- *A pseudo-term* M *is a* s-term *in context* Γ *with* $s \in S$ *if* $\Gamma \vdash M : A$ *and* $\Gamma \vdash A : s$ *for some pseudo-term* A.

- *A pseudo-term* M *is a* s-type *if it is a* s-type *in context* Γ *for some context* Γ.

- *A pseudo-term* M *is a* s-term *if it is a* s-term *in context* Γ *for some context* Γ.

- *A pseudo-term* M *is a* type *if it is a* s-type *for some sort* s.

The next definition is concerned with normalization properties of legal terms.

Definition 2.10 $\mathbb{T}$ *is*

- *normalizing, notation* $\mathbb{T} \models WN$, *if* $M \in WN_\gamma$ *for every legal* M;

- *strongly normalizing, notation* $\mathbb{T} \models SN$, *if* $M \in SN_\gamma$ *for every legal* M;

- *type-normalizing, notation* $\mathbb{T} \models WN(type)$, *if* $M \in WN_\gamma$ *for every type* M;

- *type-strongly-normalizing, notation* $\mathbb{T} \models SN(type)$, *if* $M \in SN_\gamma$ *for every type* M.

2.2.3 Morphisms of computational type systems

This subsubsection is devoted to technical definitions which will be used in Section 7. Its reading may be postponed until that point.

First, we define the notion of morphism of computational type systems. There are many possible definitions. Here we only need a very simple one.

Definition 2.11 *Let* $\mathbb{T}_i = (V, S_i, C, \mathcal{D}, \mathcal{F}, \gamma, \vdash)$ *be a computational type system (i = 1, 2). A morphism from* $\mathbb{T}_1$ *to* $\mathbb{T}_2$ *is a map* $|.| : S_1 \to S_2$.

Every morphism induces in the obvious way a map on pseudo-terms, contexts and judgements. By abuse of notation, we let $|.|$ denote these maps.

Definition 2.12 *A morphism* $|.|$ *of computational type systems is:*

- sound *if for every context* Γ *and pseudo-terms* M *and* A

$$\Gamma \vdash_1 M : A \quad \Rightarrow \quad |\Gamma| \vdash_2 |M| : |A|$$

- complete *w.r.t.* $s \in S$ *if for every context* Γ *and pseudo-terms* N *and* A

$$\left. \begin{array}{c} \Gamma \vdash_1 A : s \\ |\Gamma| \vdash_2 N : |A| \end{array} \right\} \quad \Rightarrow \quad \exists M \in \mathcal{U}_1.\ \Gamma \vdash_1 |M| : |A|$$

If $S_1 \subseteq S_2$ *and* $|.|$ *is the identity, we say that* $\mathbb{T}_2$ *is* conservative *over* $\mathbb{T}_1$ *w.r.t.* s.

The next lemma collects some basic facts concerning complete morphisms.

Lemma 2.13

(i) *The identity morphism attached to a computational type system is complete.*

(ii) *The composition of two complete morphisms is complete.*

(iii) *If* $H' \circ H$ *is a complete morphism and* H' *is sound, then* H *is complete.*

2.2.4   Environments

This subsubsection is devoted to technical definitions which will be used in the study of strong normalization. Its reading should therefore be postponed until the reader reaches Section 6.

First, we define the notion of environment, which conveniently captures by the notion of infinite context [79,56] and allows for a simple formulation of the criterion.

Definition 2.14 *An environment $\mathcal{E}$ is an infinite sequence of variable declarations $\mathcal{E} \equiv x_1 : E_1, x_2 : E_2, \ldots$ such that for every $i > 0$:*

- $\mathcal{E}^i \equiv x_1 : E_1, \ldots, x_i : E_i$ *is a legal context;*

- *if $\mathcal{E}^i \vdash A : s$ and $s \in S$ then there exists infinitely many k s.t. $E_k \equiv A$.*

We write $\mathcal{E} \vdash M : A$ whenever there exists $i \geq 0$ s.t. $\mathcal{E}^i \vdash M : A$.

Second, we define the notion of layered computational type system which provides a suitable criterion for strong normalization.

Definition 2.15

- *Let $\mathcal{E}$ be an environment. A pseudo-term M is an $\mathcal{E}$-prototype if there exists $n \geq 0$, $s \in S$ and $P_1, \ldots, P_n \in \mathcal{T}$ s.t. $\mathcal{E} \vdash M\ P_1\ \ldots\ P_n : s$. The set of $\mathcal{E}$-prototypes is denoted by $\mathcal{E}$-Proto.*

- *Let $\mathcal{E}$ be an environment. The relation $\prec_\mathcal{E}$ on pseudo-terms is defined as the smallest relation such that for all $M, N \in \mathcal{T}$, we have $N \prec M$ and $M\ N \prec M$ whenever $M\ N \in \mathcal{E}$-Proto.*

- *A type system is layered if $\prec_\mathcal{E}$ is well-founded for every environment $\mathcal{E}$.*

3   A notion of classical pure type system

The design of a classical λ-calculus supposes many choices. Rather than trying to discuss each choice here, we limit ourselves to giving the definition of a notion of classical pure type system and postpone –in as much as possible– the discussion until Section 8.

3.1   Specifications

Specifications are tuples expressing certain dependencies and are used to generate type systems. In our case, specifications come equipped with a distinguished sort of propositions.

Definition 3.1 *A logical specification S is a quadruple $(S, Prop, A, R)$ where*

- *S is a set of sorts and $\mathsf{Prop} \in S$;*

- *$A \subseteq S \times S$ is the set of axioms;*

- *$R \subseteq S \times S \times S$ is the set of rules;*

satisfying the following properties:

- *Property 1:* $(\mathsf{Prop}, \mathsf{Prop}, \mathsf{Prop}) \in R$;

- *Property 2: all the rules involving* Prop *are of the form* $(s, \mathsf{Prop}, \mathsf{Prop})$ *or* $(\mathsf{Prop}, s_1, s_1)$;

- *Property 3: there is no sort* s *for which* $(s, \mathsf{Prop}) \in A$;

- *Property 4: there is a sort* s *for which* $(\mathsf{Prop}, s) \in A$.

As usual, rules of the form (s_1, s_2, s_2) *are abbreviated as* (s_1, s_2).

The meaning of a specification may be intuitively explained as follows: Prop stands for the universe of propositions. The other sorts are the possible universes in which non-propositional types –to be thought e.g. as sets– may live. Axioms correspond to basic assumptions which determine the belonging of a certain sort to a certain universe. This is reflected in the deductive system of classical pure type systems through the (Axiom) rule. For example, Property 3 ensures that no universe inhabits Prop whereas Property 4 implies the existence of a universe for which Prop is an inhabitant. Finally, the rules indicate which products may be formed. For example, Property 1 states that it is possible to define from two propositions A and B the proposition $A \to B$.

Remark 3.2 *Our notion of logical specification is closely related but not identical to the notion of logical specification in [20]. The latter notion requires the specification to be functional –see Definition 3.2– and more importantly does not require Property 2. However, [20] mainly focuses on proof-irrelevant specifications –see Definition 3.3 below–, which occur as special cases of logical specifications. Property 4, which also occurs in Coquand and Herbelin's definition, ensures that variables inhabiting* Prop *may be introduced. It is only needed in Section 5.*

Definition 3.3 *A logical specification* $S = (S, \mathsf{Prop}, A, R)$ *is proof-irrelevant if for every* $(s_1, s_2, s_3) \in R$ *we have* $s_1 \neq \mathsf{Prop}$ *or* $s_2 \equiv s_3 \equiv \mathsf{Prop}$.

We close this section with the definition of top-sort.

Definition 3.4 *A sort* $s \in S$ *is a top-sort if there is no* $s' \in S$ *s.t.* $(s, s') \in A$. *The set of top-sorts is denoted by* $S^\top$.

3.2 Classes of specifications

Throughout the paper, we will consider various classes of specifications. These are defined below.

Definition 3.5 *A logical specification* (S, Prop, A, R) *is:*

- functional *if for all* $s_1, s_2, s, s' \in S$,
  - $(s_1, s) \in A \quad \wedge \quad (s_1, s') \in A \quad \Rightarrow \quad s \equiv s'$
  - $(s_1, s_2, s) \in R \quad \wedge \quad (s_1, s_2, s') \in R \quad \Rightarrow \quad s \equiv s'$

- injective *if it is functional and for all* $s_1, s_3, s, s' \in S$
  - $(s, s_1) \in A \quad \wedge \quad (s', s_1) \in A \quad \Rightarrow \quad s \equiv s'$
  - $(s_1, s, s_3) \in R \quad \wedge \quad (s_1, s', s_3) \in R \quad \Rightarrow \quad s \equiv s'$
- proof-irrelevant *if there is no rule* (Prop, s) *with* $s \not\equiv \mathsf{Prop}$.

The notion of functional specification ensures that a term has at most one type in a given context –see Lemma 4.13– whereas the notion of injective specification allows for a characterization of the terms of a given universe – see Propositions 4.18 and 4.19. The notion of proof-irrelevant specification ensures that only proofs –i.e. inhabitants of inhabitants of Prop– may depend on proofs.

3.3 Examples of specifications

In [5], Barendregt gives a fine-grain analysis of the Calculus of Constructions in form of the λ-cube whereas in [30], Geuvers defines the logic cube (or L-cube) which represents some of the most important logics.[4]

Definition 3.6 *The* λ-cube *and* L-cube *specifications are given in Figure 2 and Figure 3 respectively. The* λ- *and* L-cube *are depicted in Figure 4.*

Note that the L-cube specifications are proof-irrelevant. The correspondence between specifications of the λ-cube, L-cube and logics is given in Figure 5. The nature and significance of the correspondence are discussed –for the constructive case– in [6,13,30,94]. See also Section 7.

3.4 Classical pure type systems

In this section, we let $S = (S, \mathsf{Prop}, A, R)$ be a logical specification and define its induced classical pure type system. The set of terms is built from the usual constants (sorts, variables) and constructions (λ-abstraction, Π-abstraction, application) as well as from a new constant $\bot$ for *falsum* and an extra binding *double negation* construction Δ –see Section 8.

Definition 3.7 *Let* V *be an infinite set of variables. The set of* pseudo-terms *is given by the abstract syntax*

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

As usual, $A \to B$ *is used to denote* $\Pi x{:}A.\,B$ *when* $x \notin FV(B)$. *Moreover we write* $\neg A$ *for* $A \to \bot$.[5]

---

[4] Closely related logic cubes have been proposed independently by Barendregt [6] and Berardi [13].

[5] The first convention will also apply to all the systems considered in this paper. The second convention applies also to domain-free classical pure type systems and to pure type systems and domain-free pure type systems when working in a context of the form $\bot : \mathsf{Prop}, \Gamma$.

Sorts: $*, \square$

Prop $*$

Axioms: $* : \square$

Rules:

$\rightarrow$	$(*, *)$			
2	$(*, *)$	$(\square, *)$		
$\underline{\omega}$	$(*, *)$	$(\square, \square)$		
ω	$(*, *)$	$(\square, *)$	$(\square, \square)$	
P	$(*, *)$	$(*, \square)$		
$P2$	$(*, *)$	$(\square, *)$	$(*, \square)$	
$P\underline{\omega}$	$(*, *)$	$(*, \square)$	$(\square, \square)$	
$P\omega = C$	$(*, *)$	$(\square, *)$	$(*, \square)$	$(\square, \square)$

Fig. 2. THE λ-CUBE SPECIFICATIONS

Sorts: $*^p, *^s, \square^p, \square^s$

Prop $*^p$

Axioms: $*^p : \square^p, *^s : \square^s$

Rules:

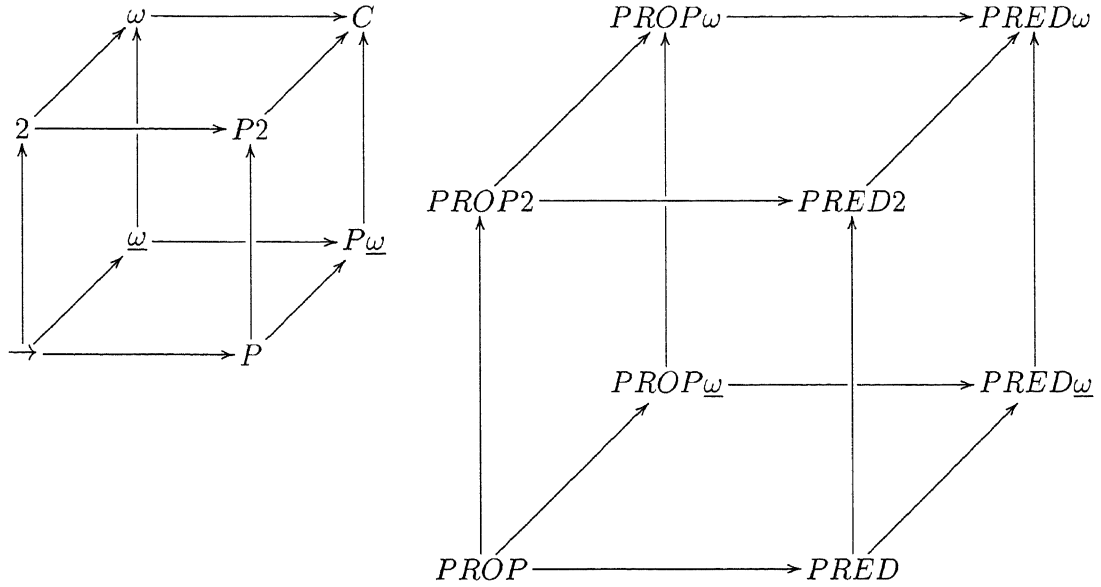|							
$PROP$	$(*^p, *^p)$						
$PROP2$	$(*^p, *^p)$	$(\square^p, *^p)$					
$PROP\underline{\omega}$	$(*^p, *^p)$	$(\square^p, \square^p)$					
$PROP\omega$	$(*^p, *^p)$	$(\square^p, *^p)$	$(\square^p, \square^p)$				
$PRED$	$(*^p, *^p)$	$(*^s, \square^p)$	$(*^s, *^s)$	$(*^s, *^p)$			
$PRED2$	$(*^p, *^p)$	$(*^s, \square^p)$	$(*^s, *^s)$	$(*^s, *^p)$	$(\square^p, *^p)$		
$PRED\underline{\omega}$	$(*^p, *^p)$	$(*^s, \square^p)$	$(*^s, *^s)$	$(*^s, *^p)$	$(\square^p, \square^p)$	$(\square^p, *^s)$	
$PRED\omega$	$(*^p, *^p)$	$(*^s, \square^p)$	$(*^s, *^s)$	$(*^s, *^p)$	$(\square^p, *^p)$	$(\square^p, \square^p)$	$(\square^p, *^s)$

Fig. 3. THE L-CUBE SPECIFICATIONS

Fig. 4. PICTURE OF THE CUBES

PROP	$\rightarrow$	first-order propositional logic
PROP2	2	second-order propositional logic
$PROP\underline{\omega}$	$\underline{\omega}$	weak higher-order propositional logic
$PROP\omega$	ω	higher-order propositional logic
PRED	P	first-order predicate logic
PRED2	P2	second-order predicate logic
$PRED\underline{\omega}$	$P\underline{\omega}$	weak higher-order predicate logic
$PRED\omega$	C	higher-order predicate logic

Fig. 5. CORRESPONDENCE BETWEEN λ-CUBE SPECIFICATIONS, L-CUBE SPECIFICATIONS AND LOGICS

One of the main difficulties in trying to define a notion of classical pure type system is to choose a notion of reduction for the Δ-operator. We take a minimalist approach and consider a single rule which makes applications of double negation atomic. The rule is inspired from normalisation procedures for classical natural deduction [80,81,87,90] and occurs in the majority of reduction systems for control operators, see for example [28,29]. For further discussion on the definition of reduction, see Section 8.

16

Definition 3.8

- *The notion of reduction β is defined by the contraction rule*

$$(\lambda x{:}A.\ b)\ c \to_\beta b\{x := c\}$$

- *The notion of reduction Δ is defined by the contraction rule*

$$(\Delta x{:}\neg(\Pi v{:}A_1.A_2).a)\,b \to_\Delta \Delta y{:}\neg A_2\{v := b\}.a\{x := \lambda z{:}(\Pi v{:}A_1.A_2).y\,(z\,b)\}$$

Finally, we define the derivability relation for classical pure type systems.

Definition 3.9

- *The relation* $\Gamma \vdash A : B$ *is defined by the rules of Figure 6.*[6]
- *The* classical pure type system $\lambda\Delta S$ *is the computational type system*

$$(V, S, \{\bot\}, \{\lambda, \Pi, \Delta\}, \emptyset, \beta\Delta, \vdash)$$

3.5 Domain-free classical pure type systems

In this section, we define a variant of classical pure type system, called domain-free classical pure type systems, in which abstractions come without domain tags. So let $S = (S, \mathsf{Prop}, A, R)$ be a logical specification and let V be an infinite set of variables.

Definition 3.10

- *The set of* pseudo-terms *is given by the abstract syntax*

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

- *The notion of reduction $\underline{\beta}$ is defined by the contraction rule*

$$(\lambda x.b)\ c \to_\beta b\{x := c\}$$

- *The notion of reduction $\underline{\Delta}$ is defined by the contraction rule*

$$(\Delta x.\ a)\ b \to_\Delta \Delta y.\ a\{x := \lambda z.y\,(z\,b)\}$$

- *The relation* $\Gamma \Vdash A : B$ *is defined by the rules of Figure 7.*
- *The* domain-free classical pure type system $\lambda\Delta S$ *is the computational type system*

$$(V, S, \{\bot\}, \{\Pi\}, \{\lambda\Delta\}, \underline{\beta\Delta}, \Vdash)$$

The unusual conversion rule is adopted to enforce the Classification Lemma –see Propositions 4.18 and 4.19.

---

[6] Strictly speaking, we should write $\vdash_S$ rather than $\vdash$. However, the subscript is dropped in order to avoid clutter and will only be used when there is a risk of confusion.

17

(axiom-$\perp$) $<> \vdash \perp : \mathsf{Prop}$

(axiom) $<> \vdash s_1 : s_2$ if $(s_1, s_2) \in A$

(start) $\dfrac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}$ if $x \notin \Gamma$

(weakening) $\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B}$ if $x \notin \Gamma$

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

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

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

(double negation) $\dfrac{\Gamma, x : \neg A \vdash b : \perp \quad \Gamma \vdash A : \mathsf{Prop}}{\Gamma \vdash \Delta x{:}\neg A.\ b : A}$

(conversion) $\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s}{\Gamma \vdash A : B'}$ if $B =_{\beta \Delta} B'$

Fig. 6. Deductive rules for classical pure type systems

3.6 Pure type systems and domain-free pure type systems

Every specification **S** generates both a pure type system and a domain-free pure type system.

Definition 3.11 *Let $S = (S, \mathsf{Prop}, A, R)$ be a pre-logical specification and let V be a fixed set of variables.*

18

(axiom-$\perp$)	$<> \Vdash \perp : \mathsf{Prop}$	
(axiom)	$<> \Vdash s_1 : s_2$	if $(s_1, s_2) \in A$
(start)	$\dfrac{\Gamma \Vdash A : s}{\Gamma, x : A \Vdash x : A}$	if $x \notin \Gamma$
(weakening)	$\dfrac{\Gamma \Vdash A : B \quad \Gamma \Vdash C : s}{\Gamma, x : C \Vdash A : B}$	if $x \notin \Gamma$
(product)	$\dfrac{\Gamma \Vdash A : s_1 \quad \Gamma, x : A \Vdash B : s_2}{\Gamma \Vdash (\Pi x{:}A.\, B) : s_3}$	if $(s_1, s_2, s_3) \in R$
(application)	$\dfrac{\Gamma \Vdash F : (\Pi x{:}A.\, B) \quad \Gamma \Vdash a : A}{\Gamma \Vdash F\,a : B\{x := a\}}$	
(abstraction)	$\dfrac{\Gamma, x : A \Vdash b : B \quad \Gamma \Vdash (\Pi x{:}A.\, B) : s}{\Gamma \Vdash \lambda x.b : \Pi x{:}A.\, B}$	
(double negation)	$\dfrac{\Gamma, x : \neg A \Vdash b : \perp \quad \Gamma \Vdash A : \mathsf{Prop}}{\Gamma \Vdash \Delta x.\, b : A}$	
(conversion)	$\dfrac{\Gamma \Vdash A : B \quad \Gamma \Vdash B : s \quad \Gamma \Vdash B' : s}{\Gamma \Vdash A : B'}$ if $B =_{\beta\Delta} B'$	

Fig. 7. DEDUCTIVE RULES FOR DOMAIN-FREE CLASSICAL PURE TYPE SYSTEMS

- *The* pure type system *generated by* S *is the computational type system*

$$\lambda S = (V, S, \emptyset, \{\lambda, \Pi\}, \emptyset, \beta, \vdash^J)$$

where β is defined as in Definition 3.8 and $\vdash^J$ is defined by the rules of pure type systems –see Figure 14, back page. The set of pseudo-terms of λS is denoted by $\mathcal{T}^J$.

- *The* domain-free pure type system *generated by* S *is the computational type*

19

system

$$\underline{\lambda}S = (V, S, \emptyset, \{\Pi\}, \{\lambda\}, \underline{\beta}, \Vdash^J)$$

where $\underline{\beta}$ is defined as in Definition 3.10 and $\Vdash^J$ is defined by the rules of domain-free pure type systems –see Figure 15, back page. The set of pseudo-terms of $\underline{\lambda}S$ is denoted by $\underline{\mathcal{T}}^J$.

The main properties of pure type systems and domain-free pure type systems may be found respectively in [6,30] and [11].

4 Basic properties of classical pure type systems

This section collects some basic results concerning CPTSs. Most results are as for pure type systems; it is therefore convenient to follow a pattern similar to that of [6, Section 5.2].

In the first subsection, we prove confluence of $\beta\Delta$-reduction and deduce some of its most important consequences. In the second subsection, we establish some basic properties for arbitrary CPTSs. In the third subsection, we establish some further properties for specific classes of CPTSs. In the fourth subsection, we consider which of the previously established results apply to DFCPTSs. In the fifth subsection, we consider the relationship between CPTSs and DFCPTSs. Throughout this section and unless explicitly stated, $S = (S, \mathsf{Prop}, A, R)$ denotes a fixed logical specification.

4.1 Confluence of $\beta\Delta$ and applications

Reduction is closed under substitution.

Lemma 4.1

(i) $G =_{\beta\Delta} H \Rightarrow E\{x := G\} =_{\beta\Delta} E\{x := H\}$;

(ii) $E =_{\beta\Delta} F, G =_{\beta\Delta} H \Rightarrow E\{x := G\} =_{\beta\Delta} F\{x := H\}$.

Proof.

(i) By induction on E.

(ii) By induction on the derivation of $E =_{\beta\Delta} F$ using (i).

$\square$

The following Proposition may be proved by several means.

Proposition 4.2 (Confluence) *The notion of reduction $\beta\Delta$ is confluent.*

Proof. Using –for example– the technique of Tait and Martin-Löf. $\square$

The following consequences of confluence are often crucial, e.g. in proving subject reduction.

Corollary 4.3 (Key Lemma)

20

- $\Pi v{:}\,A_1.\,A_2 =_{\beta\Delta} \Pi v{:}\,A_1'.\,A_2' \quad \Rightarrow \quad A_1 =_{\beta\Delta} A_1',\, A_2 =_{\beta\Delta} A_2'$
- $s =_{\beta\Delta} s' \quad \Rightarrow \quad s \equiv s'$

4.2 Basic results for abritrary classical pure type systems

Lemma 4.4 (Free Variables) *If* $\langle x_1 : A_1, \ldots, x_n : A_n \rangle \vdash B : C$ *then:*

(i) $x_1, \ldots, x_n$ *are distinct;*

(ii) $FV(B) \cup FV(C) \subseteq \{x_1, \ldots, x_n\};$

(iii) $\langle x_1 : A_1, \ldots, x_{i-1} : A_{i-1} \rangle \vdash A_i : s$ *for each* $i = 1, \ldots, n.$

Proof. By induction on the derivation of $\langle x_1 : A_1, \ldots, x_n : A_n \rangle \vdash B : C.$ $\square$

Lemma 4.5 (Start) *If* Γ *is legal then:*

(i) $\Gamma \vdash s_1 : s_2$ *for all* $(s_1, s_2) \in A;$

(ii) $\Gamma \vdash \bot : \mathsf{Prop};$

(iii) $\Gamma \vdash x : A$ *for all* $x : A \in \Gamma.$

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

Lemma 4.6 (Transitivity) *If* Δ *is legal then*

$$\Delta \vdash \Gamma \quad \wedge \quad \Gamma \vdash A : B \quad \Rightarrow \quad \Delta \vdash A : B$$

Proof. By induction on the derivation of $\Gamma \vdash A : B$, using the Start Lemma. $\square$

Lemma 4.7 (Thinning) *If* $\Delta \subseteq \Gamma$ *are both legal then:*

$$\Gamma \vdash A : B \Rightarrow \Delta \vdash A : B$$

Proof. This follows from the Start Lemma and the Transitivity Lemma. $\square$

Lemma 4.8 (Substitution)

$$\left.\begin{array}{l} \Gamma, x : A, \Delta \vdash B : C \\[2mm] \Gamma \vdash a : A \end{array}\right\} \quad \Rightarrow \quad \Gamma, \Delta\{x := a\} \vdash B\{x := a\} : C\{x := a\}$$

Proof. By induction on the derivation of $\Gamma, x : A, \Delta \vdash B : C.$ $\square$

The next lemma is useful to determine how a judgement can be derived and is used throughout the paper.

Lemma 4.9 (Generation)

(i) $\Gamma \vdash \bot : C \Rightarrow C =_{\beta\Delta} \mathsf{Prop}$

(ii) $\Gamma \vdash s : C \Rightarrow \exists (s, s') \in A.\ C =_{\beta\Delta} s'$

21

(iii) $\Gamma \vdash x : C \Rightarrow \exists s \in S, \ D \in \mathcal{T}. \ C =_{\beta\Delta} D, x : D \in \Gamma, \Gamma \vdash D : s$

(iv) $\Gamma \vdash \lambda x{:}A. \ b : C \Rightarrow \exists s \in S, \ B \in \mathcal{T}. \ C =_{\beta\Delta} \Pi x{:}A. \ B, \Gamma, x : A \vdash b : B, \Gamma \vdash \Pi x{:}A. \ B : s$

(v) $\Gamma \vdash \Delta x{:}A. \ b : C \Rightarrow \exists B \in \mathcal{T}. \ C =_{\beta\Delta} B, \neg B \equiv A, \Gamma, x : \neg B \vdash b : \bot, \Gamma \vdash B : \mathsf{Prop}$

(vi) $\Gamma \vdash \Pi x{:}A. \ B : C \Rightarrow \exists (s_1, s_2, s_3) \in R. \ C =_{\beta\Delta} s_3, \Gamma \vdash A : s_1, \Gamma, x : A \vdash B : s_2$

(vii) $\Gamma \vdash F \ a : C \Rightarrow \exists x \in V, \ A, B \in \mathcal{T}. \ C =_{\beta\Delta} B\{x := a\}, \Gamma \vdash F : \Pi x{:}A. \ B, \Gamma \vdash a : A$

Proof. By induction on the derivation of (i)-(vii) using the Thinning Lemma.□

Lemma 4.10 (Correctness of types)

$$\Gamma \vdash A : B \quad \Rightarrow \quad B \in S^\top \quad \vee \quad \exists s \in S. \ \Gamma \vdash B : s$$

Proof. By induction on the derivation of $\Gamma \vdash A : B$. □

Proposition 4.11 (Subject and predicate reduction)

(i) $\Gamma \vdash M : A \quad \wedge \quad M \to_{\beta\Delta} N \quad \Rightarrow \quad \Gamma \vdash N : A$

(ii) $\Gamma \vdash M : A \quad \wedge \quad A \to_{\beta\Delta} B \quad \Rightarrow \quad \Gamma \vdash M : B$

Proof. (ii) follows from (i) by Correctness of types. As for (i), we prove the following two statements by simultaneous induction on the derivation of $\Gamma \vdash M : A$:

- $M \to_{\beta\Delta} N \quad \Rightarrow \quad \Gamma \vdash N : A$
- if $\Gamma \to_{\beta\Delta} \Delta \quad \Rightarrow \quad \Delta \vdash M : B$

We treat the first item in the case of an application when

$$M \equiv (\Delta x{:}\neg(\Pi y{:}A_1. \ A_2). \ a) \ b$$

and N is obtained from M by contracting the outermost Δ-redex, i.e.

$$N \equiv \Delta y{:}\neg A_2\{v := b\}. \ a\{x := \lambda z{:}(\Pi v{:}A_1. \ A_2). \ (y \ (z \ b))\}$$

So the last rule is

$$\frac{\Gamma \vdash \Delta x{:}\neg(\Pi v{:}A_1. \ A_2). \ a : (\Pi v{:}A_3. \ A_4) \quad \Gamma \vdash b : A_3 \quad \Gamma \vdash \Pi v{:}A_1. \ A_2 : s}{\Gamma \vdash (\Delta x{:}\neg(\Pi y{:}A_1. \ A_2). \ a) \ b : A_4\{v := b\}}$$

It is convenient to introduce the notation $C^\dagger$ for $C\{v := b\}$. We are to show

22

$\Gamma \vdash N : A_4^\dagger$. By generation, we have

$$\Pi v{:}A_1.\, A_2 =_{\beta\Delta} \Pi v{:}A_3.\, A_4$$

$$\Gamma, x : \neg(\Pi v{:}A_1.\, A_2) \vdash a : \bot$$

$$\Gamma \vdash \Pi v{:}A_1.\, A_2 : \mathsf{Prop}$$

$$\Gamma \vdash A_1 : s_1$$

$$\Gamma, v : A_1 \vdash A_2 : s_2$$

with $(s_1, s_2, \mathsf{Prop}) \in R$. By Property 2, $s_2 \equiv \mathsf{Prop}$. By assumption, $\Gamma \vdash b : A_3$. By confluence, $A_1 =_{\beta\Delta} A_3$ and $A_2^\dagger =_{\beta\Delta} A_4^\dagger$. Hence

$\Gamma \vdash b : A_1$	(conversion)
$\Gamma \vdash A_2^\dagger : \mathsf{Prop}$	(substitution)
$\Gamma \vdash \neg A_2^\dagger : \mathsf{Prop}$	(thinning, Property 1)
$\Gamma, y : \neg A_2^\dagger, z : \Pi v{:}A_1.\, A_2 \vdash y\,(z\,b) : \bot$	(start, weakening, application)
$\Gamma, y : \neg A_2^\dagger \vdash \lambda z{:}(\Pi v{:}A_1.\, A_2).\, y\,(z\,b) : \neg(\Pi v{:}A_1.\, A_2)$	(abstraction, Property 1)
$\Gamma, y : \neg A_2^\dagger \vdash a\{x := \lambda z{:}(\Pi v{:}A_1.\, A_2).\, (y\,(z\,b))\} : \bot$	(substitution)
$\Gamma \vdash N : A_2^\dagger$	(double negation)
$\Gamma \vdash A_4^\dagger : s_3$	(correctness of types)
$\Gamma \vdash N : A_4^\dagger$	(conversion)

$\square$

Proposition 4.12 (Consistency)

$$\lambda S \models WN_{\beta\Delta} \quad \Rightarrow \quad \forall M \in \mathcal{T}.\neg\,(\vdash M : \bot)$$

Proof. Define a trivially consistent context to be one of the form

$$x_1 : \neg\bot,\ \ldots\ ,x_n : \neg\bot$$

We prove that there is no pseudo-term M s.t. $\Gamma \vdash M : \bot$ for some trivial context Γ.

Assume towards a contradiction that such a Γ and M exists. Without loss of generality, one may assume that M is in normal form and is minimal, i.e. does not contain any subterm N s.t. $\Gamma' \vdash N : \bot$ for some trivially consistent context Γ'. M cannot be:

- a variable because $\bot$ is not convertible with $\neg\bot$;
- a sort because $\bot$ is not convertible with a sort;

- $\perp$ because $\perp$ is not convertible with Prop;
- a product because $\perp$ is not convertible with a sort;
- a λ-abstraction because $\perp$ is not convertible with a Π-term;
- a Δ-abstraction because of the minimality of M.

So necessarily M must be an application. Assume that $M \equiv M_1\, M_2\, \ldots\, M_n$ where M_1 is not an application. For typability reasons, M_1 can only be a variable, a λ-abstraction or a Δ-abstraction; the last two cases are impossible because M is in $\beta\Delta$-normal form. So M_1 is a variable. Necessarily M_1 occurs in the context and has type $\neg\perp$ and hence $n = 2$ and $M = M_1\, M_2$. Therefore M_2 is of type $\perp$, contradicting the minimality of M. $\qquad\square$

4.3 Basic results for specific classes of CPTSs

4.3.1 Functional CPTSs

In this Section, three properties are examined: Uniqueness of Types, Strengthening and Decidability of Type-Checking. Functionality is obviously crucial for the former property. As for the latter properties, it should be possible to eliminate the assumption of functionality by following an approach similar to [12] –however proofs become more involved. Throughout this section, **S** denotes a functional logical specification.

Proposition 4.13 (Uniqueness of types)

(i) $\Gamma \vdash M : B \quad \wedge \quad \Gamma \vdash M : B' \quad \Rightarrow \quad B =_{\beta\Delta} B'$

(ii) $\Gamma \vdash M : B \quad \wedge \quad \Gamma \vdash M' : B' \quad \wedge \quad M =_{\beta\Delta} M' \quad \Rightarrow \quad B =_{\beta\Delta} B'$

Proof.

(i) By induction on M using generation and confluence.

(ii) If $M =_{\beta\Delta} M'$ then by confluence there is an M'' such that $M \twoheadrightarrow_{\beta\Delta} M''$ and $M' \twoheadrightarrow_{\beta\Delta} M''$. By subject reduction $\Gamma \vdash M'' : B$ and $\Gamma \vdash M'' : B'$. Hence by (i) $B =_{\beta\Delta} B'$.

$\qquad\square$

Proposition 4.14 (Strengthening)

$\Gamma_1, x : A, \Gamma_2 \vdash b : B \quad \wedge \quad x \notin FV(\Gamma_2) \cup FV(b) \cup FV(B) \quad \Rightarrow \quad \Gamma_1, \Gamma_2 \vdash b : B$

Proof. Prove by induction on the derivation of $\Gamma_1, x : A, \Gamma_2 \vdash b : B$ that

$$\left. \begin{array}{l} \Gamma_1, x : A, \Gamma_2 \vdash b : B \\[2mm] x \notin FV(\Gamma_2) \cup FV(b) \cup FV(B) \end{array} \right\} \quad \Rightarrow \quad \exists B \in \mathcal{T}.\, B \twoheadrightarrow_{\beta\Delta} B' \quad \wedge \quad \Gamma_1, \Gamma_2 \vdash b : B' \quad (*)$$

Then assume $\Gamma_1, x : A, \Gamma_2 \vdash b : B$. Use $(*)$ to find B' s.t. $\Gamma_1, \Gamma_2 \vdash b : B'$ with $B \twoheadrightarrow_{\beta\Delta} B'$. By Correctness of Types, either $B \in S^\top$, in which case $B \equiv B'$

and we are done, or $\Gamma_1, x : A, \Gamma_2 \vdash B : s$ for some $s \in S$. By $(*)$, $\Gamma_1, \Gamma_2 \vdash B : s$. We conclude by applying the conversion rule. $\qquad\square$

Definition 4.15

- *The* type-checking problem *(TC) consists in deciding whether a given judgement $\Gamma \vdash M : A$ is derivable.*

- *The* type-synthesis problem *(TS) consists in deciding whether a pseudoterm M has a type in a given pseudo-context Γ, i.e. whether there exists A s.t. $\Gamma \vdash M : A$ is derivable.*

Decidablity of type-synthesis is especially useful for theorem proving.

Proposition 4.16 (Decidability of type checking and type synthesis)

If $\lambda S \models WN_{\beta\Delta}$ and both sets A and R are decidable, then TC and TS are decidable.

Proof. Define an algorithm $\mathsf{ty}_\Gamma(M)$ which returns, when it exists, a type for M in context Γ and returns $\uparrow$ otherwise. The algorithm is given in Figure 8; it makes use of an auxiliary function $\mathsf{leg}(\Gamma)$ which checks whether a context is legal. More efficient algorithms can be derived, see [77]. $\qquad\square$

4.3.2 Injective CPTSs

The central result of this section is a classification lemma for injective specifications. As usual with this kind of result, we partition the set of variables V as $\bigcup_{s \in S} V^s$ in such a way that each V^s is countably infinite and $V^s \cap V^{s'} = \emptyset$ for $s \not\equiv s'$. Moreover manipulate variables according to the rules:

$$(\text{start-s}) \qquad \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \qquad \text{if } x \notin \Gamma \text{ and } x \in V^s$$

$$(\text{weakening-s}) \qquad \frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B} \qquad \text{if } x \notin \Gamma \text{ and } x \in V^s$$

Throughout this subsection, $S = (S, \mathsf{Prop}, A, R)$ is an injective logical specification. Moreover, for every $s \in S$, we define s^+ as the unique (if it exists) sort s' s.t. $(s, s') \in A$, s^- as the unique (if it exists) sort s' s.t. $(s', s) \in A$. The definition below gives a syntactic description of the classes of legal terms.

$$\mathsf{ty}_\Gamma(x) = \begin{cases} A \text{ if } \mathsf{leg}(\Gamma) \text{ and } x : B \in \Gamma \text{ for some } B =_{\beta\Delta} A \\ \uparrow \text{ otherwise} \end{cases}$$

$$\mathsf{ty}_\Gamma(s) = \begin{cases} s' \text{ if } \mathsf{leg}(\Gamma) \text{ and } (s, s') \in A \\ \uparrow \text{ otherwise} \end{cases}$$

$$\mathsf{ty}_\Gamma(\bot) = \begin{cases} \mathsf{Prop} \text{ if } \mathsf{leg}(\Gamma) \\ \uparrow \quad \text{ otherwise} \end{cases}$$

$$\mathsf{ty}_\Gamma(M\ N) = \begin{cases} B\{x := N\} \text{ if } \mathsf{ty}_\Gamma(M) \twoheadrightarrow_{wh} \Pi x{:}A.\ B \text{ and } \mathsf{ty}_\Gamma(N) =_{\beta\Delta} A \\ \uparrow \qquad\qquad \text{ otherwise} \end{cases}$$

$$\mathsf{ty}_\Gamma(\lambda x{:}A.\ M) = \begin{cases} \Pi x{:}A.\ B \text{ if } \mathsf{ty}_{(\Gamma,x:A)}(M) = B \\ \uparrow \qquad\qquad \text{ otherwise} \end{cases}$$

$$\mathsf{ty}_\Gamma(\Pi x{:}A.\ B) = \begin{cases} s_3 \text{ if } (s_1, s_2, s_3) \in R, \mathsf{ty}_\Gamma(A) \twoheadrightarrow_{wh} s_1 \text{ and } \mathsf{ty}_{(\Gamma,x:A)}(B) \twoheadrightarrow_{wh} s_2 \\ \uparrow \text{ otherwise} \end{cases}$$

$$\mathsf{ty}_\Gamma(\Delta x{:}A.\ M) = \begin{cases} A' \text{ if } A = \neg A', \mathsf{ty}_\Gamma(A') \twoheadrightarrow_{wh} \mathsf{Prop} \text{ and } \mathsf{ty}_{\Gamma,x:A}(M) \twoheadrightarrow_{wh} \bot \\ \uparrow \text{ otherwise} \end{cases}$$

$$\mathsf{leg}(\langle\rangle) = true$$

$$\mathsf{leg}(\Gamma, x : A) = \begin{cases} true \text{ if } ty_\Gamma(A) \twoheadrightarrow_{wh} s \text{ for some } s \in S \text{ and } x \text{ fresh in} \Gamma \\ false \text{ otherwise} \end{cases}$$

Fig. 8. Type checking algorithm for functional normalising CPTSs

Definition 4.17 *Let* s, s_1, s_2, s_3 *range over sorts.*

$$\mathsf{Term}^s = V^s \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad |$$

$$(s^-)^- \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad |$$

$$\Pi V^{s_1} : \mathsf{Type}^{s_1}.\mathsf{Type}^{s_2} \qquad \text{IF} \quad (s_1, s_2, s^-) \in R \ |$$

$$\lambda V^{s_1} : \mathsf{Type}^{s_1}.\mathsf{Term}^{s_2} \qquad \text{IF} \quad (s_1, s_2, s) \in R \quad |$$

$$\mathsf{Term}^{s_3}\ \mathsf{Term}^{s_1} \qquad\qquad\quad \text{IF} \quad (s_1, s, s_3) \in R \quad |$$

$$\Delta V^{\mathsf{Prop}} : \mathsf{Type}^{\mathsf{Prop}}.\mathsf{Term}^{Prop} \ \text{IF} \quad s \equiv \mathsf{Prop} \qquad |$$

$$\bot \qquad\qquad\qquad\qquad\qquad \text{IF} \quad s \equiv \mathsf{Prop}^+$$

If s is not a top-sort, $\mathsf{Type}^s = \mathsf{Term}^{s^+}$. *Otherwise,*

$$\mathsf{Type}^s = s^- | \Pi V^{s_1} : \mathsf{Type}^{s_1}.\mathsf{Type}^{s_2} \quad \text{IF} \quad (s_1, s_2, s) \in R$$

Moreover, let $\mathsf{Proof} = \mathsf{Term}^{\mathsf{Prop}}$ *and* $Form = \mathsf{Type}^{\mathsf{Prop}}$.

The Classification Lemma shows that terms may be partitioned according to the sort of the types in which they can live.

Proposition 4.18 (Classification 1) *Let* Txxx *range over* Term *and* Type *and let* s, s' *be sorts.*

(i) Txxx^s *is closed under reduction.*

(ii) $s \not\equiv s' \quad \Rightarrow \quad \mathsf{Txxx}^s \cap Txxx^{s'} = \emptyset$

(iii) $s' \in S^\top \quad \Rightarrow \quad s' \notin \mathsf{Txxx}^s$

(iv) $s' \in S^\top \quad \Rightarrow \quad \mathsf{Term}^s \cap Type^{s'} = \emptyset$

Proof. First prove closure under substitution, i.e. for every $s, s' \in S$,

$$\left. \begin{array}{c} x \in V^s \\ P \in \mathsf{Term}^s \\ M \in \mathsf{Txxx}^{s'} \end{array} \right\} \quad \Rightarrow \quad M\{x := P\} \in \mathsf{Txxx}^{s'}$$

where Txxx is either Term or Type. Then prove the proposition. All proofs proceed by induction on the structure of $Term^s$ and Type^s. $\square$

The next proposition shows that s-terms belong to Term^s and s-types belong to Type^s.

Proposition 4.19 (Classification 2) *Let* $s \in S$.

(i) $\Gamma \vdash M : A \quad \wedge \quad \Gamma \vdash A : s \quad \Rightarrow \quad M \in \mathsf{Term}^s$

(ii) $\Gamma \vdash A : s \quad \Rightarrow \quad A \in \mathsf{Type}^s$

Proof. Both statements are proved simultaneously by induction on the structure of derivations. $\square$

4.4 Properties of domain-free classical pure type systems

Domain-free classical pure type systems are not the main focus of this paper so we limit this section to some brief comments. All the results in Subsection 4.2 and Subsubsection 4.3.2 hold for DFCPTSs. In the latter case we must however be careful with variables and require the name of bound variables to be relevant. In particular, we require α-conversion to replace variables in V^s by variables in V^s. The results of Subsubsection 4.3.1, Uniqueness of Types and Decidability of Type Checking, do not apply [11].

We close this subsubsection by examining the relationship between CPTSs and DFCPTSs. Let S be a functional logical specification.

Definition 4.20 *The erasure map $|.| : \mathcal{T} \to \underline{\mathcal{T}}$ is defined as follows:*

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

Erasure preserves reduction, equality and typing:

Proposition 4.21

(i) $M \to_{\beta\Delta} N \quad \Rightarrow \quad |M| \to\!\!\!\to_{\underline{\beta\Delta}} |N|$

(ii) $M =_{\beta\Delta} N \quad \Rightarrow \quad |M| =_{\underline{\beta\Delta}} |N|$

(iii) $\Gamma \vdash M : A \quad \Rightarrow \quad |\Gamma| \Vdash |M| : |A|$

Proof. First prove by induction on M that

$$|M\{x := N\}| \equiv |M|\{x := |N|\} \tag{$*$}$$

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

Note that the proposition does not hold immediately for non-functional specifications because of the conversion rule for DFCPTSs.

Erasure, as defined above, does not preserve infinite reductions because redexes occuring in the domain of λ- and Δ-abstractions are lost during erasure. In the case of pure type systems, it is possible to define a modified erasure map $|.|_k$ that preserves reductions: it is done simply by extending domain-free pseudo-terms with a new construction $K\ M\ N$ whose rewrite behavior is given by $K\ x\ y \to_\kappa x$ and by modifying the inductive case for λ-abstractions into

$$|\lambda x : A.M|_K \equiv K\ (\lambda x.|M|_K)\ |A|_K$$

Unfortunately, the idea does not scale up to CPTSs.

Definition 4.22

- *The set $\mathcal{W}$ is defined as follows:*

$$\mathcal{W} = V \mid S \mid \bot \mid \mathcal{W}\,\mathcal{W} \mid \lambda V.\mathcal{W} \mid \Delta V.\,\mathcal{W} \mid \Pi V{:}\mathcal{W}.\,\mathcal{W} \mid K\ \mathcal{W}\ \mathcal{W}$$

- *The notion of reduction κ is defined by the contraction rule*

$$K\ x\ y \quad \to_\kappa x$$

- *The modified erasure map* $|.|_k : \mathcal{T} \to \mathcal{W}$ *is defined as follows:*

$$|x|_k = x$$
$$|\bot|_k = \bot$$
$$|s|_k = s$$
$$|t\ u|_k = |t|_k\ |u|_k$$
$$|\lambda x : A.t|_k = \mathsf{K}\ (\lambda x.|t|_K)\ |A|_k$$
$$|\Delta x : A.t|_k = \mathsf{K}\ (\Delta x.|t|_K)\ |A|_k$$
$$|\Pi x : A.B|_k = \Pi x : |A|_k.|B|_k$$

Modified erasure preserves β-reductions but not Δ-reductions.

Lemma 4.23

- $M \to_\beta N \quad \Rightarrow \quad |M|_k \twoheadrightarrow_{\beta\kappa} |N|_k$
- $M \to_\Delta N \quad \Rightarrow \quad |M|_k \twoheadrightarrow_{\underline{\Delta}\kappa} |N|_k$

Proof. The first item is proved by induction on the structure of the terms. For the second item, note that the translation of a Δ-redex does not reduce to the translation of its Δ-reduct. $\qquad\square$

In the sequel of the paper, the failure of erasure to reflect Δ-reduction will be referred to as the unorthodox behavior of Δ-reduction.

5 CPS translation and applications

In this section, we define a Contination-Passing Style translation for injective logical specifications. The CPS translation is inspired from [10] where we develop CPS translations for logical pure type systems. In the first subsection, we discuss some of the problems arising from the use of domain-specified abstractions. In the second subsection, we define the CPS translation and prove its correctness. In the third subsection, we derive, as an application of our translation, consistency of a CPTS from consistency of its associated PTS. In the fourth subsection, we look at the image of impredicatively defined connectives by the CPS translation.

5.1 Background

Pure type systems feature domain-specified λ-abstractions of the form $\lambda x : A.\ M$. Unfortunately, such abstractions are a significant obstacle to a simple and useful formulation of CPS translations. Indeed, CPS translations introduce new λ-abstractions whose domains need to be inferred. Consider the judgement

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

where A is a formula. If we decide to translate CPTS pseudo-terms into PTS pseudo-terms, the CPS translation for x should yield $\lambda k : C.\ x\ k$ for some suitable term C. It turns out that, if typability is to be preserved, C should

correspond to the top-level translation of A. As a result, the CPS translation cannot be defined by induction on the structure of the terms but should use a more complex induction principle. Such induction principles do exist in some cases, e.g. for proof-irrelevant specifications [20] or for the specifications of the λ-cube [10]. However, the existence of such an induction principle, e.g. in the case of the Calculus of Constructions, relies on heavy proof-theoretic arguments: in [10], we define a domain-specified CPS translation for the PTSs of the λ-cube using an induction principle taken from [26]. The latter is obtained as a corollary to strong normalization of a labelled version of the Calculus of Constructions. Instead of relying on such powerful proof-theoretic properties, we choose to work with domain-free systems and translate DFCPTS pseudo-terms into DFPTS pseudo-terms.

5.2 The translation

Throughout this section, **S** denotes an injective logical specification. For reasons that will appear later, we assume that we are given an infinite supply of *special* variables, ranged over by h, i, j, k, which do not appear in the legal terms of the DFCPTS.

Definition 5.1 *The CPS translations* $\mathbb{C}\langle . \rangle$ *and* $\mathbb{C}\langle\!\langle . \rangle\!\rangle$ *are defined in Figure 9. Moreover, if* $M \in \underline{\mathcal{T}}^J$ *and* $N \in \mathcal{T}$, *we let* $\mathbb{C}^M \langle N \rangle$ *and* $\mathbb{C}^M \langle\!\langle N \rangle\!\rangle$ *denote respectively* $\mathbb{C}\langle N \rangle \{ \bot := M \}$ *and* $\mathbb{C}\langle\!\langle N \rangle\!\rangle \{ \bot := M \}$.

The CPS translation is correct in the following sense.

Theorem 5.2 $\Gamma \Vdash A : B \quad \Rightarrow \quad \mathbb{C}\langle\!\langle \Gamma \rangle\!\rangle \Vdash^J \mathbb{C}\langle A \rangle : \mathbb{C}\langle\!\langle B \rangle\!\rangle$

Proof. We proceed in four steps:

(i) show that for $B \in \mathsf{Proof}$, $\mathbb{C}\langle B \rangle \equiv \lambda k.C$. Therefore, $\lambda k.\mathbb{C}\langle B \rangle\, k \to_\beta \mathbb{C}\langle B \rangle$;

(ii) prove by induction on the structure of $A \in \mathsf{Txxx}^s$ that for $x \in V^{s'}$ and $B \in \mathsf{Term}^{s'}$
$$\mathbb{C}\langle A \rangle \{ x := \mathbb{C}\langle B \rangle \} \twoheadrightarrow_{\underline{\beta}} \mathbb{C}\langle A \{ x := B \} \rangle$$

(iii) prove by induction on the derivation of $A \in \mathsf{Txxx}^s$ that
$$A \to_{\underline{\beta\Delta}} B \Rightarrow \mathbb{C}\langle A \rangle =_{\underline{\beta}} \mathbb{C}\langle B \rangle$$

(iv) prove by induction on the structure of derivations
$$\Gamma \Vdash A : B \quad \Rightarrow \quad \mathbb{C}\langle\!\langle \Gamma \rangle\!\rangle \Vdash^J \mathbb{C}\langle A \rangle : \mathbb{C}\langle\!\langle B \rangle\!\rangle$$
$\square$

5.3 Applications of the CPS translation to consistency of CPTSs

One specific application of Theorem 5.2 is to prove consistency of CPTSs.

$$\mathbb{C}\langle x \rangle \quad = \begin{cases} \lambda k.x\ k & \text{if } x \in V^{\mathsf{Prop}} \\ x & \text{otherwise} \end{cases}$$

$$\mathbb{C}\langle s \rangle \quad = s$$

$$\mathbb{C}\langle \bot \rangle \quad = \bot$$

$$\mathbb{C}\langle \lambda x.M \rangle \quad = \begin{cases} \lambda k.k\ (\lambda x.\mathbb{C}\langle M \rangle) & \text{if } \lambda x.M \in \mathsf{Proof} \\ \lambda x.\mathbb{C}\langle M \rangle & \text{otherwise} \end{cases}$$

$$\mathbb{C}\langle M\ M' \rangle \quad = \begin{cases} \lambda k.\mathbb{C}\langle M \rangle\ (\lambda j.j\ \mathbb{C}\langle M' \rangle\ k) & \text{if } M\ M' \in \mathsf{Proof} \\ \mathbb{C}\langle M \rangle\ \mathbb{C}\langle M' \rangle & \text{otherwise} \end{cases}$$

$$\mathbb{C}\langle \Delta x.\ M \rangle \quad = \lambda k.\mathbb{C}\langle M \rangle \{ x := \lambda h.h\ \lambda j.\lambda i.i\ (j\ k) \} \lambda z.z$$

$$\mathbb{C}\langle \Pi x{:}A.\ B \rangle \ = \Pi x{:}\mathbb{C}(\!|A|\!).\ \mathbb{C}(\!|B|\!)$$

$$\mathbb{C}(\!|M|\!) \quad = \begin{cases} \neg\neg\mathbb{C}\langle M \rangle & \text{if } M \in \mathsf{Form} \\ \mathbb{C}\langle M \rangle & \text{otherwise} \end{cases}$$

$$\mathbb{C}(\!|[\ ]|\!) \quad = \bot : \mathsf{Prop}$$

$$\mathbb{C}(\!|\Gamma, x : A|\!) \ = \mathbb{C}(\!|\Gamma|\!), x : \mathbb{C}(\!|A|\!)$$

Fig. 9. CPS TRANSLATION

Proposition 5.3 *Assume* $\lambda^J S \models WN(\text{type})$. *Then the four conditions below are equivalent*

(i) $\vdash M : \bot$ *for some* M;

(ii) $\Vdash M : \bot$ *for some* M;

(iii) $\perp : \mathsf{Prop} \vdash^J M : \perp$ *for some* M;

(iv) $\perp : \mathsf{Prop} \Vdash^J M : \perp$ *for some* M.

Proof. Obviously (iii) implies (i) implies (ii). The equivalence between (iii) and (iv) is proved in [11]. We prove (ii) implies (iv) and we are done. Suppose $\Vdash \quad A : \perp$. By Theorem 5.2 $\perp : \mathsf{Prop} \quad \Vdash^J \quad \mathbb{C}\langle A \rangle : (\perp \to \perp) \to \perp$, so $\Vdash^J \mathbb{C}\langle A \rangle \ \lambda z.z : \perp$. $\qquad\square$

Theorem 5.2 may also be used to infer consistency of some contexts in a CPTS. For the sake of brevity, we only treat the case of the classical L-cube. As expected, the CPS translation acts as the identity on 'sets', i.e. inhabitants of $*^s$.

Lemma 5.4 *If* $A \in \mathsf{Type}^{*^s}$, *then* $\perp \notin FV(\mathbb{C}\langle\!\langle A \rangle\!\rangle)$ *and* $\mathbb{C}\langle\!\langle A \rangle\!\rangle \equiv A$.

Moreover, the CPS translation preserves the internal equality on sets.

Lemma 5.5 *If* $A \in \mathsf{Type}^{*^s}$ *and* $\Gamma \Vdash \quad M : a_1 =_A a_2$ *for some* M, *then* $\mathbb{C}^{(a_1 =_A a_2)}\langle\!\langle \Gamma \rangle\!\rangle \Vdash^J M : a_1 =_A a_2$ *for some* M.

Proof. Let $D \equiv a_1 =_A a_2$. By Theorem 5.2,

$$\mathbb{C}\langle\!\langle \Gamma \rangle\!\rangle \Vdash^J \mathbb{C}\langle M \rangle : \mathbb{C}\langle\!\langle D \rangle\!\rangle$$

and by the Substitution Lemma,

$$\mathbb{C}^D \langle\!\langle \Gamma \rangle\!\rangle \Vdash^J \mathbb{C}^D \langle M \rangle : \mathbb{C}^D \langle\!\langle D \rangle\!\rangle$$

Moreover we have

$$\mathbb{C}\langle\!\langle D \rangle\!\rangle \equiv \neg\neg\Pi P{:}A \to *^p.\ \neg\neg(\neg\neg P\ x) \to (\neg\neg P\ y)$$

Thus one may construct a term P of type $\mathbb{C}^D \langle\!\langle D \rangle\!\rangle \to D$. Hence $P\,\mathbb{C}^D \langle M \rangle$ has type D and we are done. $\qquad\square$

These two facts may be used to infer consistency of classical arithmetic in higher-order predicate logic. One possible formalisation of arithmetic is given by the context Peano in Figure 10 –here addition and multiplication are not taken as primitives but are defined by recursion.

Proposition 5.6 *Peano is a consistent context in* $\lambda\triangle PRED\omega$.

Proof. It is easy to see that Peano is consistent iff there is no term M of type $\mathsf{z} =_{\mathsf{N}} (\mathsf{s}\ \mathsf{z})$ in context Peano. By Theorem 5.2, Lemma 5.5 and some elementary reasoning,

$$\exists M.\ \mathsf{Peano} \Vdash M : \mathsf{z} =_{\mathsf{N}} (\mathsf{s}\ \mathsf{z}) \quad \Rightarrow \quad \exists M.\ \mathbb{C}^{(\mathsf{z}=_{\mathsf{N}}(\mathsf{s}\ \mathsf{z}))}\langle\!\langle \mathsf{Peano} \rangle\!\rangle \Vdash^J M : \mathsf{z} =_{\mathsf{N}} (\mathsf{s}\ \mathsf{z})$$

Conclude from the fact that $\mathbb{C}^{(\mathsf{z}=_{\mathsf{N}}(\mathsf{s}\ \mathsf{z}))}\langle\!\langle \mathsf{Peano} \rangle\!\rangle$ is a consistent context in $\lambda PRED\omega$. $\square$

$\mathsf{N} : *^s,$

$\mathsf{z} : \mathsf{N}, \quad \mathsf{s} : \mathsf{N} \to \mathsf{N} \to \mathsf{N},$

$\mathsf{rec} : \mathsf{N} \to (\mathsf{N} \to \mathsf{N} \to \mathsf{N}) \to \mathsf{N} \to \mathsf{N},$

$< : \mathsf{N} \to \mathsf{N} \to *^p,$

$\mathsf{ind} : \Pi P : \mathsf{N} \to *^p. \ P \ \mathsf{z} \to (\Pi n : \mathsf{N}. \ (P \ n) \to (P \ (\mathsf{s} \ n))) \to \Pi n : \mathsf{N}. \ P \ n,$

$\mathsf{rec}_0 : \Pi f_0 : \mathsf{N}. \ \Pi f_s : \mathsf{N} \to \mathsf{N} \to \mathsf{N}. \ (\mathsf{rec} \ f_0 \ f_s \ \mathsf{z}) =_\mathsf{N} f_0,$

$\mathsf{rec}_s : \Pi f_0 : \mathsf{N}. \ \Pi f_s : \mathsf{N} \to \mathsf{N} \to \mathsf{N}. \ (\mathsf{rec} \ f_0 \ f_s \ (\mathsf{s} \ n)) =_\mathsf{N} (f_s \ n \ (\mathsf{rec} \ f_0 \ f_s \ n)),$

$\mathsf{s}_i : \Pi m, n : \mathsf{N}. \ (\mathsf{s} \ m =_\mathsf{N} \mathsf{s} \ n) \to (m =_\mathsf{N} n),$

$\mathsf{p}_0 : \Pi n : \mathsf{N}. \ \neg \ (\mathsf{s} \ n =_\mathsf{N} \mathsf{z}),$

$\mathsf{o}_1 : \Pi m, n : \mathsf{N}. \ (x < \mathsf{s} \ y) \to (x < y \lor x =_\mathsf{N} y),$

$\mathsf{o}_2 : \Pi m, n : \mathsf{N}. \ (x < y \lor x =_\mathsf{N} y) \to (x < \mathsf{s} \ y)$

Fig. 10. A context for Peano arithmetic

It is important to realize that Propositions 5.3 and 5.6 are proved without invoking any normalization property of CPTSs.

Remark 5.7 *In [88], Seldin shows that so-called strongly negation consistent contexts are consistent in the Calculus of Constructions. One can elaborate on his ideas and use the CPS translation to isolate some classes of consistent contexts. This development is omitted here.*

6 Strong normalization

This section is concerned with strong normalisation of domain-free classical pure type systems and classical pure type systems. Strong normalisation of the former is reduced to strong normalisation of pure type systems by a refinement of the CPS translation. In contrast, strong normalisation of the latter is proved by a model construction. Proofs are omitted and will be presented in the full version of the paper.

Throughout this section, we assume that S is an injective logical specification.

6.1 Strong normalization by CPS translation for DFCPTSs

The CPS translation does not preserve reduction. Yet one may use an optimized translation which contracts some of the administrative redexes and

preserve –in a weak sense– reductions. Using some elementary reasoning, one concludes:

Theorem 6.1 (Domain-Free Preservation of Strong Normalization)

$$\underline{\lambda}S \models SN \quad \Rightarrow \quad \underline{\lambda}\Delta S \models SN$$

6.2 Strong normalization by CPS translation for CPTSs

If S is proof-irrelevant, one may apply Theorem 6.1 to deduce strong normalization of $\lambda\Delta S$ from strong normalization of λS. To do so, the following two observations are needed.

Lemma 6.2 If S is proof-irrelevant and $M \in \mathcal{T}$ is a type in $\lambda\Delta S$, then M is a type in λS.

Proof. By induction on the structure of derivations, prove that proofs may only occur as subterms in proofs. □

In particular, it follows that $\lambda S \models$ SN(type) implies $\lambda\Delta S \models$ SN(type).

Lemma 6.3 If S is proof-irrelevant, $\lambda\Delta S \models SN(type)$ and $\underline{\lambda}\Delta S \models SN$, then $\lambda\Delta S \models SN$.

Proof. First show that there cannot be an infinite $\beta\Delta$-reduction sequence starting from a legal term M and such that

$$
\begin{cases}
M \equiv M_0 \rightarrow_{\beta\Delta} M_1 \rightarrow_{\beta\Delta} \cdots \\
|M_0| \equiv |M_1| \equiv \ldots
\end{cases}
\tag{\#}
$$

To prove (#), use correctness of types to deduce that M is either a top-sort or $\Gamma \vdash M : A$ for some Γ and A. Then proceed by induction on the derivation of $\Gamma \vdash M : A$ using $\lambda\Delta S \models$ SN(type).

Second, conclude by using $\underline{\lambda}\Delta S \models$ SN. □

Putting it all together,

Theorem 6.4 (Domain-Specified Preservation of Strong Normalization)
If S is proof-irrelevant,

$$\lambda S \models SN \quad \Rightarrow \quad \lambda\Delta S \models SN$$

Proof. Assume $\lambda S \models$ SN. Then $\underline{\lambda}S \models$ SN –see [11]. By Theorem 6.1, $\underline{\lambda}\Delta S \models$ SN. Moreover, $\lambda\Delta S \models$ SN(type). By Lemma 6.3, $\lambda\Delta S \models$ SN. □

Unfortunately, it is not possible to extend immediately the result to proof-relevant CPTSs. Indeed, the obvious solution would consist in extending the domain-free systems with the K-combinator –see [9] for a definition of pure type system with the K-combinator–, prove a result analogous to Theorem

6.1 for such systems and conclude strong normalisation of $\lambda\Delta S$ from strong normalisation of $\underline{\lambda\Delta K}$ –the domain-free classical pure type system extended with a K-combinator. But such a reduction does not work because of the unorthodox behavior of Δ-reduction –see Subsection 6.1.

6.3 Strong normalization by a model construction

Terlouw has given a general criterion for a $\lambda\Pi$ type system to be strongly normalizing [93] –the criterion is also implicitly present in [56]. A similar criterion can be used for CPTSs.

Theorem 6.5 If $\lambda\Delta S$ is layered, then $\lambda\Delta S \models SN$.

Proof. See Appendix. $\square$

In order to prove that a CPTS $\lambda\Delta S$ is layered, one can use the correctness of the CPS translation.

Proposition 6.6 If λS is layered and $\lambda S \models WN(\text{type})$, then $\lambda\Delta S$ is layered.

Proof. See Appendix. $\square$

Alternatively, it is sometimes equally easy to proceed by hand.

Lemma 6.7 If S is a specification of the λ-cube, then $\lambda\Delta S$ is layered.

Proof. Let $\mathcal{E}$ be an environment. Define a measure on $\mathcal{E}$-types as follows:

- $\nu(A) = 0$ if A is an $\mathcal{E}$-proposition,
- $\nu(*) = 1$,
- $\nu(\Pi x{:}A.\ B) = \nu(A) + \nu(B) + 1$ if $\Pi x{:}A.\ B$ is an $\mathcal{E}$-kind.

The measure is preserved by conversion. By uniqueness of types, ν yields a measure $\underline{\nu}$ on pseudo-terms:

$$\underline{\nu}(M) = \begin{cases} n & \text{if } \exists A \in \mathcal{E}-\text{Type. } \mathcal{E} \vdash M : A \text{ and } \nu(A) = n \\ 0 & \text{otherwise} \end{cases}$$

For every $M, N \in \mathcal{T}$, $N \prec_{\mathcal{E}} M$ implies $\underline{\nu}(N) < \underline{\nu}(M)$. Hence $\prec_{\mathcal{E}}$ is well-founded. $\square$

It follows:

Corollary 6.8 Systems of the classical λ-cube are strongly normalising.

Proof. By Theorem 6.5 and Lemma 6.7. $\square$

6.4 Strong normalization à la Harper-Honsell-Plotkin

In [36], Harper, Honsell and Plotkin prove strong normalisation for Edinburgh Logical Frameworks, which is essentially equivalent to λP, by defining

35

a reduction and derivation preserving mapping from legal λP-terms to legal $\underline{\lambda}^\tau \to$-terms –here $\underline{\lambda}^\tau \to$ denotes an extension of $\underline{\lambda} \to$ with some pairing operators. Later Geuvers and Nederhof generalized this translation by taking as source theory the Calculus of Constructions λC and as target theory Girard's higher-order λ-calculus $\lambda\omega$ [31]. Both translations use two mappings:

- a mapping τ which acts on constructors, i.e. $\Box$-terms, and kinds, i.e. $\Box$-types;

- a mapping $[.]$ which acts on legal terms and preserves reductions.

Unfortunately, the reduction-preserving mapping techniques of Harper-Honsell-Plotkin and Geuvers-Nederhof do not seem to extend to the framework of CPTSs. Interestingly, the technique applies to DFCPTSs. We present our analysis of this fact for the Harper-Honsell-Plotkin mapping. A similar analysis may be conducted for the Geuvers-Nederhof mapping.

Definition 6.9 *[36] The Harper-Honsell-Plotkin translation* $[.]$ *maps λP-pseudo-terms to $\underline{\lambda}^\tau \to$pseudo-terms. It is defined as follows:*

$$[x] = x$$
$$[*] = *$$
$$[P\,Q] = [P]\,[Q]$$
$$[\lambda x{:}A.\,M] = (\lambda y.\lambda x.[M])\,[A]$$
$$[\Pi x{:}A.\,B] = \pi_{\tau(A)}\,[A]\,\lambda x.[B]$$

where $\pi_{\tau(A)}$ is a constant.

As mentioned earlier

Lemma 6.10 *[36] $[.]$ preserves reductions.*

Let us now consider extending the translation to $\lambda\Delta P$-pseudo-terms. The obvious choices are

$$[\bot] = \bot$$
$$[\Delta x{:}A.\,M] = (\lambda y.\Delta x.\,[M])\,[A]$$

Unfortunately, the extended translation $[.]$ does not preserve reductions. Indeed, consider the terms

$$M \equiv (\Delta x{:}\neg(\Pi v{:}A_1.\,A_2).\,a)\,b$$
$$N \equiv \Delta y{:}\neg A_2\{v := b\}.\,a\{x := \lambda z{:}(\Pi v{:}A_1.\,A_2).\,(y\,(z\,b))\}$$

We have $M \to_{\Delta N}$ but only $[M] =_{\beta\Delta} [N]$. As a result, it is not possible to prove strong normalization à la Harper-Honsell-Plotkin for $\lambda\Delta P$. However, one may define a domain-free variant of the translation which preserves reductions.

Definition 6.11 *The translation $[.]_{df}$ is defined as follows:*

$$[x]_{df} = x$$
$$[\bot]_{df} = \bot$$
$$[*]_{df} = *$$

36

$$[P\,Q]_{df} = [P]\,[Q]$$
$$[\lambda x.M]_{df} = \lambda x.[M]$$
$$[\Delta x.\,M]_{df} = \Delta x.\,[M]$$
$$[\Pi x{:}A.\,B]_{df} = \pi_{\tau(A)}\,[A]\,\lambda x.[B]$$

where π_C is a constant and τ is defined by

$$\tau(x) = x$$
$$\tau(*) = \omega$$
$$\tau(\bot) = \bot$$
$$\tau(P\,Q) = \tau(P)\,\tau(Q)$$
$$\tau(\lambda x.M) = \tau(M)$$
$$\tau(\Pi x{:}A.\,B) = \tau(A) \to \tau(B)$$

The translation $[.]_{df}$ may be used to prove strong normalization of $\lambda\Delta P$.

7 Classical Pure Type Systems as Logics

One central goal in our work is to extend the Curry-Howard isomorphism to classical λ-calculi and natural deduction systems for classical logic. The isomorphism involves a logic L and a type theory T –with a notion of proposition– and, in its strongest form, establishes a correspondence between:

(i) inhabited propositions in T and provable formulae in L;

(ii) proof-terms in T and proofs L;

(iii) normalization in T and cut-elimination in L.

Our notion of classical pure type system has been defined so as to preserve this three-fold correspondence with classical natural deduction as introduced by Prawitz [80]: classical logic is introduced via a double negation rule and reduction for the classical operator, which makes applications of double negation atomic, is closely related to Prawitz's original rule. In fact, his reduction relation makes all instances of double negation atomic while ours only makes atomic those instances of double negation whose conclusion is used as the function of an application rule –the reason for not using Prawitz's rule is that it is label-sensitive, see Subsection 8 for an explanation. Nevertheless, we shall not delve into the process of establishing the correspondence formally. There are several reasons for not pursuing this line of work: most importantly, the correspondence becomes clear as one works with classical pure type systems. Besides, as emerges from [30], stating the correspondence, let alone prove it, is long and tedious: it requires the introduction of logics as formal systems and of intermediate formal systems which arise as hybrid combinations between logics and type systems. Finally, the whole process requires great technical skill but does not improve our understanding of type systems.[7] Therefore we

---

[7] The studies by Tonino and Fujita [94] and Geuvers [30], in which the intrinsic technicalities of the Curry-Howard isomorphism are addressed, are of genuine interest. There is however

choose not to establish the Curry-Howard isomorphism formally and refer the interested reader to [30] for a thourough description of the proof techniques involved in this process.

Remark 7.1 *Of course encoding classical logic as an intuionistic context allows to derive the first and second parts of the isomorphism for the impredicative systems of the classical L-cube from the first and second parts of the isomorphism for the impredicative systems of the constructive L-cube.*

7.1 Impredicative specifications

The next definition is inspired from [20].

Definition 7.2 *A logical specification* $S = (S, \mathsf{Prop}, A, R)$ *is impredicative if there exists* $s \in S$ *s.t.* $(\mathsf{Prop}, s) \in A$ *and* $(s, \mathsf{Prop}) \in R$.

Impredicative specifications permit quantification over the universe of propositions. Impredicativity may be used to encode classical logic.

Lemma 7.3 *Let* S *be an impredicative and proof-irrelevant logical specification. Moreover let* $\Delta \equiv \bot : Prop$, $H : \Pi A : \mathsf{Prop}.\neg\neg A \to A$. *Then for every judgement* (Γ, M, A),

$$\text{there exists } M \text{ s.t. } \Gamma \vdash M : A \quad \Leftrightarrow \quad \text{there exists } N \text{ s.t. } \Delta, \Gamma \vdash^J N : A$$

Proof. To obtain N from M, replace recursively each subterm of the form $\Delta x : \neg A.\ P$ by $H\ A\ (\lambda x : \neg A.\ P')$ where P' has been obtained from P by the same process. To obtain M from N, replace each occurrence of H by

$$\lambda A : \mathsf{Prop}.\ \lambda y : \neg\neg A.\ \Delta x : \neg A.\ y\ x$$

$\square$

Note that one can also encode $\bot$ as $\Pi x : \mathsf{Prop}.\ x$.

7.2 The formulae-as-sets embedding

The formulae-as-sets embedding establishes the existence of a natural translation of the systems of the L-cube to the systems of the λ-cube. Technically, the translation is achieved through the notion of morphism of specifications.

Definition 7.4 *Let* $S_1 = (S_1, \mathsf{Prop}, A_1, R_1)$ *and* $S_2 = (S_2, \mathsf{Prop}, A_2, R_2)$ *be two logical specifications. A (set-theoretic) map* $H : S_1 \to S_2$ *is a morphism of logical specifications if for all* $s, s', s'' \in S_1$

- $H(\mathsf{Prop}) = \mathsf{Prop}$
- $(s, s') \in A_1 \quad \Rightarrow \quad (H\ s, H\ s') \in A_2$

---

little interest repeating them for the classical variants of the logics they consider.

- $(s, s', s'') \in R_1 \quad \Rightarrow \quad (H\ s, H\ s', H\ s'') \in R_2$

The formulae-as-sets embedding may now be described as a morphism of specifications from a system of the L-cube to its corresponding system in the λ-cube –as described in Figure 5.

Definition 7.5 *The formulae-as-sets embedding is the morphism of specifications $|.|$ given by:*

$$| *^p | = | *^s | = *$$
$$|\square^p| = |\square^s| = \square$$

The formulae-as-sets embedding can be extended to –both PTS and CPTS–pseudo-terms in an obvious way. The embedding is sound but incomplete.

Proposition 7.6 (Non-conservativity of the formulae-as-sets embedding)
The formulae-as-sets embedding is not complete for dependent systems of the classical logic cube (i.e. $\lambda\Delta PRED$, $\lambda\Delta PRED2$, $\lambda\Delta PRED\underline{\omega}$ and $\lambda\Delta PRED\omega$).

Proof. The following example is inspired from [30]. Let

$$\Gamma \equiv A : Set, P : A \to \mathsf{Prop}, \phi : Prop$$
$$\psi \equiv \neg(\Pi x{:}A. \neg\ P\ x) \to (A \to \phi) \to \phi$$

Then there is no M s.t. $\Gamma \vdash_{\lambda\Delta PRED\omega} M : \psi$ but

$$|\Gamma| \vdash_{\lambda\Delta P} \lambda p{:}\neg(\Pi x{:}A. \neg\ P\ x). \lambda q{:}A \to \phi. \Delta z{:}\neg\phi.\ p\ (\lambda a{:}A.\ z\ (q\ a)) : |\psi|$$

$\square$

Independently, one can study conservativity between systems of the classical λ- and L-cubes. Such a study for the PTS case may be found e.g. in [13,30]. Using techniques from [30], one proves

Theorem 7.7 *Let $S_1 \subseteq S_2$ be two systems of the classical λ-cube. Then S_2 is conservative over S_1 unless $S_2 = \lambda\Delta C$ and $S_1 = \lambda\Delta P2$.*

7.3 The Classical Calculus of Constructions is proof-relevant

Earlier work by Berardi, Coquand, Pottinger and Seldin has shown that classical logic is unexpectedly powerful in the Calculus of Constructions. For example, one can prove that classical logic and the axiom of descriptions imply proof-irrelevance. Formally, there exists a term P s.t. the following judgement is derivable in λC:

$$H : \mathsf{CL}, H' : \mathsf{AD}_m, H'' : \mathsf{AD}_c \vdash^J P : \mathsf{PI}$$

where CL formalizes classical logic, AD_m and AD_c formalizes the axiom of descriptions and PI formalizes the principle of proof-irrelevance, i.e.

$$\mathsf{CL} \equiv \Pi A{:}\, {*}.\, (\neg\neg A) \to A$$
$$\mathsf{AD}_m \equiv \Pi A{:}\, {*}.\, \Pi P{:} A \to {*}.\, (\exists! a : A.\ P\ a) \to A$$
$$\mathsf{AD}_c \equiv \Pi A{:}\, {*}.\, \Pi P{:} A \to {*}.\, \Pi z{:} (\exists! a : A.\ P\ a).\ P\ (\mathsf{AD}_m\ A\ P\ z)$$
$$\mathsf{PI} \equiv \Pi A{:}\, {*}.\, \Pi x, y{:} A.\ x =_A y$$

where $=_A$ is *Leibniz equality* on A and $\exists!$ is the unique existence quantifier. It follows that in $\lambda\Delta C$, one can find a term Q s.t. the following judgement is derivable

$$H' : \mathsf{AD}_m, H'' : \mathsf{AD}_c \vdash Q : \mathsf{PI}$$

However, Seldin showed –using a weak normalization procedure closely related to ours– that one cannot deduce proof-irrelevance from classical logic in λC [88]. [8] We prove a stronger result.

Proposition 7.8 *There is no M s.t. $\vdash M : \mathsf{PI}$ in $\lambda\Delta C$.*

Proof. Without loss of generality, one may assume that M is a normal term of the form

$$\lambda A : {*}.\lambda x, y : A.\lambda P : A \to {*}.\lambda H : P\ x.N$$

with N of type $P\ y$. We show by case analysis that it is impossible. □

8 Discussion and related work

Our definition of classical pure type system represents one of the multiple possibilities for such a notion. In this section we discuss some of the motivations behind our choice and relate our $\lambda\Delta$-calculi to some of the alternatives found in the literature. Because of the nature of the paper, we only focus on syntactic issues. Categorical issues and the way they influence the design of a classical λ-calculus have been discussed by other authors elsewhere [42,69].

8.1 Classical natural deduction and classical λ-calculi

Our presentation of classical pure type systems is based on Prawitz's format for classical natural deduction. However there are many other formats which also inspired classical λ-calculi. We review some of these formats here; in order to constrain the discussion, we restrict ourselves to those formats where classical logic is forced by a rule –and not an axiom. The most conventional formats are obtained by extending intuitionistic natural deduction with one rule for one of the three formulae:

- excluded middle $A \vee \neg A$;
- double negation $\neg\neg A \to A$;
- Pierce's law $((A \to B) \to A) \to A$.

---

[8] Such a result was proved by model-theoretic means in [91].

Single-Conclusioned	Multi-Conclusioned
DN [34,82]	Act/Pass [14,39,72]
EM [24]	Intuitionistic Act/Pass [39,66]
PL [41,86]	Truly MC [95]
$\lambda\mu$[69]	
Symmetric [4]	

Fig. 11. CLASSICAL LAMBDA CALCULI AND NATURAL DEDUCTION FORMATS

Remarkably, all these formats and derived endemic formats such as the variant of Pierce's law

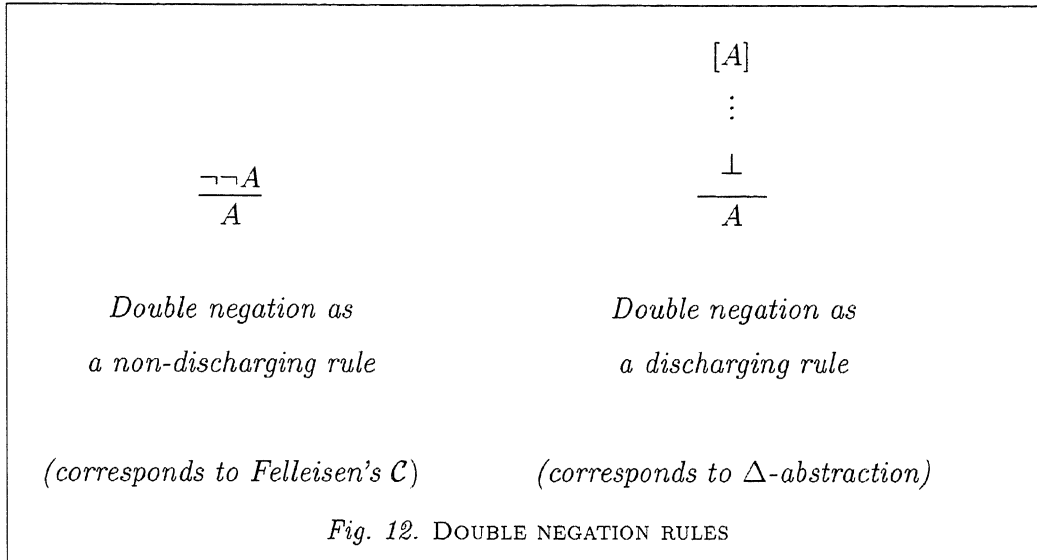$$((\forall B : \mathsf{Prop}.A \to B) \to A) \to A$$

have been used as a basis for classical λ-calculi.

In addition to these formats, one may conceive a variety of non-standard formats for classical natural deduction. One such format, proposed by L. Ong [69] as an explanation of the $\lambda\mu$-calculus, allows for two sorts of variables of type $\neg A$: continuation variables which may only be used as argument of an application rule and traditional variables which may be used in the usual way. The symmetric λ-calculus of Barbanera and Berardi provides another non-standard format of classical natural deduction in which negation is idempotent and implication is encoded [4].

All the natural deduction formats mentioned so far are single-conclusioned. In addition, one may find several formats that allow for multiple conclusions. Some formats, which stem from Parigot's $\lambda\mu$-calculus and linear logic, allow for multiple conclusions by distinguishing between active and passive formulae. There are other, more radical formats, which do not impose any such distinction; e.g. A. Ungar [95] has recently proposed an intriguing multi-conclusioned natural deduction system for classical natural deduction.

Each format for classical natural deduction can potentially yield one –or more– classical λ-calculus. Figure 11 attempts to classify the existing classical lambda calculi w.r.t. their corresponding format of classical natural deduction. Some of the calculi mentioned in Figure 11 are calculi of explicit substitutions but this fact is ignored for the sake of simplicity. From the point of view of classical pure type systems, classical λ-calculi based on single-conclusioned seem amenable to generalization. In contrast, it is unclear whether classical λ-calculi based on multi-conclusioned systems could be used for proof-relevant specifications.

Remark 8.1 *One could refine this classification further by distinguishing between those formats in which the classical rule is a discharging rule and those*

41

$$[A]$$
$$\vdots$$

$$\frac{\neg\neg A}{A} \qquad\qquad \frac{\bot}{A}$$

Double negation as   *Double negation as*

a non-discharging rule   *a discharging rule*

(corresponds to Felleisen's C)   *(corresponds to $\triangle$-abstraction)*

Fig. 12. DOUBLE NEGATION RULES

in which it is not; the difference between the two formats is illustrated in Figure 12. Probably such a distinction has little impact on the theory of classical λ-calculi and it is therefore not used here.

8.2 Reduction rules for classical natural deduction

Classical λ-calculi provide a computational analysis of classical logic by treating the classical operators as computationally meaningful. Many sets of reduction rules may be found in the literature:

- reduction rules inspired from classical proof theory and classical cut-elimination proofs;

- reduction rules inspired from programming languages and evaluation rules for control operators.

It is impossible to discuss here all the sets of reduction rules which can be found in the literature. However we find it instructive to consider some criteria according to which reduction rules may be classified. We list five fairly general such criteria concerned with the applicability of the rule.

- Local vs. global: the former only manipulate terms whereas the latter may manipulate contexts. The latter are typically found in programming languages; for example, the evaluation rules below are taken from [28,29]:

$$E[\mathcal{C}(M)] \to M(\lambda x.\mathcal{A}E[x])$$
$$E[\mathcal{A}(M)] \to M$$

The major problem with such rules is that they complicate the meta-theory of the system quite significantly. For example, it is unclear whether systems with such reduction rules are strongly normalising. Moreover, T. Coquand has shown us that global rules can fail to have the Subject Reduction prop-

erty for systems of dependent types.

- Compatible vs. context-sensitive: the former apply in an arbitrary context whereas the latter may be restricted to specific contexts. An example of such rule is found in [29], where the applicability of the rule $(\Delta_\perp)$ below is restricted to the empty context. In Felleisen's terminology, this is a top-level rule.

$$\Delta x{:}\neg\perp. M \quad \rightarrow \quad M\{x := \lambda z{:}\perp. z\} \qquad (\Delta_\perp)$$

There does not seem to be any problem with using "well-behaved" context-sensitive reduction rules. However, compatible reduction rules fit in the existing formats of higher-order rewriting – see [49] for a survey– whereas the study of context-sensitive higher-order rewriting systems has just begun [45].

- Substitutive vs. non-substitutive: the former apply to arbitrary instances whereas the latter may apply only for specific instances. An example of such rule is (Δ_v) below, which is needed to simulate the simply typed $\lambda\mu$-calculus in $\lambda\Delta \rightarrow$.

$$y\,(\Delta x{:}\neg A. M) \quad \rightarrow \quad M\{x := y\} \quad \text{if } y \in V \text{ and } y : \neg A \qquad (\Delta_v)$$

The main disadvantage of non-substitutive rules is that they may cause the failure of the substitution lemma –if equality is not substitutive– and hence of the subject reduction property.

- Type-insensitive vs. type-sensitive: the former apply without restrictions whereas the latter may apply only under specific typing assumptions. An example of such rule is (Δ_v). The main disadvantage of type-sensitive rules is that they cannot be used in the conversion rule. Otherwise reduction would depend on typing; this may create a vicious circle in the definition of the system as typing already depends on reduction –in fact equality– through the conversion rule.

- Label-insensitive vs. label-sensitive: the former apply independently of the domains of λ- and Δ-abstractions whereas the latter may only apply for some domains. The rule $(\Delta_\perp)$ and Prawitz's original reduction rule (Δ') below –with $C \equiv \Pi v{:}A. B$– provide examples of the latter.

$$\Delta x : \neg C. M \rightarrow \lambda v : A.\Delta y : \neg B. M\{x := \lambda w : C. y\,(w\,v)\} \qquad (\Delta')$$

The main disadvantage of label-sensitive reduction rules is that their domain-free variants are not always meaningful. For example, the domain-free variant of (Δ') is not normalising and does not have the subject reduction property.

In the simply typed context, all the above combinations may be envisaged. In the context of classical pure type systems, the situation is radically different. In order for a notion of classical pure type system to have a reasonable theory,

the reduction relation must satisfy several basic properties which are often violated by classical λ-calculi: compatibility, substitutivity, type-insensitivity and context-insensitivity, type-insensitivity and confluence. The latter is needed to prove the Key Lemma, which is in turn crucial in the proof of subject reduction. Of course, it has been continuously argued that reduction rules for a classical operator should not be confluent and in fact should not have the unique normal form property. However, this would lead to inconsistent calculi because of the conversion rules.

This leads us to distinguish between the notions of *minor rule* and *principal rule*: a principal rule determines the notion of computational equality whereas a minor rule does not. In other words, the proviso in the conversion rule should be $A =_{\mathcal{P}} B$ where $\mathcal{P}$ is the union of the principal rules. Such a distinction is justified conceptually and pragmatically: some reduction rules, especially those involving a non-deterministic choice, do not make sense as equalities and would cause the inconsistency of classical pure type systems if considered as a principal rule. On the other hand, these rules have a neat operational semantics and are useful in several applications. An example of such a rule is Δ_η^+ which has an obvious interpretation in the catch/throw paradigm and is useful for extracting a witness from a classical proof:

$$\Delta x{:}\neg A.\, C[x\, M] \to M \qquad \text{if } FV(M) \subseteq FV(\Delta x{:}\neg A.\, C[x\, M]) \qquad (\Delta_\eta^+)$$

The distinction between principal and minor rules allow us to combine the best of both worlds: logical consistency and computational power. The pragmatics and implications of the principal/minor rules distinction is left for future work. We simply close this subsection by stating the following result:

Theorem 8.2 *For every system of the classical λ- cube,*

(i) $\Gamma \vdash M : A,\ M \to_{\beta\Delta + \Delta_\perp} N \quad \Rightarrow \quad \Gamma \vdash N : A$

(ii) $\Gamma \vdash M : A \quad \Rightarrow \quad M \text{ is } \beta\Delta'\Delta_\eta + \Delta_\perp\text{-strongly normalizing}$

8.3 Related work

As mentioned in the introduction, the existing classical λ-calculi are proof-irrelevant, which makes it somewhat difficult to relate our calculi to existing ones –the main novelty of the paper is to consider double negation for proof-relevant systems and it seems therefore inappropriate to devote considerable attention to $\lambda\Delta \to$. Instead, we conisder two existing calculi and show how they also give rise to notions of classical pure type system.

8.3.1 The $\lambda\mu$-calculus

The $\lambda\mu$-calculus by Parigot is one of the most established classical λ-calculi. Unlike our calculi, the $\lambda\mu$-calculus provides an explicit treatment of continuations by distinguishing between names which are bound by continuation μ-abstractions and variables which are bound by λ-abstractions. Parigot's

original formulation uses multi-conclusioned sequents and cannot easily be generalised. Recently, Bierman and Ong have defined variants of the $\lambda\mu$-calculus. Bierman's variant is multic-onclusioned and it is unclear whether it can be generalised into a notion of classical pure type system. In contrast, Ong's variant is single-conclusioned and may be generalised into a notion of classical pure type systems. This is the purpose of the next definition. Note that the distinction between names and variables is handled by the sorts of the type system.

Definition 8.3

- A continuation-based *logical specification is a logical specification* S *with a distinguished sort of continuations* Cont.

- *The set of pseudo-terms is given by the abstract syntax:*

$$\mathcal{T} = V \mid S \mid \bot \mid \mathcal{T}\,\mathcal{T} \mid \lambda V{:}\mathcal{T}.\,\mathcal{T} \mid \Pi V{:}\mathcal{T}.\,\mathcal{T} \mid \mu V : \mathcal{T}.\mathcal{T} \mid \mathcal{T} \bullet \mathcal{T} \mid cont\ \mathcal{T}$$

- *The notion of reduction* μ *is given by the contractions:*

$$(\mu x : \text{cont}\ (\Pi v{:}A.\ B).M)\ N \to \mu y : cont\ (B\{v := N\}).M\{(x \bullet u) := y \bullet (u\ N)\}$$

$$\mu x : \text{cont}\ A.x \bullet M \to M \qquad \textit{if } x \notin FV(M)$$

$$x \bullet (\mu y : \text{cont}\ A.M) \to M\{y := x\}$$

- *The derivation rules are those of Table 13.*

Our notion of reduction are inspired by [71,72] rather than by [69], mostly because Ong's rules are not closed under substitution. In [69], Ong provides back and forth translations between $\lambda\Delta \to$ and $\lambda\mu \to$. One can define similar translations for an arbitrary specification S. The properties of the translations and more generally of this notion of classical pure type system will appear somewhere else.

8.3.2 The λ_Δ-calculus

The λ_Δ-calculus was introduced by Rehof and Sørensen [82]. It is a call-by-name calculus, closely related to classical natural deduction and with the ability to capture without any simulation the catch/throw mechanism. It is possible to generalise the $\lambda\Delta$-calculus to an arbitrary logical specification S. This is the purpose of the following definition –note that our notion of reduction below is slightly stronger than the one of [82].

Definition 8.4 *Let* Δ^+ *be the notion of reduction* $\Delta \cup \Delta_1 \cup \Delta_2$ *with*

$$\Delta x.x\ M \to_{\Delta_1} M \qquad\qquad \textit{if } x \notin FV(M)$$

$$\Delta x.x\ (\Delta y.M) \to_{\Delta_2} \Delta y.M\{x := y\}$$

(axiom-$\perp$) $<> \vdash \perp : \mathsf{Prop}$

(axiom) $<> \vdash s_1 : s_2$ if $(s_1, s_2) \in A$

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

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

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

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

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

(continuation) $$\frac{\Gamma \vdash A : Prop}{\Gamma \vdash \mathsf{cont}\ A : \mathsf{Cont}}$$

(c-application) $$\frac{\Gamma \vdash x : \mathsf{cont}\ A \quad \Gamma \vdash a : A}{\Gamma \vdash x \bullet a : \perp}$$

(c-abstraction) $$\frac{\Gamma, x : \mathsf{cont}\ A \vdash b : \perp \quad \Gamma \vdash A : \mathsf{Prop}}{\Gamma \vdash \mu x : \neg A.b : A}$$

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

Fig. 13. DEDUCTIVE RULES FOR $\lambda\mu$-PURE TYPE SYSTEMS

The domain-free type system $\lambda_\Delta S$ is obtained by replacing $\beta\Delta$ by $\beta\Delta^+$ in the conversion rule.

All the properties of domain-free classical pure type systems, including the correctness of the CPS translation and strong normalisation by CPS translation, should extend to these calculi. This will be reported elsewhere.

However, there is a major problem in defining domain-specified versions of these calculi. The rule Δ_2 becomes non left-linear: this causes all the techniques to prove confluence and subject reduction of the system to fail. In contrast, there is no major problem with the Δ_1-reduction, even in the domain-specified case.

8.3.3 Werner's proof-irrelevant generalised pure type systems

In unpublished work [97], Werner studies a notion of proof-irrelevant classical pure type system. His reduction rules depend on typing and therefore his setting does not make sense in the proof-relevant case. A more detailed comparison between his framework and ours will appear in the full version of the paper.

9 Conclusion

In this paper, we introduced a framework for classical λ-calculi and proved that proof-relevant systems of this framework are well-behaved. Much work remains to be done. In particular, we are currently investigating:

- extensions to CPTSs of the Kreisel-Friedman theorem on the computational content of classical proofs;
- criteria for distinguishing between principal and minor rules;
- syntactic proofs of strong normalisation for proof-relevant CPTSs.

At a more general level, the appropriateness of CPTSs as a foundation for classical theorem proving and program extraction should be investigated. Finally, a systematic comparison of the existing simply typed calssical λ-calculi would bring a much needed clarification to the area.

References

[1] S. Abramsky, D. Gabbay, and T. Maibaum, editors. *Handbook of Logic in Computer Science*. Oxford Science Publications, 1992.

[2] A. Appel. *Compiling with Continuations*. Cambridge University Press, 1992.

[3] F. Barbanera and S. Berardi. A strong normalization result for classical logic. *Annals of Pure and Applied Logic*, 76(2):99–116, December 1995.

[4] F. Barbanera and S. Berardi. A symmetric lambda calculus for classical program extraction. *Information and Computation*, 125(2):103–117, March 1996.

[5] H. Barendregt. Introduction to Generalised Type Systems. *J. Functional Programming*, 1(2):125–154, April 1991.

[6] H. Barendregt. Lambda calculi with types. In Abramsky et al. [1], pages 117–309. Volume 2.

[7] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, revised edition, 1984.

[8] B. Barras, S. Boutin, C. Cornes, J. Courant, J-C. Filliatre, E. Gimenez, H. Herbelin, G. Huet, C. Muñoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saibi, and B. Werner. The Coq proof assistant user's guide. Version 6.1. Technical report, INRIA – Rocquencourt, December 1996. Available by ftp from ftp.inria.fr along with the implementation.

[9] G. Barthe. Extensions of pure type systems. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Proceedings of TLCA'95*, volume 902 of *Lecture Notes in Computer Science*, pages 16–31. Springer-Verlag, April 1995.

[10] G. Barthe, J. Hatcliff, and M.H. Sørensen. CPS-translation and applications: the cube and beyond. In O. Danvy, editor, *Proceedings of the Second ACM SIGPLAN Workshop on Continuations*, number NS-96-13 in BRICS Notes, pages 4/1–4/31, 1996.

[11] G. Barthe and M.H. Sørensen. Domain-free pure type systems. In S. Adian and A. Nerode, editors, *Proceedings of LFCS'97*, Lecture Notes in Computer Science, 1997. To appear.

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

[13] S. Berardi. *Type dependence and Constructive Mathematics*. PhD thesis, University of Torino, 1990.

[14] G.M. Bierman. Towards a classical linear λ-calculus. In *Proceedings of Tokyo Conference on Linear Logic*, volume 3 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1996.

[15] L.E.J. Brouwer. Intuïtionistische splitsing van mathematische grondbegrippen. *Nederl. Akad. Wetensch. Verslagen*, 32:877–880, 1923.

[16] R. Constable. Constructive mathematics and automatic program writers. In *Procceddings of the IFIP Congress*, pages 229–233, Ljubljana, 1971.

[17] R. Constable. programs as proofs: A synopsis. *Information Processing Letters*, 16(3):105–112, 1983.

[18] R. Constable and C.R. Murthy. Finding computational contents in classical proofs. In G. Huet and G. Plotkin, editors, *Proceedings of the First Workshop on Logical Frameworks*, pages 341–362. Cambridge University Press, 1990.

[19] R.L. Constable, S.F. Allen, H.M. Bromley, W.R. Cleaveland, J.F. Cremer, R.W. Harper, D.J. Howe, T.B. Knoblock, N.P. Mendler, P. Panangaden, J.T. Sasaki, and S.F. Smith. *Implementing Mathematics with the NuPrl Development System*. Prentice-Hall, Inc., 1986.

[20] T. Coquand and H. Herbelin. A-translation and looping combinators in pure type systems. *Journal of Functional Programming*, 4(1):77–88, 1994.

[21] H.B. Curry and R Feys. *Combinatory Logic*. North-Holland, 1958.

[22] O. Danvy, editor. *Proceedings of the Second ACM SIGPLAN Workshop on Continuations*, number NS-96-13 in BRICS Notes, 1996.

[23] P. de Groote. A CPS-translation of the $\lambda\mu$-calculus. In S. Tison, editor, *Proceedings of CAAP'94*, volume 787 of *Lecture Notes in Computer Science*, pages 85–99. Springer-Verlag, 1994.

[24] P. de Groote. A simple calculus of exception handling. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Typed Lambda Calculus and Applications*, volume 902 of *Lecture Notes in Computer Science*, pages 201–215. Springer-Verlag, 1995.

[25] P. de Groote, editor. *The Curry-Howard isomorphism*, volume 8 of *Cahiers du centre de logique*. Universite catholique de Louvain, 1995.

[26] G. Dowek, G. Huet, and B. Werner, On the existence of long $\beta\eta$-normal forms in the cube, In H. Geuvers, editor, *Informal Proceedings of TYPES'93*, pages 115–130, 1993. Available from http://www.dcs.ed.ac.uk/lfcsinfo/research/types-bra/proc/index.html.

[27] B.F. Duba, R. Harper, and D. MacQueen. Typing first-class continuations in ML. In *Conference Record of the Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1991.

[28] M. Felleisen. *The Calculi of λ_v-CS Conversion: A Syntactic theory of Control and State in Imperative Higher Order programming Languages*. PhD thesis, Indiana University, 1987.

[29] M. Felleisen, D. Friedman, E. Kohlbecker, and B. Duba. A syntactic theory of sequential control. *Theoretical Computer Science*, 52(3):205–237, 1987.

[30] H. Geuvers. *Logics and Type Systems*. PhD thesis, University of Nijmegen, 1993.

[31] H. Geuvers and M.-J. Nederhof. A modular proof of strong normalisation for the Calculus of Constructions. *Journal of Functional Programming*, 1:155–189, 1991.

[32] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Number 7 in Tracts in Theoretical Computer Science. Cambridge University Press, 1989.

[33] M.J.C. Gordon and T.F. Melham, editors. *Introduction to HOL: A theorem proving environment for higher-order logic*. Cambridge University Press, 1993.

[34] T.G. Griffin. A formulae-as-types notion of control. In *Conference Record of the Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 47–58. ACM Press, 1990.

[35] T.G. Griffin. Logical interpretations and computational simulations. Manuscript, 1992.

[36] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *Proceedings of LICS'87*, pages 194–204. IEEE Computer Society Press, 1987.

[37] R. Harper and M. Lillibridge. Explicit polymorphism and CPS conversion. In *Conference Record of the Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 206–219. ACM Press, 1993.

[38] R. Harper and M. Lillibridge. Polymorphic type assignment and CPS conversion. *LISP and Symbolic Computation*, 6:361–380, 1993.

[39] H. Herbelin. *Séquents qu'on calcule*. PhD thesis, Université Paris 7, 1995.

[40] A. Heyting, *Mathematische Grundlagenforschung. Intuitionismus. Beweistheorie*, Springer, 1934.

[41] S. Hirokawa, Y. Komori, and I. Takeuti. A reduction rule for the Peirce formula. Manuscript, 1994.

[42] M. Hofmann. Sound and complete axiomatisations of call-by-balue control operators. *Mathematical Structures in Computer Science*, 5(4):461–482, December 1995.

[43] W. Howard. The formulae-as-types notion of construction. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press Limited, 1980.

[44] B. Jacobs. Categorical logic and type theory. Book. In preparation, 199x.

[45] Z. Khasidashvili and V. van Oostrom. Context-sensitive conditional expression reduction systems. *Elsevier, Electronic Notes in Theoretical Computer Science*, 2, 1995. Available at http://www.elsevier/nl/mcs/tcs/pc/volume2.htm.

[46] S.C. Kleene. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic*, 10:109–124, 1945.

[47] S.C. Kleene. *Introduction to Metamathematics*. Van Nostrand, 1952.

[48] J.W. Klop. Term-rewriting systems. In Abramsky et al. [1], pages 1–116. Volume 2.

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

[50] A. Kolmogorov. Sur le principe de tertium non datur. *Matématicéskij Sbornik*, 32:646–667, 1925. English translation in [96].

[51] A. Kolmogorov. Zur Deutung der intuitionistischen Logik. *Mathematische Zeitschrift*, 35:58–65, 1932.

[52] J. Lambek. Deductive systems and categories I. Syntactic calculus and residuated categories. *Mathematical Systems Theory*, 2(4):287–318, 1968.

[53] J. Lambek. From λ-calculus to cartesian closed categories. In J.R. Hindley and J.P. Seldin, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 375–402. Academic Press, 1980.

[54] J. Lambek and P.J. Scott. *Introduction to Higher-Order Categorical Logic*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1986.

[55] F.W. Lawvere. Ajointness in foundations. *Dialectica*, 23:281–296, 1969.

[56] Z. Luo. *Computation and Reasoning: A Type Theory for Computer Science*. Number 11 in International Series of Monographs on Computer Science. Oxford University Press, 1994.

[57] Z. Luo and R. Pollack. LEGO proof development system: User's manual. Technical Report ECS-LFCS-92-211, LFCS, Computer Science Dept., University of Edinburgh, May 1992.

[58] L. Magnusson. *The implementation of ALF: a proof editor based on Martin-Löf's monomorphic type theory with explicit substitution*. PhD thesis, Department of Computer Science, Chalmers University, 1994.

[59] C. Mann. The connection between equivalences of proofs and cartesian closed categories. *Proceedings of the London Mathematical Society*, 31:289–310, 1975.

[60] P. Martin-Löf. *Intuitionistic Type Theory*, volume 1 of *Studies in Proof Theory*. Bibliopolis, Naples, 1984.

[61] A.R. Meyer and M. Wand. Continuation semantics in typed lambda-calculi (summary). In R. Parikh, editor, *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 219–224. Springer-Verlag, 1985.

[62] C. Murthy. *Extracting Constructive Contents from Classical Proofs*. PhD thesis, Cornell University, 1990.

[63] C. Murthy. An evaluation semantics for classical proofs. In *Logic in Computer Science*, 1991.

[64] C.R. Murthy. A computational analysis of Girard's translation and LC. In *Logic in Computer Science*, 1992.

[65] C.R. Murthy. Control operators, hierachies, and pseudo-classical type systems: A-translation at work. In *ACM SIGPLAN Workshop on Continuations*, 1992.

[66] H. Nakano. A constructive logic behind the catch and throw mechanism. *Annals of Pure and Applied Logic*, 69(2–3):269–301, October 1994.

[67] R. Nederpelt, H. Geuvers, and R. de Vrijer, editors. *Selected papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1994.

[68] B. Nordström, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory. An Introduction*. Number 7 in International Series of Monographs on Computer Science. Oxford University Press, 1990.

[69] C.-H. L. Ong. A semantic view of classical proofs: Type-theoretic, categorical, and denotational characterizations. In *Logic in Computer Science*, 1996. To appear.

[70] C.-H.L. Ong and C.A. Stewart. A Curry-Howard Foundation for functional computation with control. In *Proceedings of POPL '97*, pages ??–?? ACM Press, 1997.

[71] M. Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In *International Conference on Logic Programming and Automated Reasoning*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer-Verlag, 1992.

[72] M. Parigot. Strong normalization for second order classical natural deduction. In *Proceedings of LICS'93*, pages 39–46. IEEE Computer Society Press, 1993.

[73] F. Pfenning. Elf: a meta-language for deductive systems. In A. Bundy, editor, *Proceedings of CADE-12*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 811–815. Springer-Verlag, 1994.

[74] A. Pitts. Categorical logic. Technical Report 367, University of Cambridge Computer Laboratory, May 1995.

[75] A. Pitts and P. Dybjer, editors. *Semantics and Logics of Computation*. Publications of the Isaac Newton Institute. Cambridge University Press, 1997.

[76] G. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[77] R. Pollack. *The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, 1994. Available by anonymous ftp from `ftp.cs.chalmers.se` in directory `pub/users/pollack`.

[78] G. Pottinger. Normalization as a homomorphic image of cut-elimination. *Archive for Mathematical Logic*, 12:323–357, 1977.

[79] G. Pottinger. Strong normalisation for terms of the theory of constructions. Technical Report TR 11-7, Odissey Research Associates, 1987.

[80] D. Prawitz. *Natural Deduction: A proof theoretical study.* Almquist & Wiksell, 1965.

[81] D. Prawitz. Ideas and results of proof theory. In J.E. Fenstad, editor, *The 2nd Scandinavian Logical Symposium*, pages 235–307. North-Holland, 1970.

[82] N.J. Rehof and M.H. Sørensen. The λ_Δ calculus. In M. Hagiya and J. Mitchell, editors, *Proceedings of TACS'94*, volume 789 of *Lecture Notes in Computer Science*, pages 516–542. Springer-Verlag, 1994.

[83] J.C. Reynolds. The discoveries of continuations. *LISP and Symbolic Computation*, 6:233–248, 1993.

[84] E. Ritter, D. Pym, and L. Wallen. Proof-terms for classical and intuitionistic logic (extended abstract). In M. McRobbie and J. Slaney, editors, *Proceeedings of CADE'96*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages ??–??, 1996.

[85] R. Seely. Modelling computations: A 2-categorical framework. In *Proceedings of LICS'87*, pages 65–71. IEEE Computer Society Press, 1987.

[86] J.P. Seldin. On the proof theory of the intermediate logic MH. *Journal of Symbolic Logic*, 51(3):626–647, 1986.

[87] J.P. Seldin. Normalization and excluded middle. *Studia Logica*, XLVIII(2):193–217, 1989.

[88] J.P. Seldin. On the proof theory of Coquand's calculus of constructions. *Annals of Pure and Applied Logic*, 83(1):23–101, 6 January 1997.

[89] N. Shankar, S. Owre, and J.M. Rushby. *The PVS Proof Checker: A Reference Manual.* Computer Science Laboratory, SRI International, Menlo Park, CA, February 1993. A new edition for PVS Version 2 is expected in late 1996.

[90] G. Stålmarck. Normalization theorems for full first order classical natural deduction. *Journal of Symbolic Logic*, 56(1):129–149, 1991.

[91] M. Stefanova and H. Geuvers. A simple set-theoretic semantics for the Calculus of Constructions. In S. Berardi and M. Coppo, editors, *Proceedings of TYPES'95*, volume 1158 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[92] Th. Streicher. *Semantics of Type Theory. Correctness, Completeness and Independence results.* Progress in Theoretical Computer Science. Birkhäuser, 1991.

[93] J. Terlouw. Strong normalization in type systems: a model theoretic approach. *Annals of Pure and Applied Logic*, 73(1):53–78, May 1995.

[94] H. Tonino and K. Fujita. On the adequacy of representing higher order intuitionistic logic as a pure type system. *Annals of Pure and Applied Logic*, 57(3):251–276, June 1992.

[95] A. Ungar. *Normalization, cut-elimination and the theory of proofs*. Number 28 in Lecture Notes. Center for the Study of Language and Information, Stanford, CA., 1992.

[96] J. van Heijenoort, editor. *From Frege to Gödel: A Source-Book in Mathematical Logic, 1879–1931*. Harvard University Press, 1967.

[97] B. Werner. Continuations, evaluation styles and types systems. Manuscript, 1992.

[98] J. Zucker. The correspondence between cut-elimination and normalization. *Archive for Mathematical Logic*, 7:113–155, 1974.

(axiom) $\langle \rangle \vdash^J s_1 : s_2$ if $(s_1, s_2) \in A$

(start) $$\frac{\Gamma \vdash^J A : s}{\Gamma, x : A \vdash^J x : A}$$ if $x \notin \Gamma$

(weakening) $$\frac{\Gamma \vdash^J A : B \quad \Gamma \vdash^J C : s}{\Gamma, x : C \vdash^J A : B}$$ if $x \notin \Gamma$

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

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

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

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

Fig. 14. Pure Type Systems

(axiom) $\qquad \langle\rangle \Vdash^J s_1 : s_2 \qquad\qquad\qquad\qquad$ if $(s_1, s_2) \in A$

(start) $\qquad \dfrac{\Gamma \Vdash^J A : s}{\Gamma, x : A \Vdash^J x : A} \qquad\qquad$ if $x \notin \Gamma$

(weakening) $\quad \dfrac{\Gamma \Vdash^J A : B \quad \Gamma \Vdash^J C : s}{\Gamma, x : C \Vdash^J A : B} \qquad$ if $x \notin \Gamma$

(product) $\qquad \dfrac{\Gamma \Vdash^J A : s_1 \quad \Gamma, x : A \Vdash^J B : s_2}{\Gamma \Vdash^J (\Pi x{:}A.\, B) : s_3} \qquad$ if $(s_1, s_2, s_3) \in R$

(application) $\quad \dfrac{\Gamma \Vdash^J F : (\Pi x{:}A.\, B) \quad \Gamma \Vdash^J a : A}{\Gamma \Vdash^J F\, a : B\{x := a\}}$

(abstraction) $\quad \dfrac{\Gamma, x : A \Vdash^J b : B \quad \Gamma \Vdash^J (\Pi x{:}A.\, B) : s}{\Gamma \Vdash^J \lambda x.b : \Pi x{:}A.\, B}$

(conversion) $\quad \dfrac{\Gamma \Vdash^J A : B \quad \Gamma \Vdash^J B : s \quad \Gamma \Vdash^J B' : s}{\Gamma \Vdash^J A : B'} \quad$ if $B =_\beta B'$

Fig. 15. DOMAIN-FREE PURE TYPE SYSTEMS