Bridging Curry and Church's typing style

Fairouz Kamareddine[a,*], Jonathan P. Seldin[b,**], J.B. Wells[a]

[a] *School of Maths and Computer Sc., Heriot-Watt Univ., Edinburgh, UK*
[b] *Maths and Computer Sc., Univ. of lethbridge, Canada*

Abstract

There are two versions of type assignment in the λ-calculus: Church-style, in which the type of each variable is fixed, and Curry-style (also called "domain free"), in which it is not. As an example, in Church-style typing, $\lambda_{x:A}.x$ is the identity function on type A, and it has type $A \to A$ but not $B \to B$ for a type B different from A. In Curry-style typing, $\lambda_x.x$ is a general identity function with type $C \to C$ for *every* type C. In this paper, we will show how to interpret in a Curry-style system every Pure Type System (PTS) in the Church-style without losing any typing information. We will also prove a kind of conservative extension result for this interpretation, a result which implies that for most consistent PTSs of the Church-style, the corresponding Curry-style system is consistent. We will then show how to interpret in a system of the Church-style (a modified PTS, stronger than a PTS) every PTS-like system in the Curry style.

Keywords: Church-style typing, Curry-style typing, domain-full typing, domain-free typing

*Corresponding author
**Corresponding author
Email addresses: fairouz@macs.hw.ac.uk (Fairouz Kamareddine), seldin@cs.uleth.ca (Jonathan P. Seldin)

1. Introduction

There are two main styles of type theory in λ -calculus: the Church-style, in which each abstraction indicates the type of the variable, as in

$$\lambda_{x:A}.M,$$

and the Curry-style, in which no such type is given:

$$\lambda_x.M.$$

These two styles of typing are often called the *domain-full* and the *domain-free* styles respectively. These styles are compared and discussed in [3].

Remark 1. Barthe and Sørensen [3] distinguish between domain-free systems, which they regard as Church-style systems with the types of the bound variables omitted, and what they think of as the Curry view, in which typing rules assign types to terms that already exist in the pure λ-calculus. In this they are following Barendregt [2, Definition 4.1.7], who identifies as the Curry-version of $\lambda 2$ a system in which the rules for $\forall$ are given as follows:

$$\text{($\forall$-elimination)} \qquad \frac{\Gamma \vdash M : (\forall\alpha.\sigma)}{\Gamma \vdash M : [\tau/\alpha]\sigma}$$

$$\text{($\forall$-introduction)} \qquad \frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : (\forall\alpha.\sigma)} \qquad \alpha \notin \mathsf{FV}(\Gamma)$$

But these two rules seem to be closer to the ideas of the intersection type systems than to any of the systems that interested Curry. As should be clear from [7, Chapter 14], Curry was basically interested in the system usually called $\lambda \to$, and the basic characteristic of his version is that $\lambda_x.x$ in λ-calculus and I in a system of combinators can have any type of the form $\alpha \to \alpha$, which Curry wrote $\mathsf{F}\alpha\alpha$. This is what Curry called *functionality*. He also suggested what he called *generalized functionality*, in which the constant F was replaced by G, where $\mathsf{G}\alpha\beta$ is the type we now write $(\Pi_{x:\alpha}.\beta x)$. Seldin treated a basic form of generalized functionality in Curry's style in his paper [20]. On the other hand, Church's typing is probably best exemplified by his simple type theory of [5],

which is characterized by the presence of the type of each bound variable, as in $\lambda_{x^\alpha}.M$. Hence, to be historically accurate, it is better to identify the Curry-style with domain-free type systems.

In a Curry-style system, there are terms like $\lambda_y.yy$ that have no types. But there is a sense in which this is also true in a Church-style system: $\lambda_{y:A}.yy$ is a perfectly good pseudoterm of a Church-style system, but it will not have a type in many of the usual systems. Perhaps the vocabulary used may disguise the similarity here: a pseudoterm in a Church-style system corresponds to a term in a Curry-style system.

There is one standard interpretation of a Church-style system in a corresponding Curry-style system: the function Erase, which simply deletes the domains from a formula, so that

$$\text{Erase}(\lambda_{x:A}.M) \equiv \lambda_x.M.$$

Erase has been used extensively to relate Church-style systems and Curry-style systems. For example, Erase and modifications of Erase are used by Steffen van Bakel et al [22] to compare Church-style PTSs (which they call typed systems) and Curry-style PTSs (which they call type assignment systems). But using Erase to interpret a Church-style PTS in a Curry-style PTS causes some type information to be lost. One might think that this information can be restored from the type $(\Pi_{x:A}.B)$ of an abstraction term $(\lambda_x.M)$ by mapping this to $(\lambda_{x:A}.M)$. But as is shown in [22, Example 3.5, Theorem 3.6], this is too simple and may not work properly for some systems.

For these reasons, we think there is a need for a method of relating the Church-style typing and the Curry-style typing that does not lose this type information.

In this paper, we propose to show how to interpret a system of each style in an appropriate system of the other without this kind of loss of typing information. In one direction, the direction from Church-style to Curry-style, the interpretation is defined by allowing an abstraction of the form $(\lambda_{x:A}.M)$ to be an abbreviation for a term of the Curry-style system, so the information

about the type of the bound variable in the λ-abstraction is not lost. This interpretation extends previous work with Garrel Pottinger [18],[1] which carried through the interpretation for three systems from the Barendregt cube: $\lambda\!\rightarrow$ [5], $\lambda 2$ [8, 19], λC [6], and its extension, the system ECC [14, 15].

In the other direction, the Church-style system into which the Curry-style PTS is interpreted is not a PTS, but is obtained from a PTS by the addition of a rule. The idea here is to provide a dummy type A to be the "domain" of a Curry-style abstraction term $(\lambda_x.M)$, so that the Church-style abstraction which interprets this Curry-style abstraction is $(\lambda_{x:\mathsf{A}}.M)$. This dummy type A does not have any sort as its type, so it can only play a very limited role in the Church-style system. But to make the interpretation work a rule must be added to the Church-style PTS to allow an inference from $\Gamma \vdash (\lambda_{x:B}.M) : (\Pi_{x:B}.C)$ to $\Gamma \vdash (\lambda_{x:\mathsf{A}}.M) : (\Pi_{x:B}.C)$. This rule corresponds in a sense to the fact that in the Curry-style system, the term interpreting $(\lambda_{x:B}.M)$ β-reduces to $(\lambda_x.M)$.

An earlier version of this paper, which was never published, is [21].

This article is divided as follows:

- In Section 2, we introduce the type free λ-calculus in both Church's and Curry's notations (systems $\mathcal{T}$ and $\mathcal{T}_c$) and we introduce 3 extra systems ($\mathcal{T}_c$ + Label, $\mathcal{T}_l$ and $\mathcal{T}'$) that will be used to study the interpretations between $\mathcal{T}$ and $\mathcal{T}_c$ with minimal loss of information. Reductions are introduced and shown to be confluent, and interpretations are given to establish bijective correspondences.

- In Section 3 we introduce typings into the various systems and establish important properties such as Generation, Subject Reduction, and Preservation of Types. The systems introduced include PTSs [2], DFPTSs [3] which cannot support faithful translations between Church's and Curry's notation, and 3 systems that support faithful translations: L-complete DFPTSs, lPTSs and l'PTSs.

[1] But the reader will not need knowledge of [18] to understand the present paper.

- In Section 4 we show that DFPTSs do not faithfully capture Church's typing.
- In Section 5 we study the conditions needed for a DFPTS to be able to capture Church's typing in $\mathcal{T}_c$ + Label. We find that these DFPTSs have to be L-complete and that although numerous L-complete PTSs exist, a PTS in Church's style cannot be interpreted in a DFPTS (Curry's style) if the original PTS did not obey these L-completeness conditions.
- In Section 6 we explain why lPTSs (which interpret $\mathcal{T}_l$) are better for capturing Church's typing since they require no restrictions on the type system.
- In Section 7 we give the l'PTSs which require restrictions but not as much as the DFPTSs.
- In Section 8 we study the unicity of types, classification and consistency lemmas.
- In Section 9 we give the interpretation in the reverse direction.
- In Section 10 we conclude and discuss future work/open questions.

The new interpretations and the various results are summarised in Figure 1. They differ in that:

- For the first, the syntax of the λ-calculus (terms in $\mathcal{T}_c$) is not extended but for this approach to work, the term $\mathsf{Label} \equiv \lambda uzx.zx$ needs to be typed and for this, we need to impose L-completeness conditions on the specifications by adding extra axioms in $\boldsymbol{A}$ and rules in $\boldsymbol{R}$.
- In the second approach the syntax is extended with terms of the form $lA(\lambda_x.b)$ and this does not require any extra axioms or rules. This is in line with [11, 12] where it was argued that for some functions (here the term l), it is enough to consider them with their full list of arguments (here A and $(\lambda_x.B)$) because these functions are never used without their full arguments so why put many conditions on the typing systems to type these functions on their own. In

Terms	r-reduction	$\vdash_r$ Rules	Name
$\mathcal{T}$	$(\lambda_{x:A}.B)C$ $\to_\beta B[x := C]$	$\vdash_\beta$ of Figures 2 and 3	PTSs [2]
$\mathcal{T}_c$ $[\mathcal{T}]^c \subset \mathcal{T}_c$	$(\lambda_x.B)C$ $\to_{\overline{\beta}} B[x := C]$ $A =_\beta B \overset{Lem\ 10}{\Longrightarrow}$ $[A]^c =_{\overline{\beta}} [B]^c$	$\vdash_{\overline{\beta}}$ of Figures 2 and 4 $\Gamma \vdash^{\mathcal{S}}_\beta A : B \overset{Lem\ 25}{\Longrightarrow} [\Gamma]^c \vdash^{\mathcal{S}}_{\overline{\beta}} [A]^c : [B]^c$ $[\Gamma]^c \vdash^{\mathcal{S}}_{\overline{\beta}} [A]^c : [B]^c \overset{Lem\ 25}{\not\Longrightarrow} \Gamma \vdash^{\mathcal{S}}_\beta A : B$	DFPTSs [3]
$\mathcal{T}_c$ + Label Label $\equiv$ $\lambda_{uzx}.zx$ $[\mathcal{T}]^L \subset \mathcal{T}_c$	$(\lambda_x.B)C$ $\to_{\overline{\beta}} B[x := C]$	$\vdash_{\overline{\beta}}$ of Figures 2 and 4 Label must be typeable $\Gamma \vdash^{\mathcal{S}}_\beta A : B \overset{Lem\ 39}{\Longleftrightarrow} [\Gamma]^L \vdash^{\mathcal{S}}_{\overline{\beta}} [A]^L : [B]^L$	DFPTS is L-complete
$\mathcal{T}_l$ $[\mathcal{T}]^l \subset \mathcal{T}_l$	$\underline{\beta} = l \cup \overline{\beta}$ where $lA(\lambda_x.b) \to_l \lambda_x.b$	$\vdash_{\underline{\beta}}$ of Figures 2, 4 and 5 $\Gamma \vdash^{\mathcal{S}}_\beta A : B \overset{Lem\ 40}{\Longleftrightarrow} [\Gamma]^l \vdash^{\mathcal{S}}_{\underline{\beta}} [A]^l : [B]^l$	lPTSs
$\mathcal{T}'$ $[\mathcal{T}]' \subset \mathcal{T}'$	$\beta' = l' \cup \overline{\beta}$ where $l'A(\lambda_x.b) \to_{l'} \lambda_x.b$	$\vdash_{\beta'}$ of Figures 2, 4 and 6 $\Gamma \vdash^{\mathcal{S}}_\beta A : B \overset{Lem\ 42}{\Longleftrightarrow} [\Gamma]' \vdash^{\mathcal{S}}_{\beta'} [A]' : [B]'$	l'PTSs l'-complete
$-^{\mathrm{Ch}}$: $\mathcal{T}_c \mapsto \mathcal{T}$		$\vdash_{\beta\mathsf{A}}$ of Figures 2, 3 and 10 $\Gamma \vdash^{\mathcal{S}}_{\overline{\beta}} M : B \overset{Lem\ 54}{\Longleftrightarrow}$ $\Gamma^{\mathrm{Ch}} \vdash^{\mathcal{S}}_{\beta\mathsf{A}} M^{\mathrm{Ch}} : B^{\mathrm{Ch}}$	

Figure 1: Systems studied in this paper

this case, we do not need any new axioms/rules to type $lA(\lambda_x.B)$ whereas for the first approach, we needed to impose L-completeness in order to type Label when in fact, we only need Label with its arguments A and $(\lambda_x.B)$. By adding extra axioms and rules, we go up in the hierarchy of types and hence lose nice properties such as decidability.

- The third approach attempts to study the middle grounds between the first and second. Here, we extend the syntax with terms of the form $l'A$ and in order to type $l'A(\lambda_x.B)$, we need to also type $l'A$. To do this, new axioms and rules need to be added but less than those needed for the first approach (compare L-completeness with l'-completeness).

2. Notions of reduction in Church's and Curry's notations

The basic idea of the interpretation from the Church syntax to that of Curry is due to Garrel Pottinger [16, §9], who proposed using a constant Label so that in the Curry-style syntax

$$(\lambda_{x:A}.M)$$

is an abbreviation for

$$\mathsf{Label}A(\lambda_x.M).$$

(Pottinger used "ϕ" for "Label".) By the analogy with β-reduction, we will want

$$\mathsf{Label}A(\lambda_x.M)N \twoheadrightarrow M[x := N].$$

This suggests that Label should have the reduction rule

$$\mathsf{Label}\, UZX \twoheadrightarrow ZX,$$

as proposed by Pottinger in [16, §9]. This would suggest, in turn, that we define Label as follows:

$$\mathsf{Label} \equiv \lambda_{uzx}.zx.$$

However, as we shall see, there are problems typing this definition. So we shall also look at some alternatives.

The next definition introduces a number of sets of terms. $\mathcal{T}$ is the set of terms written à la Church while $\mathcal{T}_c$ is the set of terms written à la Curry. Terms in $\mathcal{T}_l$ and $\mathcal{T}_{l'}$ are also terms written à la Curry, but instead of using Label to save the type A of x in $\lambda_{x:A}.B$, we let the built-in l and l' do the saving work. For each set of terms we introduce the sets of terms with one hole which will be used in the proofs.

Definition 2. [Terms and translations]

1. We let $\boldsymbol{S}$ be a set of *sorts* and $\mathcal{V}$ a countably infinite set of *variables*.
   We let s, s', s_1, etc. range over $\boldsymbol{S}$ and x, y, z, x_1, u, v, etc. range $\mathcal{V}$.
   We assume that $\boldsymbol{S} \cap \mathcal{V} = \emptyset$.
2. We define the set of terms $\mathcal{T}$ by: $\mathcal{T} ::= \boldsymbol{S} \mid \mathcal{V} \mid \lambda_{\mathcal{V}:\mathcal{T}}.\mathcal{T} \mid \Pi_{\mathcal{V}:\mathcal{T}}.\mathcal{T} \mid \mathcal{T}\mathcal{T}$.
   We define the set $\mathcal{C}$ of $\mathcal{T}$-terms with one hole by:
   $\mathcal{C} ::= \boxtimes \mid \lambda_{\mathcal{V}:\mathcal{C}}.\mathcal{T} \mid \lambda_{\mathcal{V}:\mathcal{T}}.\mathcal{C} \mid \Pi_{\mathcal{V}:\mathcal{C}}.\mathcal{T} \mid \Pi_{\mathcal{V}:\mathcal{T}}.\mathcal{C} \mid \mathcal{C}\mathcal{T} \mid \mathcal{T}\mathcal{C}$.
3. We define the set of terms $\mathcal{T}_c$ by: $\mathcal{T}_c ::= \boldsymbol{S} \mid \mathcal{V} \mid \lambda_{\mathcal{V}}.\mathcal{T}_c \mid \Pi_{\mathcal{V}:\mathcal{T}_c}.\mathcal{T}_c \mid \mathcal{T}_c\mathcal{T}_c$.
   We denote the term $\lambda_z.\lambda_x.\lambda_y.xy$ of $\mathcal{T}_c$ by Label.
   We define the set $\mathcal{C}_c$ of $\mathcal{T}_c$-terms with one hole by:
   $\mathcal{C}_c ::= \boxtimes \mid \lambda_{\mathcal{V}}.\mathcal{C}_c \mid \Pi_{\mathcal{V}:\mathcal{C}_c}.\mathcal{T}_c \mid \Pi_{\mathcal{V}:\mathcal{T}_c}.\mathcal{C}_c \mid \mathcal{C}_c\mathcal{T}_c \mid \mathcal{T}_c\mathcal{C}_c$.
4. We define the set of terms $\mathcal{T}_l$ by: $\mathcal{T}_l ::= \boldsymbol{S} \mid \mathcal{V} \mid \lambda_{\mathcal{V}}.\mathcal{T}_l \mid \Pi_{\mathcal{V}:\mathcal{T}_l}.\mathcal{T}_l \mid \mathcal{T}_l\mathcal{T}_l \mid l\mathcal{T}_l\mathcal{T}_l$.
   We define the set $\mathcal{C}_l$ of $\mathcal{T}_l$-terms with one hole by:
   $\mathcal{C}_l ::= \boxtimes \mid \lambda_{\mathcal{V}}.\mathcal{C}_l \mid \Pi_{\mathcal{V}:\mathcal{C}_l}.\mathcal{T}_l \mid \Pi_{\mathcal{V}:\mathcal{T}_l}.\mathcal{C}_l \mid \mathcal{C}_l\mathcal{T}_l \mid \mathcal{T}_l\mathcal{C}_l \mid l\mathcal{C}_l(\lambda_{\mathcal{V}}.\mathcal{T}_l) \mid l\mathcal{T}_l(\lambda_{\mathcal{V}}.\mathcal{C}_l)$.
5. We define the set of terms $\mathcal{T}'$ by: $\mathcal{T}' ::= \boldsymbol{S} \mid \mathcal{V} \mid \lambda_{\mathcal{V}}.\mathcal{T}' \mid \Pi_{\mathcal{V}:\mathcal{T}'}.\mathcal{T}' \mid \mathcal{T}'\mathcal{T}' \mid l'\mathcal{T}'$.
   We define the set $\mathcal{C}'$ of $\mathcal{T}'$-terms with one hole by:
   $\mathcal{C}' ::= \boxtimes \mid \lambda_{\mathcal{V}}.\mathcal{C}' \mid \Pi_{\mathcal{V}:\mathcal{C}'}.\mathcal{T}' \mid \Pi_{\mathcal{V}:\mathcal{T}'}.\mathcal{C}' \mid \mathcal{C}'\mathcal{T}' \mid \mathcal{T}'\mathcal{C}' \mid l'\mathcal{C}'$.
6. We take A, A_1, A_2, B, a, b, t, M, etc. to range over $\mathcal{T}$, $\mathcal{T}_c$, $\mathcal{T}_l$ and $\mathcal{T}'$.
   We take $\mathbf{C}$, $\mathbf{C}_1$, $\mathbf{C'}$ etc. to range over $\mathcal{C}$, $\mathcal{C}_c$, $\mathcal{C}_l$ and $\mathcal{C}'$.
7. We define translation functions $[\,]^c$, $[\,]^L$, $[\,]^l$ and $[\,]'$ from $\mathcal{T}$ resp. to $\mathcal{T}_c$, $\mathcal{T}_c$, $\mathcal{T}_l$ and $\mathcal{T}'$ as follows:
   - $[s]^c \equiv s, [x]^c \equiv x, [AB]^c \equiv [A]^c\,[B]^c, [\Pi_{x:A}.B]^c \equiv \Pi_{x:[A]^c}.[B]^c$, $[\lambda_{x:A}.B]^c \equiv \lambda_x.[B]^c$.
   - $[s]^L \equiv s, [x]^L \equiv x, [AB]^L \equiv [A]^L\,[B]^L, [\Pi_{x:A}.B]^L \equiv \Pi_{x:[A]^L}.[B]^L$,

$[\lambda_{x:A}.B]^L \equiv \mathsf{Label}[A]^L(\lambda_x.[B]^L)$.

- $[s]^l \equiv s, [x]^l \equiv x, [AB]^l \equiv [A]^l\ [B]^l, [\Pi_{x:A}.B]^l \equiv \Pi_{x:[A]^l}.[B]^l$, $[\lambda_{x:A}.B]^l \equiv l[A]^l(\lambda_x.[B]^l)$.
- $[s]' \equiv s, [x]' \equiv x, [AB]' \equiv [A]'\ [B]', [\Pi_{x:A}.B]' \equiv \Pi_{x:[A]'}.[B]'$, $[\lambda_{x:A}.B]' \equiv l'[A]'(\lambda_x.[B]')$.

8. For $A \in \mathcal{T}_l$, we define $A^\circ \in \mathcal{T}'$ by:
   $s^\circ \equiv s, x^\circ \equiv x, (AB)^\circ \equiv A^\circ\ B^\circ, (\Pi_{x:A}.B)^\circ \equiv \Pi_{x:A^\circ}.B^\circ, (\lambda_x.B)^\circ \equiv \lambda_x.B^\circ$, $(lA(\lambda_x.B))^\circ \equiv l'A^\circ(\lambda_x.B^\circ)$.
9. For $\mathbf{C} \in \mathcal{C}$, we define $[\mathbf{C}]^c$, $[\mathbf{C}]^L$, $[\mathbf{C}]^l$ and $[\mathbf{C}]'$ in the obvious way. Similarly, if $\mathbf{C} \in \mathcal{C}_l$, we define $\mathbf{C}^\circ$ in the obvious way.
10. Let $\mathcal{M} \in \{\mathcal{T}, \mathcal{T}_c, \mathcal{T}_l, \mathcal{T}'\}$. If $\mathbf{C}$ is an $\mathcal{M}$-term with one hole and a is an $\mathcal{M}$-term, we define $\mathbf{C}[a]$ to be the $\mathcal{M}$-term resulting from replacing $\boxtimes$ by a in $\mathbf{C}$.

Note again in the translations given above, the $[\,]^c$ translation $\lambda_x.B'$ of the Church term $\lambda_{x:A}.B$ loses the type A. The $[\,]^L$ translation keeps the type since $[\lambda_{x:A}.B]^L \equiv \mathsf{Label}[A]^L\lambda_x.[B]^L$. Similarly, the translations $[\,]^l$ and $[\,]'$ keep the type of the variable x.

Notation 3. *We assume familiarity with the notion of* compatibility*. We use* $\mathrm{FV}(A)$ *to denote the free variables of A (defined as usual), and $A[x := B]$ to denote the substitution of all the free occurrences of x in A by B (again defined as usual). In particular,* $\mathrm{FV}(lAB) = \mathrm{FV}(A) \cup \mathrm{FV}(B)$, $\mathrm{FV}(\lambda_{x:A}.B) = \mathrm{FV}(A) \cup \mathrm{FV}(B) \setminus \{x\}$, *and* $lAC[x := B] \equiv lA[x := B]C[x := B]$. *Note that as usual, we take terms to be equivalent up to variable renaming and let $\equiv$ denote syntactic equality [1]. We assume the Barendregt convention (BC) where names of bound variables are chosen to differ from free ones in a term and where different abstraction operators bind different variables. Hence, for example, we write* $(\Pi_{y:A}.y)x$ *instead of* $(\Pi_{x:A}.x)x$ *and* $\Pi_{x:A}.\Pi_{y:B}.C$ *instead of* $\Pi_{x:A}.\Pi_{x:B}.C$. *We also assume (BC) for contexts and typings so that for example, if* $\Gamma \vdash \Pi_{x:A}.B : C$ *then x will not occur in Γ. For $\pi \in \{\lambda, \Pi\}$, we write* $\pi_{x_m:A_m} \ldots \pi_{x_n:A_n}.A$ *as* $\pi^{i:m..n}_{x_i:A_i}.A$. *We also write* $\lambda_{x_m} \ldots \lambda_{x_n}.A$ *as* $\lambda^{i:m..n}_{x_i}.A$.
As usual, if $x \notin \mathrm{FV}(B)$ *then we write* $\Pi_{x:A}.B$ *as* $A \longrightarrow B$.

The next lemma connects terms and their translations.

Lemma 4.

1. *If A is free of any λ then $[A]^L \equiv [A]^l \equiv [A]' \equiv [A]^c \equiv A$.*
2. *Let $A \in \mathcal{T}$. We have: $\text{FV}([A]^c) \subseteq \text{FV}(A) = \text{FV}([A]^L) = \text{FV}([A]^l) = \text{FV}([A]')$.*
3. *Let $A \in \mathcal{T}_l$. We have: $\text{FV}(A) = \text{FV}(A^\circ)$.*
4. *Let $A, B \in \mathcal{T}$. We have: $[A]^c[x := [B]^c] \equiv [A[x := B]]^c$, $[A]^L[x := [B]^L] \equiv [A[x := B]]^L$, $[A]^l[x := [B]^l] \equiv [A[x := B]]^l$ and $[A]'[x := [B]'] \equiv [A[x := B]]'$.*
5. *Let $A, B \in \mathcal{T}_l$. We have: $A[x := B]^\circ \equiv A^\circ[x := B^\circ]$.*
6. *If $[A]^L \equiv [B]^L$ or $[A]^l \equiv [B]^l$ or $[A]' \equiv [B]'$ then $A \equiv B$.*
7. *It is not the case that $[A]^c \equiv [B]^c$ implies $A \equiv B$.*
8. *Let $\mathbf{C} \in \mathcal{C}$ and $A \in \mathcal{T}$. We have: $[\mathbf{C}[A]]^c \equiv [\mathbf{C}]^c[[A]^c]$, $[\mathbf{C}[A]]^L \equiv [\mathbf{C}]^L[[A]^L]$, $[\mathbf{C}[A]]^l \equiv [\mathbf{C}]^l[[A]^l]$, and $[\mathbf{C}[A]]' \equiv [\mathbf{C}]'[[A]']$.*
9. *Let $\mathbf{C} \in \mathcal{C}_l$ and $A \in \mathcal{T}_l$. We have: $(\mathbf{C}[A])^\circ \equiv \mathbf{C}^\circ[A^\circ]$.*
10. *If $[\mathbf{C}_1]^L \equiv [\mathbf{C}_2]^L$ or $[\mathbf{C}_1]^l \equiv [\mathbf{C}_2]^l$ or $[\mathbf{C}_1]' \equiv [\mathbf{C}_2]'$ then $\mathbf{C}_1 \equiv \mathbf{C}_2$.*
11. *It is not the case that $[\mathbf{C}_1]^c \equiv [\mathbf{C}_2]^c$ implies $\mathbf{C}_1 \equiv \mathbf{C}_2$.*

Proof. All of 1..6 are by induction on the structure of A. We only do the case $A \equiv \lambda_{x:C}.D$ of $[A]^L \equiv [B]^L$ of 6. Since $A \equiv \lambda_{x:C}.D$ then $[A]^L \equiv \mathsf{Label}[C]^L(\lambda_x.[D]^L) \equiv [B]^L$, hence $B \equiv \lambda_{x:E}.F$ where $[E]^L \equiv [C]^L$ and $[F]^L \equiv [D]^L$. By IH, $B \equiv \lambda_{x:C}.D \equiv A$. As for 7., let $A \equiv \lambda_{x:y}.x$ and $B \equiv \lambda_{x:z}.x$ where $y \neq z$. It is obvious that $[A]^c \equiv \lambda_x.x \equiv [B]^c$ but $A \not\equiv B$.

8 and 9 are by induction on $\mathbf{C}$.

9 is by induction on $\mathbf{C}_1$.

For 11, use a similar counterexample to that of 7. ⊠

Next we introduce the various notions of reductions used in this paper.

Definition 5. [Reductions]

- β-reduction $\to_\beta$ is the compatible closure of $(\lambda_{x:A}.B)C \to_\beta B[x := C]$.
- $\overline{\beta}$-reduction $\to_{\overline{\beta}}$ is the compatible closure of $(\lambda_x.B)C \to_{\overline{\beta}} B[x := C]$.
- l-reduction $\to_l$ is the compatible closure of $lA(\lambda_x.b) \to_l \lambda_x.b$.
- l'-reduction $\to_{l'}$ is the compatible closure of $l'A(\lambda_x.b) \to_{l'} \lambda_x.b$.

- We define the union of reduction relations as usual. For example, $\beta_{\overline{\beta}l}$-reduction $\rightarrow_{\overline{\beta}l}$, is the union of $\rightarrow_{\overline{\beta}}$ and $\rightarrow_l$. We speak of $\underline{\beta}$-reduction and use $\rightarrow_{\underline{\beta}}$ to denote $\rightarrow_{\overline{\beta}l}$. We also speak of β'-reduction and use $\rightarrow_{\beta'}$ to denote $\rightarrow_{\overline{\beta}l'}$. Note that β is defined on $\mathcal{T}$; $\overline{\beta}$ is defined on $\mathcal{T}_c$, $\mathcal{T}_l$ and $\mathcal{T}'$; l and $\underline{\beta}$ are defined on $\mathcal{T}_l$, and l' and β' are defined on $\mathcal{T}'$.
- We write $A \Rightarrow_{l\overline{\beta}} B$ when $A \equiv \mathbf{C}[lE(\lambda_x.b)a] \rightarrow_l \mathbf{C}[(\lambda_x.b)a] \rightarrow_{\overline{\beta}} \mathbf{C}[b[x := a]] \equiv B$.
- We write $A \Rightarrow_{l'\overline{\beta}} B$ when $A \equiv \mathbf{C}[l'E(\lambda_x.b)a] \rightarrow_{l'} \mathbf{C}[(\lambda_x.b)a] \rightarrow_{\overline{\beta}} \mathbf{C}[b[x := a]] \equiv B$.
- Let $r \in \{\beta, \overline{\beta}, l, l', \underline{\beta}, \beta'\}$. We define r-redexes in the usual way. Moreover:
  - $\twoheadrightarrow_r$ is the reflexive transitive closure of $\rightarrow_r$ and $=_r$ is the equivalence closure of $\rightarrow_r$. We write $\overset{+}{\twoheadrightarrow}_r$ to denote one or more steps of r-reduction. We write $\overset{\leq n}{\twoheadrightarrow}_r$ (resp. $\overset{n}{\twoheadrightarrow}_r$) to denote at most (resp. exactly) n steps of r-reduction.
  - If $A \rightarrow_r B$ (resp. $A \twoheadrightarrow_r B$), we also write $B \ {}_r\!\leftarrow A$ (resp. $B \ {}_r\!\twoheadleftarrow A$).
  - We say that A is strongly normalising with respect to $\rightarrow_r$ (we use the notation $\mathrm{SN}_{\rightarrow_r}(A)$) if there are no infinite $\rightarrow_r$-reductions starting at A.
  - We say that A is in r-normal form, notation $\mathrm{NF}_{\rightarrow_r}(A)$, if there is no B such that $A \rightarrow_r B$.
  - We use $\mathrm{nf}_r(A)$ to refer to the r-normal form of A if it exists.
  - We say that A is r-weakly normalising, notation $\mathrm{WN}_{\rightarrow_r}(A)$, if $A \twoheadrightarrow_r B$ where $\mathrm{NF}_{\rightarrow_r}(B)$.

Remark 6. [Label, l, l' save the type of a variable in a Church expression] We introduced Label to be used as a type saver when we translate a Church expression $\lambda_{y:C}.d$ whose type is $\Pi_{y:C}.D$ into a Curry expression. Recall that $[\lambda_{y:C}.d]^L \equiv \mathsf{Label}A(\lambda_y.b)$ where $[C]^L \equiv A$, $[d]^L \equiv b$. Hence the type of y of the Church expression $\lambda_{y:C}.d$ is protected in its translation into a Curry expression $\mathsf{Label}A(\lambda_y.b)$. Similarly, l and l' save the type of a variable in a Church expression: $[\lambda_{y:C}.d]^l \equiv l[C]^l([\lambda_y.d]^l)$ and $[\lambda_{y:C}.d]' \equiv l'[C]'([\lambda_y.d])$.

Note that we only use the reduction of $\mathsf{Label}AB$ to B when B is an abstract $\lambda_y.b$, and in this case this reduction holds in $\lambda\beta$-reduction.

The next lemma establishes the metasubstitution property for all our systems and shows that $=_r$ is closed under substitution.

Lemma 7. *Let* $(\mathcal{E}, r) \in \{(\mathcal{T}, \beta), (\mathcal{T}_c, \overline{\beta}), (\mathcal{T}_l, \underline{\beta}), (\mathcal{T}', \beta')\}$. *In* $(\mathcal{E}, r)$ *we have:*

1. $A[x := B][y := C] \equiv A[y := C][x := B[y := C]]$.
2. *If* $B =_r C$ *then* $A[x := B] =_r A[x := C]$.
3. *If* $A =_r B$ *and* $C =_r D$ *then* $A[x := C] =_r B[x := D]$.

Proof. 1 and 2 are by induction on the structure of A. 3 is by induction on $A =_r B$ using 1 and 2. ⊠

Theorem 8 (Church-Rosser (CR) for $\rightarrow_{\beta/\overline{\beta}/l/l'/\underline{\beta}/\beta'}$)**.** *Let* $r \in \{\beta, \overline{\beta}, l, l', \underline{\beta}, \beta'\}$. *If* $B_1 \ {}_r\!\!\twoheadleftarrow A \twoheadrightarrow_r B_2$ *then there exists* C *such that* $B_1 \twoheadrightarrow_r C \ {}_r\!\!\twoheadleftarrow B_2$.

Proof. For β and $\overline{\beta}$, see [2]. Let $(\mathcal{E}, r) \in \{(\mathcal{T}_c, \overline{\beta}), (\mathcal{T}_l, l), (\mathcal{T}', l'), (\mathcal{T}_l, \underline{\beta}), (\mathcal{T}', \beta')\}$. $(\mathcal{E}, r)$ is an ortogonal higher order term rewriting system (i.e. left linear since no variable occurs twice on the lefthand side in the l and l' rules, and all critical pairs are trivial) and hence r is Church-Rosser (see page 644 of [13]). ⊠

Corollary 9. *For* $r \in \{\beta, \overline{\beta}, l, l', \underline{\beta}, \beta'\}$, *r-normal forms are unique.*

The next lemma is needed to show the preservation and closure of $=_r$ for $r \in \{\beta, \overline{\beta}, \underline{\beta}, \beta'\}$ under the relevant translations.

Lemma 10. *Let* $A, B \in \mathcal{T}$ *and* $A_1, B_1 \in \mathcal{T}_l$.

1. $([A]^l)^\circ \equiv [A]'$.
2. *(a) If* $A_1^\circ \rightarrow_{\beta'} D$ *then there is an* E *such that* $D \equiv E^\circ$ *and* $A_1 \rightarrow_{\underline{\beta}} E$.
   (b) Hence, if $A_1^\circ \twoheadrightarrow_{\beta'} D$ *then there is an* E *such that* $D \equiv E^\circ$ *and* $A_1 \twoheadrightarrow_{\underline{\beta}} E$.
3. *If* $A \rightarrow_\beta B$ *then:*
   (a) $[A]^c \overset{\leq 1}{\twoheadrightarrow}_{\overline{\beta}} [B]^c$ *and* $[A]^L \overset{+}{\twoheadrightarrow}_{\overline{\beta}} [B]^L$.
   (b) $[A]^l \Rightarrow_{l\overline{\beta}} [B]^l$ *and hence* $[A]^l \overset{+}{\twoheadrightarrow}_{\underline{\beta}} [B]^l$.
   (c) $[A]' \Rightarrow_{l'\overline{\beta}} [B]'$ *and hence* $[A]' \overset{+}{\twoheadrightarrow}_{\beta'} [B]'$.
4. *If* $A =_\beta B$ *then* $[A]^c =_{\overline{\beta}} [B]^c$, $[A]^L =_{\overline{\beta}} [B]^L$, $[A]^l =_{\underline{\beta}} [B]^l$ *and* $[A]' =_{\beta'} [B]'$.
5. *If* $A_1 \twoheadrightarrow_{\underline{\beta}} B_1$ *then* $A_1^\circ \twoheadrightarrow_{\beta'} B_1^\circ$.
6. *If* $A_1 =_{\underline{\beta}} B_1$ *then* $A_1^\circ =_{\beta'} B_1^\circ$.

7. $[A]' =_{\beta'} [B]'$ *iff* $[A]^l =_{\underline{\beta}} [B]^l$.
8. *If* $[A]^l \Rightarrow_{l\overline{\beta}} B_1$ *then* $A \equiv \mathbf{C}[(\lambda_{x:E}.F)a] \to_\beta \mathbf{C}[F[x := a]]$ *and* $B_1 \equiv [\mathbf{C}[F[x := a]]]^l$.

Proof. 1. By induction on the structure of $A \in \mathcal{T}$.

2 (a) By induction on the structure of $A_1 \in \mathcal{T}_l$. (b) by induction on the length of the derivation $A_1^\circ \twoheadrightarrow_{\beta'} D$.

3 (a), (b) and (c) are by induction on the derivation $A \to_\beta B$ using lemma 4.

4. is by Church-Rosser of β and 3 (a), (b) and (c) above.

5. First show by induction on $A_1 \to_{\underline{\beta}} B_1$ that $A_1 \to_{\underline{\beta}} B_1$ gives $A_1^\circ \to_{\beta'} B_1^\circ$, then use induction on the length of the derivation $A_1 \twoheadrightarrow_{\underline{\beta}} B_1$.

6. is by Church-Rosser of $\underline{\beta}$ and 4 above.

7. If $[A]^l =_{\underline{\beta}} [B]^l$ then by 6 above, $([A]^l)^\circ =_{\beta'} ([B]^l)^\circ$ and by 1 above, $[A]' =_{\beta'} [B]'$. As for the other direction, if $[A]' =_{\beta'} [B]'$ then by 1 above, $([A]^l)^\circ =_{\beta'} ([B]^l)^\circ$ and hence by CR of β', there is a C such that $([A]^l)^\circ \twoheadrightarrow_{\beta'} C \ {}_{\beta'}\!\!\twoheadleftarrow ([B]^l)^\circ$. By 2 above, there are C_1, C_2 such that $C_1^\circ \equiv C \equiv C_2^\circ$ (hence $C_1 \equiv C_2$) and $[A]^l \twoheadrightarrow_{\underline{\beta}} C_1 \ {}_{\underline{\beta}}\!\!\twoheadleftarrow [B]^l$. Hence $[A]^l =_{\underline{\beta}} [B]^l$.

8. By induction on A. ⊠

Definition 11.

- Let $\beta*$ be the least compatible relation on $\mathcal{T}_l$ closed under $(lA(\lambda_x.B))C \to_{\beta*} B[x := C]$, and define $\twoheadrightarrow_{\beta*}$ and $=_{\beta*}$ as usual.
- Let $[\mathcal{T}]^l = \{[A]^l | A \in \mathcal{T}\} \subset \mathcal{T}_l$.

Lemma 12.

1. *If* $A \to_{\beta*} B$ *then* $A \twoheadrightarrow_{\underline{\beta}} B$.
2. *If* $[A]^l \twoheadrightarrow_{\beta*} B$ *then* $B \equiv [C]^l$ *where* $A \twoheadrightarrow_\beta C$.
3. *If* $A \twoheadrightarrow_\beta B$ *then* $[A]^l \twoheadrightarrow_{\beta*} [B]^l$.
4. $\beta*$ *is CR on* $[\mathcal{T}]^l$.
5. $A =_\beta B$ *iff* $[A]^l =_{\beta*} [B]^l$.

Proof. 1. Easy noting that a $\beta*$-step is a simultaneous application of an l-step with a $\overline{\beta}$-step.

2. First show by induction on the derivation of $[A]^l \to_{\beta *} B$ that if $[A]^l \to_{\beta *} B$ then $B \equiv [C]^l$ where $A \to_\beta C$. Then, show the lemma by induction on the length of the derivation $[A]^l \twoheadrightarrow_{\beta *} B$.

3. First show by induction on the derivation of $A \to_\beta B$ that if $A \to_\beta B$ then $[A]^l \to_{\beta *} [B]^l$. Then, show the lemma by induction on the length of the derivation $A \twoheadrightarrow_\beta B$.

4. If $B_1 \ {}_{\beta *}\!\!\twoheadleftarrow [A]^l \twoheadrightarrow_{\beta *} B_2$ then by 2, there are C_1, C_2 such that $B_1 \equiv [C_1]^l$, $B_2 \equiv [C_2]^l$, and $C_1 \ {}_\beta\!\!\twoheadleftarrow A \twoheadrightarrow_\beta C_2$. Hence by CR ($\beta$), there is a C such that $C_1 \twoheadrightarrow_\beta C \ {}_\beta\!\!\twoheadleftarrow C_2$ and by 3, $B_1 \twoheadrightarrow_{\beta *} [C]^l \ {}_{\beta *}\!\!\twoheadleftarrow B_2$.

5. If $A =_\beta B$ then by CR (β), there is a C such that $A \twoheadrightarrow_\beta C \ {}_\beta\!\!\twoheadleftarrow B$ and by 3, $[A]^l \twoheadrightarrow_{\beta *} [C]^l \ {}_{\beta *}\!\!\twoheadleftarrow [B]^l$ and hence $[A]^l =_{\beta *} [B]^l$.

If $[A]^l =_{\beta *} [B]^l$ then similar to above proof using 2 instead of 3. ⊠

3. Typing

In this section we introduce the type systems that will be studied in this paper and that will be used to interpret Church's style into Curry's style of typing. These systems include the PTSs of [2] which are in Church's style, the DFPTSs of [3] which are in Curry's style, the L-complete DFPTSs, the lPTSs and the l'PTSs. For all these systems, we establish the necessary properties.

Definition 13. [Declarations, contexts, $\subseteq$]

1. A *declaration* d is of the form $x : A$. We define $\mathtt{var}(d) \equiv x$, $\mathtt{type}(d) \equiv A$, and $\text{FV}(d) = \text{FV}(A)$. We let $d, d', d_1, \ldots$ range over declarations.
2. A *context* Γ is a concatenation of declarations $d_1, d_2, \cdots, d_n$ such that if $i \neq j$ then $\mathtt{var}(d_i) \not\equiv \mathtt{var}(d_j)$. We define $\text{DOM}(\Gamma) = \{\mathtt{var}(d) \mid d \in \Gamma\}$ and use $\langle\rangle$ to denote the *empty context*. We let $\Gamma, \Delta, \Gamma', \Gamma_1, \ldots$ range over contexts.
3. Assume Γ is a context such that $x \notin \text{DOM}(\Gamma)$. We define the substitution of A for x on Γ, denoted $\Gamma[x := A]$, inductively as follows:
   $\langle\rangle[x := A] \equiv \langle\rangle$, and $(\Gamma', y : B)[x := A] \equiv \Gamma'[x := A], y : B[x := A]$.
4. We define $\subseteq$ between contexts as the least reflexive transitive relation closed under: $\Gamma, \Delta \subseteq \Gamma, d, \Delta$.

For $r \in \{\beta, \overline{\beta}, l, \underline{\beta}, l', \beta'\}$, we extend r-reduction to contexts in the usual way. Similarly, we extend the translations in Definition 2 to contexts as follows:

$[\langle\rangle]^c \equiv \langle\rangle \quad [\Gamma, x : A]^c \equiv [\Gamma]^c, x : [A]^c.$

$[\langle\rangle]^l \equiv \langle\rangle \quad [\Gamma, x : A]^l \equiv [\Gamma, x : [A]]^l.$

$[\langle\rangle]^{l'} \equiv \langle\rangle \quad [\Gamma, x : A]^{l'} \equiv [\Gamma, x : [A]]^{l'}.$

$\langle\rangle^\circ \equiv \langle\rangle \quad (\Gamma, x : A)^\circ \equiv \Gamma^\circ, x : A^\circ.$

Definition 14. [Type Systems]

- A specification $\mathcal{S}$ is a triple $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ such that $\boldsymbol{S}$ is a set of sorts, $A \subseteq \boldsymbol{S} \times \boldsymbol{S}$ is a set of axioms and $\boldsymbol{R} \subseteq \boldsymbol{S} \times \boldsymbol{S} \times \boldsymbol{S}$ is a set of rules. When no confusion occurs with an axiom, a rule of the form (s_1, s_2, s_2) is written as (s_1, s_2).
  - A sort s is said to be a top sort if there is no $(s, s') \in \boldsymbol{A}$. The set of top sorts is denoted by $\boldsymbol{S}_T$.
  - (s_1, s_2) is l'-complete if there are s_3, s_4 such that $(s_1, s_4, s_2), (s_2, s_2, s_3) \in \boldsymbol{R}$.
  - A specification $\mathcal{S}$ is said to be functional (also singly-sorted) if:
    1. If $(s, s') \in \boldsymbol{A}$ and $(s, s'') \in \boldsymbol{A}$ then $s' = s''$.
    2. If $(s_1, s_2, s_3) \in \boldsymbol{R}$ and $(s_1, s_2, s_3') \in \boldsymbol{R}$ then $s_3 = s_3'$.
  - A specification $\mathcal{S}$ is said to be injective if:
    1. If $(s', s) \in \boldsymbol{A}$ and $(s'', s) \in \boldsymbol{A}$ then $s' = s''$.
    2. If $(s_1, s_2, s_3) \in \boldsymbol{R}$ and $(s_1, s_2', s_3) \in \boldsymbol{R}$ then $s_2 = s_2'$.
- Let $\mathcal{S} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ be a specification. We define:
  - $\vdash_\beta$ to be the typing relation given by the rules of Figures 2 and 3.
  - $\vdash_{\overline{\beta}}$ to be the typing relation given by the rules of Figures 2 and 4.
  - $\vdash_{\underline{\beta}}$ to be the typing relation given by the rules of Figures 2, 4 and 5.
  - $\vdash_{\beta'}$ to be the typing relation given by the rules of Figures 2, 4 and 6.

$$
\begin{array}{ll}
\text{(axiom)} & \langle\rangle \vdash_r s_1 : s_2 \qquad (s_1, s_2) \in \boldsymbol{A} \\[2ex]
\text{(start)} & \dfrac{\Gamma \vdash_r A : s \qquad x \notin \mathrm{DOM}(\Gamma)}{\Gamma, x{:}A \vdash_r x : A} \\[2ex]
\text{(weak)} & \dfrac{\Gamma \vdash_r A : B \qquad \Gamma \vdash_r C : s \qquad x \notin \mathrm{DOM}(\Gamma)}{\Gamma, x{:}C \vdash_r A : B} \\[2ex]
(\Pi) & \dfrac{\Gamma \vdash_r A : s_1 \qquad \Gamma, x{:}A \vdash_r B : s_2 \qquad (s_1, s_2, s_3) \in \boldsymbol{R}}{\Gamma \vdash_r \Pi_{x:A}.B : s_3} \\[2ex]
(\mathrm{conv}_r) & \dfrac{\Gamma \vdash_r A : B \qquad \Gamma \vdash_r B' : s \qquad B =_r B'}{\Gamma \vdash_r A : B'} \\[2ex]
\text{(appl)} & \dfrac{\Gamma \vdash_r F : \Pi_{x:A}.B \qquad \Gamma \vdash_r a : A}{\Gamma \vdash_r Fa : B[x{:=}a]}
\end{array}
$$

Figure 2: The common $\vdash_r$ typing rules

$$
(\lambda) \quad \frac{\Gamma, x{:}A \vdash_r b : B \qquad \Gamma \vdash_r \Pi_{x:A}.B : s}{\Gamma \vdash_r \lambda_{x:A}.b : \Pi_{x:A}.B}
$$

Figure 3: The λ-rule for Church's typing

- When $\vdash_r$ is a typing relation on a specification $\mathcal{S}$, we write $\vdash_r^{\mathcal{S}}$ to emphasize the dependability of type derivation on $\mathcal{S}$.
- The Pure Type System (PTS) induced by $\mathcal{S}$ is the tuple $\lambda\mathcal{S} = (\mathcal{T}, \beta, \vdash_\beta^{\mathcal{S}})$.
- The Domain Free Pure Type System (DFPTS) induced by $\mathcal{S}$ is the tuple $\overline{\lambda}\mathcal{S} = (\mathcal{T}_c, \overline{\beta}, \vdash_{\overline{\beta}}^{\mathcal{S}})$.
- The l-Labeled Pure Type System (lPTS) induced by $\mathcal{S}$ is the tuple $\underline{\lambda}\mathcal{S} = (\mathcal{T}_l, \underline{\beta}, \vdash_{\underline{\beta}}^{\mathcal{S}})$.
- The l'-Labeled Pure Type System (l'PTS) induced by $\mathcal{S}$ is the tuple $\tilde{\lambda}\mathcal{S} = (\mathcal{T}', \beta', \vdash_{\beta'}^{\mathcal{S}})$.

As special cases of PTSs, we use the eight powerful systems of Barendregt's β-cube. In the β-cube of [2], eight well-known type systems are given in a uniform way. The weakest system is Church's simply typed λ-calculus $\lambda\!\!\rightarrow$ [5], and

$$(\lambda_c) \quad \frac{\Gamma, x{:}A \vdash_r b : B \qquad \Gamma \vdash_r \Pi_{x:A}.B : s}{\Gamma \vdash_r \lambda_x.b : \Pi_{x:A}.B}$$

Figure 4: The λ_c-rule for Curry's typing

$$(\mathrm{l}) \quad \frac{\Gamma, x{:}A \vdash_r b : B \qquad \Gamma \vdash_r \Pi_{x:A}.B : s}{\Gamma \vdash_r lA(\lambda_x.b) : \Pi_{x:A}.B}$$

Figure 5: The l-label-rule

the strongest system is the Calculus of Constructions $\lambda P\omega$ [6]. The second order λ-calculus [8, 19] figures on the β-cube between $\lambda\!\!\rightarrow$ and $\lambda P\omega$ (cf. Figure 8). Moreover, via the Propositions-as-Types principle (see [10]), many logical systems can be described in the β-cube. In the β-cube, $*$ is the set of types and $\Box$ is the set of kinds and we have $* : \Box$ as a special axiom (i.e., $(* : \Box) \in \boldsymbol{A}$. If $A : *$ (resp. $A : \Box$) we say A is a type (resp. a kind). All pure type systems have the same typing rules (cf. Figure 2) but differ by the set $\boldsymbol{R}$ of triple of sorts (s_1, s_2, s_3) allowed in the *type-formation* or Π-*formation* rule, (Π).

The β-cube is a collection of 8 special systems where each system has its own set $\boldsymbol{R}$ such that $(*,*) \in \boldsymbol{R} \subseteq \{(*,*),(*,\Box),(\Box,*),(\Box,\Box)\}$. With rule (Π), the β-cube factorises the expressive power into three features: *polymorphism, type constructors*, and *dependent types*:

- $(*,*)$ is basic. All the β-cube systems have this rule.
- $(\Box,*)$ takes care of polymorphism. $\lambda 2$ is the weakest system on the β-cube that features this rule.
- $(\Box,\Box)$ takes care of type constructors. $\lambda\underline{\omega}$ is the weakest system on the

$$(l') \quad \frac{\Gamma \vdash_r A : s_1 \qquad \Gamma \vdash_r \Pi_{x:A}.B : s_2 \qquad z \neq x, z \notin \mathrm{DOM}(\Gamma)}{\Gamma \vdash_r l'A : \Pi_{x:A}.B \longrightarrow \Pi_{x:A}.B}\,(s_1, s_2)\ l'\text{-complete}$$

Figure 6: The l'-label-rule

$\lambda\!\!\rightarrow$	$(*,*)$			
$\lambda 2$	$(*,*)$	$(\Box,*)$		
λP	$(*,*)$		$(*,\Box)$	
λP2	$(*,*)$	$(\Box,*)$	$(*,\Box)$	
$\lambda\underline{\omega}$	$(*,*)$			$(\Box,\Box)$
$\lambda\omega$	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$
λP$\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$
$\lambda P\omega$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$

Figure 7: Systems of Barendregt's β-cube

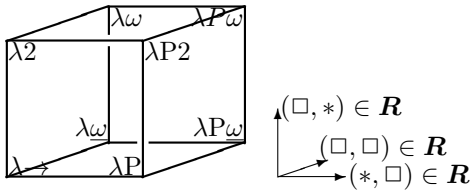

Figure 8: Barendregt's β-cube

β-cube that features this rule.

- $(*,\Box)$ takes care of term dependent types. λ_P is the weakest system on the β-cube that features this rule.

We refer to each system of Figure 8 according to the kind of PTSs we are in. So, $\lambda P\omega$, (resp. $\overline{\lambda}P\omega$, resp. $\underline{\lambda}P\omega$, resp. $\tilde{\lambda}P\omega$) is the PTS (resp. DFPTS, resp. lPTS, resp. l'PTS) calculus of constructions. Now we give basic notions of type systems:

Definition 15. [Legal-Contexts, Judgements, Types, Terms]
Let $(\mathcal{E},r) \in \{(\mathcal{T},\beta),(\mathcal{T}_c,\overline{\beta}),(\mathcal{T}_l,\underline{\beta}),(\mathcal{T}',\beta')\}$ and let $\mathcal{S}$ be a specification on $(\mathcal{E},r)$.

1. $\Gamma \vdash_r A : B$ is a *judgement* which states that A has type B in context Γ. When Γ is empty, we simply write $\vdash_r A : B$.
2. Γ is $\vdash_r$-*legal* (or simply *legal*) if there exist A, B where $\Gamma \vdash_r A : B$.
3. A is $\vdash_r$-*legal* (or simply *legal*) if there exist Γ, B where $\Gamma \vdash_r A : B$ or $\Gamma \vdash_r B : A$.
4. A is $\Gamma \vdash_r$-*legal* (or simply Γ-*legal*) if there exists B where $\Gamma \vdash_r A : B$ or $\Gamma \vdash_r$

$B : A$.

5. If $\Gamma \vdash_r x : A$ for every $x : A \in \Delta$, we write $\Gamma \vdash_r \Delta$.
6. We write $\Gamma \vdash_r A : B : C$ for $\Gamma \vdash_r A : B$ and $\Gamma \vdash_r B : C$.
7. We define $Type_r = \cup_{s \in \boldsymbol{S}} Type_r^s$ and $Term_r = \cup_{s \in \boldsymbol{S}} Term_r^s$ where:
   - $Type_r^s = \{M \in \mathcal{E} \mid \Gamma \vdash_r M : s \text{ for some } \Gamma\}$.
   - $Term_r^s = \{M \in \mathcal{E} \mid \Gamma \vdash_r M : A : s \text{ for some } \Gamma \text{ and } A\}$.
8. We say that $\mathcal{S}$ satisfies unicity of types w.r.t $\vdash_r$ if whenever $\Gamma \vdash_r A : B_1$ and $\Gamma \vdash_r A : B_2$, then $B_1 =_r B_2$.
9. We say that $\mathcal{S}$ preserves sorts w.r.t $\vdash_r$ if whenever $\Gamma \vdash_r B_1 : s_1$, $\Gamma \vdash_r B_2 : s_2$, and $B_1 =_r B_2$ then $s_1 \equiv s_2$.

3.1. Properties of our Type Systems

In this section we establish the necessary properties of the various type systems introduced. These properties all hold for all our typing relations. In the following sections we will see how and when these various type systems can be used for interpreting Church's style into Curry's style of typing.

Lemma 16 (Free Variable Lemma for $\vdash_r$ and $\to_r$)**.**

1. *If $x : A$ and $y : B$ are different elements in a legal context Γ, then $x \not\equiv y$.*
2. *If $\Gamma_1, x : A, \Gamma_2 \vdash_r B : C$ then*
   a) $\text{FV}(A) \subseteq \text{DOM}(\Gamma_1)$ and $\text{FV}(B), \text{FV}(C) \subseteq \text{DOM}(\Gamma_1, x : A, \Gamma_2)$.
   b) If $\Gamma_1 \vdash_r D : s$ and $D =_r A$ then $\Gamma_1, x : D, \Gamma_2 \vdash_r B : C$.

Proof.

1. Since Γ is legal, assume $\Gamma \vdash_r C : D$. The proof is by induction on the derivation $\Gamma \vdash_r C : D$.
2. Both a) and b) are by induction on the derivation $\Gamma_1, x : A, \Gamma_2 \vdash_r B : C$. We only do the (start) case of b).

   If $\Gamma_1, x : A, \Gamma_2, y : E \vdash_r y : E$ comes from $\Gamma_1, x : A, \Gamma_2 \vdash_r E : s'$ and $y \notin \text{DOM}(\Gamma_1, x : A, \Gamma_2)$, then by IH, $\Gamma_1, x : D, \Gamma_2 \vdash_r E : s'$ and by (start) $\Gamma_1, x : D, \Gamma_2, y : E \vdash_r y : E$.

   If $\Gamma_1, x : A \vdash_r x : A$ comes from $\Gamma_1 \vdash_r A : s'$ and $x \notin \text{DOM}(\Gamma_1)$, and if

$\Gamma_1 \vdash_r D : s$ and $D =_r A$ then by (start) $\Gamma_1, x : D \vdash_r x : D$ and by (weak) $\Gamma_1, x : D \vdash_r A : s'$. Hence by (conv) $\Gamma_1, x : D \vdash_r x : A$.

$\boxtimes$

Lemma 17 (Start/Context Lemma for $\vdash_r$ and $\to_r$)**.** *If Γ is $\vdash_r$-legal then $\Gamma \vdash_r s_1 : s_2$ for every $(s_1, s_2) \in \boldsymbol{A}$ and for all $x : A \in \Gamma$, $\Gamma \vdash_r x : A$ and $\Gamma \vdash_r A : s$.*

Proof. Since Γ is legal, assume $\Gamma \vdash_r C : D$. The proof is by induction on the derivation $\Gamma \vdash_r C : D$. $\boxtimes$

Lemma 18 (Transitivity Lemma)**.** *If Δ is $\vdash_r$-legal, $\Delta \vdash_r \Gamma$ and $\Gamma \vdash_r A : B$ then $\Delta \vdash_r A : B$.*

Proof. By induction on the derivation $\Gamma \vdash_r A : B$. $\boxtimes$

Lemma 19 (Thinning Lemma for $\vdash_r$ and $\to_r$)**.**
If Δ is $\vdash_r$-legal, $\Gamma \subseteq \Delta$, and $\Gamma \vdash_r A : B$ then $\Delta \vdash_r A : B$.

Proof. Since $\Gamma \subseteq \Delta$, and since Δ is $\vdash_r$-legal, by Start Lemma 17, $\Delta \vdash_r \Gamma$. Hence, by Transitivity Lemma 18, $\Delta \vdash_r A : B$. $\boxtimes$

Lemma 20 (Substitution Lemma for $\vdash_r$ and $\to_r$)**.** *If $\Gamma, x : A, \Delta \vdash_r B : C$ and $\Gamma \vdash_r a : A$ then $\Gamma, \Delta[x := a] \vdash_r B[x := a] : C[x := a]$.*

Proof. By induction on the derivation $\Gamma, x : A, \Delta \vdash_r B : C$. $\boxtimes$

Lemma 21 (Generation Lemma for $\vdash_r$ and $\to_r$)**.**

1. *If $\Gamma \vdash_r s : C$ then there is a s' such that $(s, s') \in \boldsymbol{A}$, $C =_r s'$ and if $C \not\equiv s'$ then $\Gamma \vdash_r C : s''$ for some sort s''.*
2. *If $\Gamma \vdash_r x : C$ then for some s, A, $x : A \in \Gamma$, $C =_r A$, and $\Gamma \vdash_r C : s$.*
3. *If $\Gamma \vdash_\beta \lambda_{x:A}.B : C$ then for some D, s, $\Gamma \vdash_\beta \Pi_{x:A}.D : s$; $\Gamma, x{:}A \vdash_\beta B : D$; $\Pi_{x:A}.D =_\beta C$ and if $\Pi_{x:A}.D \not\equiv C$ then $\Gamma \vdash_\beta C : s'$ for some sort s'.*
4. *If $\Gamma \vdash_{\underline{\beta}} lA(\lambda_x.B) : C$ then for some D, s, $\Gamma \vdash_{\underline{\beta}} \Pi_{x:A}.D : s$; $\Gamma, x{:}A \vdash_{\underline{\beta}} B : D$; $\Pi_{x:A}.D =_{\underline{\beta}} C$ and if $\Pi_{x:A}.D \not\equiv C$ then $\Gamma \vdash_{\underline{\beta}} C : s'$ for some sort s'.*

5. *If* $\Gamma \vdash_{\beta'} l'A : C$ *then there is* (s_1, s_2) l'*-complete, there is* B, z, x *where* $z \neq x, z \notin \mathrm{DOM}(\Gamma)$, $\Gamma \vdash_{\beta'} A : s_1$, $\Gamma \vdash_{\beta'} \Pi_{x:A}.B : s_2$, $C =_{\beta'} \Pi_{z:\Pi_{x:A}.B}.\Pi_{x:A}.B$ *and if* $C \not\equiv \Pi_{z:\Pi_{x:A}.B}.\Pi_{x:A}.B$ *then* $\Gamma \vdash_{\beta'} C : s$ *for some* s.
6. *If* $\Gamma \vdash_r \lambda_x.B : C$ *then for some* A, D, s, $\Gamma \vdash_r \Pi_{x:A}.D : s$; $\Gamma, x{:}A \vdash B : D$; $\Pi_{x:A}.D =_r C$ *and if* $\Pi_{x:A}.D \not\equiv C$ *then* $\Gamma \vdash_r C : s'$ *for some sort* s'.
7. *If* $\Gamma \vdash_r \Pi_{x:A}.B : C$ *then there is* $(s_1, s_2, s_3) \in \boldsymbol{R}$ *such that* $\Gamma \vdash_r A : s_1$, $\Gamma, x{:}A \vdash_r B : s_2$, $C =_r s_3$ *and if* $C \not\equiv s_3$ *then* $\Gamma \vdash_r C : s$ *for some sort* s.
8. *If* $\Gamma \vdash_r Fa : C$ *then there are* x, A, B *such that* $\Gamma \vdash_r F : \Pi_{x:A}.B$, $\Gamma \vdash_r a : A$ *and* $C =_r B[x{:=}a]$ *and if* $C \not\equiv B[x{:=}a]$ *then* $\Gamma \vdash_r C : s$ *for some* s.

Proof. By induction on the derivation $\Gamma \vdash_r M : C$ where M is of the right form. ⊠

Lemma 22 (Correctness of Types for $\vdash_r$ and $\to_r$)**.**
If $\Gamma \vdash_r A : B$ *then* ($B \in \boldsymbol{S}$ *or* $\Gamma \vdash_r B : s$ *for some sort* s).

Proof. By induction on the derivation $\Gamma \vdash_r A : B$ using Generation and Substitution Lemmas for the (appl) case. ⊠

Lemma 23 (Subject Reduction for $\vdash_r$ and $\to_r$)**.**
If $\Gamma \vdash_r A : B$ *and* $A \twoheadrightarrow_r A'$ *then* $\Gamma \vdash_r A' : B$.

Proof. First we prove by simultaneous induction on the derivation $\Gamma \vdash_r A : B$ the following (use Correctness of Types for the (λ), (l), (l') and (appl) cases and also Generation and Substitution for the (appl) case):

- If $\Gamma \vdash_r A : B$ and $A \to_r A'$ then $\Gamma \vdash_r A' : B$.
- If $\Gamma \vdash_r A : B$ and $\Gamma \to_r \Gamma'$ then $\Gamma' \vdash_r A : B$.

Then we prove the lemma by induction on the derivation $A \twoheadrightarrow_r A'$.

We only show the case $\Gamma \vdash_{\beta'} l'Aa : D[z := a]$ comes from $\Gamma \vdash_{\beta'} l'A : \Pi_{z:C}.D$ and $\Gamma \vdash_{\beta'} a : C$ where $l'Aa \to_{\beta'} a$. By Generation, $\Pi_{z:C}.D =_{\beta'} \Pi_{z:\Pi_{x:A}.B}.\Pi_{x:A}.B$ where $z \neq x$. Hence $C =_{\beta'} \Pi_{x:A}.B =_{\beta'} D$ and $z \notin \mathrm{FV}(D)$, so $D[z := a] \equiv D =_{\beta'} C$. By correctness of types, $\Gamma \vdash_{\beta'} \Pi_{z:C}.D : s$ for some s and by Generation $\Gamma, z : C \vdash_{\beta'} D : s'$ for some s'. By Substitution Lemma, $\Gamma \vdash D[z := a] : s'$ and so $\Gamma \vdash D : s'$. Hence by $(\mathrm{conv}_{\beta'})$, $\Gamma \vdash a : D \equiv D[z := a]$ and we are done. ⊠

Corollary 24 (Reduction preserves types for $\vdash_r$ and $\to_r$)**.**
If $\Gamma \vdash_r A : B$ *and* $B \twoheadrightarrow_r B'$ *then* $\Gamma \vdash_r A : B'$.

4. DFPTSs do not faithfully capture Church's typing

The next lemma shows that the direct interpretation of a PTS into a DFPTS, although an interpretation, does not preserve the type information that may be needed later.

Lemma 25 (DFPTSs interpret Church's typing, but do not preserve types)**.**

1. *It is not the case that* $[\Gamma]^c \vdash^{\mathcal{S}}_{\overline{\beta}} [A]^c : [B]^c$ *implies* $\Gamma \vdash^{\mathcal{S}}_{\beta} A : B$.
2. *It is not the case that* $[\Gamma]^c \vdash^{\mathcal{S}}_{\overline{\beta}} [A]^c : B$ *implies there is a* B_1 *such that* $B =_{\overline{\beta}} [B_1]^c$ *and* $\Gamma \vdash^{\mathcal{S}}_{\beta} A : B_1$.
3. *If* $\Gamma \vdash^{\mathcal{S}}_{\beta} A : B$ *then* $[\Gamma]^c \vdash^{\mathcal{S}}_{\overline{\beta}} [A]^c : [B]^c$.

Proof. 1. Consider a $\overline{\lambda}\mathcal{S}$ such that $(s_1, s_1, s_3) \in \boldsymbol{R}$. Note that $[\lambda_{x:z}.x]^c \equiv \lambda_x.x$ and $y : s_1, z : s_2 \vdash^{\mathcal{S}}_{\overline{\beta}} \lambda_x.x : \Pi_{x:y}.y$. Even if also $(s_2, s_2, s_4) \in \boldsymbol{R}$, and even if $\overline{\lambda}\mathcal{S}$ is L-complete (see Definition 28), we still cannot show that $y : s_1, z : s_2 \vdash^{\mathcal{S}}_{\beta} \lambda_{x:z}.x : \Pi_{x:y}.y$. Otherwise, by Generation $\Pi_{x:y}.y =_\beta \Pi_{x:z}.C$, and hence by CR, $y =_\beta z$ absurd.
2. Take the same example as 1. above. We cannot find B_1 such that $[B_1]^c =_{\overline{\beta}} \Pi_{x:y}.y$ and $\Gamma \vdash^{\mathcal{S}}_{\beta} \lambda_{x:z}.x : B_1$. Otherwise, $B_1 =_\beta \Pi_{x:y}.y$ and by generation, $B_1 =_\beta \Pi_{x:z}.C$ and $\Pi_{x:y}.y =_\beta \Pi_{x:z}.C$, absurd.
3. By induction on $\Gamma \vdash^{\mathcal{S}}_{\beta} A : B$, using Lemma 10.4. ⊠

5. Under which circumstances can DFPTSs be used to capture Church typing?

Recall Remark 6 where we discussed how we intend $\mathsf{Label}A(\lambda_x.b)$ to be a representation of $\lambda_{x:A}.b$, i.e., the first argument of Label is to be the type of the variable x. Recall also that we showed that $\mathsf{Label}A(\lambda_y.b) \twoheadrightarrow_{\overline{\beta}} \lambda_y.b$. Hence, we would expect $\mathsf{Label}A(\lambda_y.b)$ to have the same type as $\lambda_y.b$. In order to type

$\mathsf{Label}A(\lambda_y.b)$ in a DFPTS, Label and $\mathsf{Label}A$ must also be typeable. Let us start by discussing when Label is typeable.

Example 26 (Type checking $\mathsf{Label} \equiv \lambda_{uzx}.zx$ in a DFPTS)**.** *Let us see under what circumstances* $\mathsf{Label} \equiv \lambda_{uzx}.zx$ *can be typechecked in* $\mathcal{T}_c$ *with* $\vdash_{\overline{\beta}}$*. We will attempt to solve the type judgement* $\Gamma \vdash_{\overline{\beta}} \lambda_{uzx}.zx :?$

By the (λ_c) *rule, it is obvious that for some* $t_1, t_2, t_3,\ t_4,\ s_3, s_4, s_5$*:*

1. $\Gamma \vdash_{\overline{\beta}} \lambda_{uzx}.zx : \Pi_{u:t_1}.\Pi_{z:t_2}.\Pi_{x:t_3}.t_4$ *where 2.* $\Gamma \vdash_{\overline{\beta}} \Pi_{u:t_1}.\Pi_{z:t_2}.\Pi_{x:t_3}.t_4 : s_5$
3. $\Gamma, u : t_1 \vdash_{\overline{\beta}} \lambda_{zx}.zx : \Pi_{z:t_2}.\Pi_{x:t_3}.t_4$*.where 4.* $\Gamma, u : t_1 \vdash_{\overline{\beta}} \Pi_{z:t_2}.\Pi_{x:t_3}.t_4 : s_4$
5. $\Gamma, u : t_1, z : t_2 \vdash_{\overline{\beta}} \lambda_x.zx : \Pi_{x:t_3}.t_4$ *where 6.* $\Gamma, u : t_1, z : t_2 \vdash_{\overline{\beta}} \Pi_{x:t_3}.t_4 : s_3$
7. $\Gamma, u : t_1, z : t_2, x : t_3 \vdash_{\overline{\beta}} zx : t_4$.

Now we turn to (app) on 7 and we get $t_2 \equiv \Pi_{x:t_3}.t_4$ *and* $z \notin \text{FV}(\Pi_{x:t_3}.t_4)$.

- *To derive 6.* $\Gamma, u : t_1, z : t_2 \vdash_{\overline{\beta}} \Pi_{x:t_3}.t_4 : s_3$ *it is enough by weakening to derive* $\Gamma, u : t_1 \vdash_{\overline{\beta}} \Pi_{x:t_3}.t_4 : s_3$*. For this we need* s_1, s_2 *such that:*

  10. $\Gamma, u : t_1 \vdash_{\overline{\beta}} t_3 : s_1$*, 11.* $\Gamma, u : t_1, x : t_3 \vdash_{\overline{\beta}} t_4 : s_2$ *and 12.* $(s_1, s_2, s_3) \in \boldsymbol{R}$.
- *To derive 4.* $\Gamma, u : t_1 \vdash_{\overline{\beta}} \Pi_{z:\Pi_{x:t_3}.t_4}.\Pi_{x:t_3}.t_4 : s_4$ *we need: 13.* $(s_3, s_3, s_4) \in \boldsymbol{R}$.
- *To derive 2.* $\Gamma \vdash_{\overline{\beta}} \Pi_{u:t_1}.\Pi_{z:\Pi_{x:t_3}.t_4}.\Pi_{x:t_3}.t_4 : s_5$ *we need* s' *such that:*

  14. $\Gamma \vdash_{\overline{\beta}} t_1 : s_1'$ *and* $(s_1', s_4, s_5) \in \boldsymbol{R}$.

To summarise, for $\Gamma \vdash_{\overline{\beta}} \mathsf{Label} : \Pi_{u:t_1}.\Pi_{x:t_3}.t_4 \longrightarrow \Pi_{x:t_3}.t_4$ *we need*[2]*:*

L1. $(s_1, s_2, s_3), (s_3, s_3, s_4), (s_1', s_4, s_5) \in \boldsymbol{R}$.

L2. $\Gamma \vdash_{\overline{\beta}} t_1 : s_1'$.

L3. $\Gamma, u : t_1 \vdash_{\overline{\beta}} t_3 : s_1$.

L4. $\Gamma, u : t_1, x : t_3 \vdash_{\overline{\beta}} t_4 : s_2$.

Since Label saves the type of the abstracted variable in its first argument, assuming that $\Gamma \vdash_{\overline{\beta}} \lambda_x.b : \Pi_{x:A}.B$, we would expect $\Gamma \vdash_{\overline{\beta}} \mathsf{Label}A(\lambda_x.b) : \Pi_{x:A}.B$. Let us use Example 26 to see under which circumstances $\mathsf{Label}A$ is typeable and when is $\Gamma \vdash_{\overline{\beta}} \mathsf{Label}A(\lambda_x.b) : \Pi_{x:A}.B$ considering that $\Gamma \vdash_{\overline{\beta}} \lambda_x.b : \Pi_{x:A}.B$.

Example 27 (When can we have $\Gamma \vdash_{\overline{\beta}} \mathsf{Label}A(\lambda_x.b) : \Pi_{x:A}.B?$)**.** *Assume a DFPTS where* $\Gamma \vdash_{\overline{\beta}} \lambda_x.b : \Pi_{x:A}.B$*. In order to have* $\Gamma \vdash_{\overline{\beta}} \mathsf{Label}A(\lambda_x.b) : \Pi_{x:A}.B$*,*

[2]Recall that $z \notin \text{FV}(\Pi_{x:t_3}.t_4)$.

we also need to type in Γ, Label *and* $\mathsf{Label}A$. *Looking back at Example 26, we see that in the type of* Label *and in L1..L4, we need* $t_1 \equiv s_1$, $t_3[u := A] \equiv A$ *and* $t_4[u := A] \equiv B$. *We also need* $\Gamma \vdash A : s_1$, $\Gamma \vdash_{\overline{\beta}} \Pi_{x:A}.B : s_3$ *and* $\Gamma, x : A \vdash_{\overline{\beta}} b : B : s_2$. *Hence, we get* $\Gamma \vdash_{\overline{\beta}} \mathsf{Label}A : \Pi_{x:A}.B \longrightarrow \Pi_{x:A}.B$ *and* $\Gamma \vdash_{\overline{\beta}} \mathsf{Label}A(\lambda_x.b) : \Pi_{x:A}.B$.

This way, the conditions to type Label *which behaves as type saver for the variable* x *are as follows:*

c1. $(s_1, s_2, s_3), (s_3, s_3, s_4), (s'_1, s_4, s_5) \in \boldsymbol{R}$ *and* $(s_1, s'_1) \in \boldsymbol{A}$.

c2. $\Gamma \vdash_{\overline{\beta}} \mathsf{Label} : \Pi_{u:s_1}.(\Pi_{x:A}.B \longrightarrow \Pi_{x:A}.B)$ *where* $\Gamma, x : A \vdash_{\overline{\beta}} B : s_2$.

In order to type $\mathsf{Label}A$, *it must hold that* $\Gamma \vdash_{\overline{\beta}} A : s_1$.

This way by (app), $\Gamma \vdash_{\overline{\beta}} \mathsf{Label}A : \Pi_{x:A}.B \longrightarrow \Pi_{x:A}.B$.

Hence if $\Gamma \vdash_{\overline{\beta}} \lambda_x.b : \Pi_{x:A}.B$ *we get by (app)* $\Gamma \vdash_{\overline{\beta}} \mathsf{Label}A(\lambda_x.b) : \Pi_{x:A}.B$.

To summarise all these findings, in order to type $\mathsf{Label} \equiv \lambda_{uzx}.zx$ in a DF-PTS and to obtain the expected behaviour that: $\Gamma \vdash_{\overline{\beta}} \mathsf{Label}A(\lambda_x.b) : \Pi_{x:A}.B$ whenever $\Gamma \vdash_{\overline{\beta}} (\lambda_x.b) : \Pi_{x:A}.B$ the following conditions need to hold:

- The DFPTS has $(s_1, s_2, s_3), (s_3, s_3, s_4), (s'_1, s_4, s_5) \in \boldsymbol{R}$, $(s_1, s'_1) \in \boldsymbol{A}$,
- $\Gamma \vdash_{\overline{\beta}} A : s_1$,
- $\Gamma, x : A \vdash_{\overline{\beta}} B : s_2$.

The next definition recalls the notion of L-compatible sorts which will be used to capture these findings.

Definition 28. [L-compatible and L-complete sorts]

- We call a tuple of sorts $(s'_1, s_1, s_2, s_3, s_4, s_5)$ L-compatible if $(s_1, s_2, s_3), (s_3, s_3, s_4), (s'_1, s_4, s_5) \in \boldsymbol{R}$ and $(s_1, s'_1) \in \boldsymbol{A}$.
- We call (s_1, s_2) L-complete if there are s'_1, s_3, s_4, s_5 such that $(s'_1, s_1, s_2, s_3, s_4, s_5)$ is L-compatible.

Example 29.

1. *Let us see which corresponding systems of the cube have L-compatible tuple(s)* $(s'_1, s_1, s_2, s_3, s_4, s_5)$. *Since we are working with the cube and also*

with L-compatible tuples, the following must hold:

- $(s_1, s_1') \in \boldsymbol{A}$, $s_1 = *$, $s_2 = *$ *and* $s_1' = \Box$.
- $(*, *, s_3), (s_3, s_3, s_4), (\Box, s_4, s_5) \in \boldsymbol{R}$ *and hence* $* = s_3 = s_4 = s_5$.

Hence, $(\Box, *, *, *, *, *)$ *is the only L-compatible tuple and it holds in* $\overline{\lambda 2}$, $\overline{\lambda}P2$, $\overline{\lambda}\omega$ *and* $\overline{\lambda}P\omega$. *In these systems, the only L-complete pair of sorts is* $(*, *)$.

2. *If we take a DFPTS where* $\boldsymbol{S} = \{*, \Box\}$, $\boldsymbol{A} = \{(*, *)\}$ *and* $\boldsymbol{R} = \{(*, *, *), (*, \Box, \Box), (\Box, \Box, \Box), (\Box, *, *)\}$ *then* $(\Box, *, *, *, *, *)$ *and* $(\Box, *, \Box, \Box, \Box, \Box)$ *are L-compatible. Hence,* $(*, *)$ *and* $(*, \Box)$ *are L-complete.*

The next lemma discusses under what conditions Label is typeable in a DFPTS.

Lemma 30. *Assume a DFPTS with L-compatible sorts* $(s_1', s_1, s_2, s_3, s_4, s_5)$ *such that for some* Γ, B *and some* x *and* u *distinct we have* $\Gamma, u : s_1, x : u \vdash_{\overline{\beta}} B : s_2$. *The following holds:* $\Gamma \vdash_{\overline{\beta}} \mathsf{Label} : \Pi_{u:s_1}.(\Pi_{x:u}.B \longrightarrow \Pi_{x:u}.B) : s_5$.

Proof.

1. $\Gamma, u : s_1 \vdash_{\overline{\beta}} \Pi_{x:u}.B : s_3$ since $(s_1, s_2, s_3) \in \boldsymbol{R}$.
2. $\Gamma, u : s_1, z : \Pi_{x:u}.B \vdash_{\overline{\beta}} \Pi_{x:u}.B : s_3$ by 1., (weak), for z fresh.
3. $\Gamma, u : s_1 \vdash_{\overline{\beta}} \Pi_{z:\Pi_{x:u}.B}.\Pi_{x:u}.B : s_4$ by 1., 2., (Π) and $(s_3, s_3, s_4) \in \boldsymbol{R}$.
4. $\Gamma \vdash_{\overline{\beta}} s_1 : s_1'$ by Start Lemma since $(s_1, s_1') \in \boldsymbol{A}$.
5. $\Gamma \vdash_{\overline{\beta}} \Pi_{u:s_1}.\Pi_{z:\Pi_{x:u}.B}.\Pi_{x:u}.B : s_5$ by 3., 4., (Π) and $(s_1', s_4, s_5) \in \boldsymbol{R}$.
6. $\Gamma, u : s_1, z : \Pi_{x:u}.B, x : u$ is legal by 2., and Generation.
7. $\Gamma, u : s_1, z : \Pi_{x:u}.B, x : u \vdash_{\overline{\beta}} z : \Pi_{x:u}.B$ by 6., and Start Lemma.
8. $\Gamma, u : s_1, z : \Pi_{x:u}.B, x : u \vdash_{\overline{\beta}} x : u$ by 6., and Start Lemma.
9. $\Gamma, u : s_1, z : \Pi_{x:u}.B, x : u \vdash_{\overline{\beta}} zx : B$ by 7., 8., and (app).
10. $\Gamma, u : s_1, z : \Pi_{x:u}.B \vdash_{\overline{\beta}} \lambda_x.zx : \Pi_{x:u}.B$ by 9., 2., and (λ_c).
11. $\Gamma, u : s_1 \vdash_{\overline{\beta}} \lambda_{zx}.zx : \Pi_{z:\Pi_{x:u}.B}.\Pi_{x:u}.B$ by 10., 3., and (λ_c).

12. $\Gamma \vdash_{\overline{\beta}} \lambda_{uzx}.zx : \Pi_{u:s_1}.\Pi_{z:\Pi_{x:u}.B}.\Pi_{x:u}.B : s_5$ by 11., 5., and (λ_c).

Hence, $\Gamma \vdash_{\overline{\beta}} \mathsf{Label} : \Pi_{u:s_1}.(\Pi_{x:u}.B \longrightarrow \Pi_{x:u}.B) : s_5$. ⊠

Example 31.

1. *Since* $(\Box, *, *, *, *, *)$ *is L-compatible in* $\overline{\lambda 2}$, $\overline{\lambda}P2$, $\overline{\lambda}\omega$ *and* $\overline{\lambda}P\omega$, *we have in these systems that for any* B, Γ *and* x, u *mutually exclusive, if* $\Gamma, u : *, x : u \vdash_{\overline{\beta}} B : *$ *then* $\Gamma \vdash_{\overline{\beta}} \mathsf{Label} : \Pi_{u:*}.(\Pi_{x:u}.B \longrightarrow \Pi_{x:u}.B) : *$.
2. *If we take the DFPTS where* $\boldsymbol{S} = \{*, \Box\}$, $\boldsymbol{A} = \{(*, *)\}$ *and* $\boldsymbol{R} = \{(*, *, *), (*, \Box, \Box), (\Box, \Box, \Box), (\Box, *, *)\}$ *then since* $(\Box, *, *, *, *, *)$ *and* $(\Box, *, \Box, \Box, \Box, \Box)$ *are L-compatible. Let* $s \in \{*, \Box\}$ *and* B, Γ, x, u *mutually exclusive such that* $\Gamma, u : *, x : u \vdash_{\overline{\beta}} B : s$.
   We have $\Gamma \vdash_{\overline{\beta}} \mathsf{Label} : \Pi_{u:*}.(\Pi_{x:u}.B \longrightarrow \Pi_{x:u}.B) : s$.

By Example 31.2 we see that $\mathsf{Label} \in Term^* \cup Term^{\Box}$ and hence Label is not uniquely sorted. In the cube however (Example 31.1), $\mathsf{Label} \in Term^*$ is uniquely sorted. The next lemma and corollary show that in DFPTSs that preserve sorts, if Label is typeable then it is uniquely sorted.

Lemma 32. *Assume a DFPTS* $\overline{\lambda}\mathcal{S}$ *which preserves sorts such that*

$$\Gamma \vdash_{\overline{\beta}} \Pi_{u:s_1}.(\Pi_{x:u}.B \longrightarrow \Pi_{x:u}.B) : s_5.$$

There is a unique L-compatible tuple $(s_1', s_1, s_2, s_3, s_4, s_5)$.

Proof. By Generation, there are s_1', s_4 such that $\Gamma, u : s_1 \vdash_{\overline{\beta}} \Pi_{x:u}.B \longrightarrow \Pi_{x:u}.B : s_4$ and $\Gamma \vdash_{\overline{\beta}} s_1 : s_1'$ where $(s_1', s_4, s_5) \in \boldsymbol{R}$. By Start Lemma, $(s_1, s_1') \in \boldsymbol{A}$. By Generation, there are s_3', s_3 such that $\Gamma, u : s_1, z : \Pi_{x:u}.B \vdash_{\overline{\beta}} \Pi_{x:u}.B : s_3'$ and $\Gamma, u : s_1 \vdash_{\overline{\beta}} \Pi_{x:u}.B : s_3$ where $(s_3, s_3', s_4) \in \boldsymbol{R}$. By Weakening and Preservation of Sorts, $s_3 = s_3'$.
Since $\Gamma, u : s_1 \vdash_{\overline{\beta}} \Pi_{x:u}.B : s_3$, by Preservation of Sorts and Generation, there is s_2 such that $\Gamma, u : s_1 \vdash_{\overline{\beta}} u : s_1$ and $\Gamma, u : s_1, x : u \vdash_{\overline{\beta}} B : s_2$ and $(s_1, s_2, s_3) \in \boldsymbol{R}$. Hence, $(s_1', s_1, s_2, s_3, s_4, s_5)$ is L-compatible.
Take $(s_1'', s_1, s_2, s_3', s_4', s_5)$ L-compatible. As $(s_1, s_1'), (s_1, s_1'') \in \boldsymbol{A}$, then $s_1 = s_1'$

by Preservation of Sorts and Start Lemma.

Since $\Gamma, u : s_1, x : u \vdash_{\overline{\beta}} B : s_2$ and $\Gamma, u : s_1 \vdash_{\overline{\beta}} u : s_1$, by (s_1, s_2, s_3'), $\Gamma, u : s_1 \vdash_{\overline{\beta}} \Pi_{x:u}.B : s_3'$. But $\Gamma, u : s_1 \vdash_{\overline{\beta}} \Pi_{x:u}.B : s_3$ and hence by Preservation of Sorts, $s_3 = s_3'$.

Since $\Gamma, u : s_1 \vdash_{\overline{\beta}} \Pi_{x:u}.B : s_3$, by weakening $\Gamma, u : s_1, z : \Pi_{x:u}.B \vdash_{\overline{\beta}} \Pi_{x:u}.B : s_3$ and by (Π) and (s_3, s_3, s_4') we have $\Gamma, u : s_1 \vdash_{\overline{\beta}} \Pi_{x:u}.B \longrightarrow \Pi_{x:u}.B : s_4'$ and hence by Preservation of Sorts, $s_4 = s_4'$. ⊠

Corollary 33. *Assume a DFPTS $\lambda\mathcal{S}$ which preserves sorts. Then we have:*

1. *There exist Γ, B and distinct u, x such that:*

$$\Gamma \vdash_{\overline{\beta}} \mathsf{Label} : \Pi_{u:s_1}.(\Pi_{x:u}.B \longrightarrow \Pi_{x:u}.B) : s_5$$

   iff

   There is a unique L-compatible tuple $(s_1', s_1, s_2, s_3, s_4, s_5)$.

2. *If Label is typeable then there is a unique sort s such that $\mathsf{Label} \in Term^s$.*

Proof.

1. If $\Gamma \vdash_{\overline{\beta}} \mathsf{Label} : \Pi_{u:s_1}.(\Pi_{x:u}.B \longrightarrow \Pi_{x:u}.B) : s_5$ then by Lemma 32, there is a unique L-compatible tuple $(s_1', s_1, s_2, s_3, s_4, s_5)$. If there is a unique L-compatible tuple $(s_1', s_1, s_2, s_3, s_4, s_5)$ then since for mutually distinct u, x, we can construct Γ, B such that $\Gamma, u : s_1, x : u \vdash_{\overline{\beta}} B : s_2$, we are done by Lemma 30.
2. If Label is typeable then following Example 27, we can deduce that for some L-compatible tuple $(s_1', s_1, s_2, s_3, s_4, s_5)$ and for some Γ, B and distinct x, u, we have $\Gamma \vdash_{\overline{\beta}} \mathsf{Label} : \Pi_{u:s_1}.(\Pi_{x:u}.B \longrightarrow \Pi_{x:u}.B) : s_5$. By Lemma 32, s_5 is unique and $\mathsf{Label} \in Term^{s_5}$.

⊠

Starting from $\Gamma \vdash_{\overline{\beta}} A : s_1$ and $\Gamma \vdash_{\overline{\beta}} \Pi_{x:A}.B[u := A] : s_3$ (needed for Correctness of Types and hence Subject Reduction to hold), we ensure that for every b where $\Gamma, x : A \vdash_{\overline{\beta}} b : B[u := A] : s_2$ (and hence by (λ_c) $\Gamma \vdash_{\overline{\beta}} \lambda_x.b : \Pi_{x:A}.B[u := A]$), we have $\Gamma \vdash_{\overline{\beta}} \mathsf{Label}A(\lambda_x.b) : \Pi_{x:A}.B[u := A]$.

Lemma 34. *Assume a DFPTS with L-complete (s_1, s_2). Let Γ, A, B, u, x such that $\Gamma \vdash_{\overline{\beta}} A : s_1$ and $\Gamma, u : s_1, x : u \vdash_{\overline{\beta}} B : s_2$.*

1. *There is s_5 such that $\Gamma \vdash_{\overline{\beta}} \mathsf{Label} : \Pi_{u:s_1}.(\Pi_{x:u}.B \longrightarrow \Pi_{x:u}.B) : s_5$.*
2. *There are s_3, s_4 such that $\Gamma \vdash_{\overline{\beta}} \mathsf{Label}A : \Pi_{x:A}.B[u := A] \longrightarrow \Pi_{x:A}.B[u := A] : s_4$ and $\Gamma \vdash_{\overline{\beta}} \Pi_{x:A}.B[u := A] : s_3$.*
3. *For every b where $\Gamma, x : A \vdash_{\overline{\beta}} b : B[u := A]$, we have $\Gamma \vdash_{\overline{\beta}} \mathsf{Label}A(\lambda_x.b) : \Pi_{x:A}.B[u := A] : s_3$.*

Proof.

1. By Lemma 30, $\Gamma \vdash_{\overline{\beta}} \mathsf{Label} : \Pi_{u:s_1}.(\Pi_{x:u}.B \longrightarrow \Pi_{x:u}.B) : s_5$.
2. By definition of L-complete (s_1, s_2), there is an L-compatible tuple of sorts $(s_1', s_1, s_2, s_3, s_4, s_5)$. By Substitution Lemma 20, $\Gamma, x : A \vdash_{\overline{\beta}} B[u := A] : s_2$. Since $(s_1, s_2, s_3) \in \boldsymbol{R}$, $\Gamma \vdash_{\overline{\beta}} \Pi_{x:A}.B[u := A] : s_3$ is by (Π). Since $(s_3, s_3, s_4) \in \boldsymbol{R}$, again by (Π), $\Gamma \vdash_{\overline{\beta}} \Pi_{x:A}.B[u := A] \longrightarrow \Pi_{x:A}.B[u := A] : s_4$. By (app) since $x \notin \mathrm{FV}(A) \subseteq \mathrm{DOM}(\Gamma)$, $\Gamma \vdash_{\overline{\beta}} \mathsf{Label}A : \Pi_{x:A}.B[u := A] \longrightarrow \Pi_{x:A}.B[u := A]$.
3. By (λ_c), $\Gamma \vdash \lambda_x.b : \Pi_{x:A}.B[u := A]$.
   By (app) $\Gamma \vdash_{\overline{\beta}} \mathsf{Label}A(\lambda_x.b) : \Pi_{x:A}.B[u := A] : s_3$.

⊠

Corollary 35. *In a DFPTS with L-complete (s_1, s_2), where $\Gamma \vdash_{\overline{\beta}} A : s_1$ and $\Gamma, u : s_1, x : u \vdash_{\overline{\beta}} B : s_2$, we have:*
$\Gamma \vdash_{\overline{\beta}} \lambda_x.b : \Pi_{x:A}.B[u := A]$ iff $\Gamma \vdash_{\overline{\beta}} \mathsf{Label}A(\lambda_x.b) : \Pi_{x:A}.B[u := A]$.

Proof. Assume $\Gamma \vdash_{\overline{\beta}} \lambda_x.b : \Pi_{x:A}.B[u := A]$. By Lemma 34.3, $\Gamma \vdash_{\overline{\beta}} \mathsf{Label}A(\lambda_x.b) : \Pi_{x:A}.B[u := A]$.
Assume $\Gamma \vdash_{\overline{\beta}} \mathsf{Label}A(\lambda_x.b) : \Pi_{x:A}.B[u := A]$. Since $\mathsf{Label}A(\lambda_x.b) \twoheadrightarrow_{\overline{\beta}} \lambda_x.b$, by Subject Reduction $\Gamma \vdash_{\overline{\beta}} \lambda_x.b : \Pi_{x:A}.B[u := A]$. ⊠

Definition 36. [L-complete DFPTSs] We say that a DFPTS is L-complete if every (s_1, s_2) for which there are Γ, A, B such that $\Gamma \vdash_{\overline{\beta}} A : s_1$ and $\Gamma, x : A \vdash_{\overline{\beta}} B : s_2$, it holds that (s_1, s_2) is L-complete.

Example 37. *Recall that in the extended calculus of constructions ECC we have:*

$$\begin{aligned}
\boldsymbol{S} = &\ \{*\} \cup \{\Box_n \text{ where } n \text{ is a nonnegative integer}\} \\
\boldsymbol{A} = &\ \{* : \Box_0\} \cup \{\Box_n : \Box_{n+1} \text{ where } n \text{ is a nonnegative integer}\} \\
\boldsymbol{R} = &\ \{(*,*,*), (*,*,\Box_n), (\Box_n,*,*), (\Box_n,*,\Box_m) \text{ where } 0 \leq n \leq m\} \\
&\cup \{(*,\Box_n,\Box_m) \text{ where } n \leq m\} \\
&\cup \{(\Box_n,\Box_m,\Box_r) \text{ where } 0 \leq n \leq r \text{ and } 0 \leq m \leq r\ \}
\end{aligned}$$

Let us show that the DFPTS corresponding to ECC is L-complete. Assume $\Gamma \vdash_{\overline{\beta}} A : s_1$ *and* $\Gamma, x : A \vdash_{\overline{\beta}} B : s_2$*. We will show that* (s_1, s_2) *is L-complete:*

- *If* $s_1 = s_2 = *$ *then* $(\Box_0, *, *, *, *, *)$ *is L-compatible.*
- *If* $s_1 = *$ *and* $s_2 = \Box_i$ *for* $i \geq 0$ *then* $(\Box_0, *, \Box_i, \Box_{i+1}, \Box_{i+2}, \Box_{i+3})$ *is L-compatible.*
- *If* $s_1 = \Box_i$ *and* $s_2 = *$ *for* $i \geq 0$ *then* $(\Box_{i+1}, \Box_i, *, \Box_{i+1}, \Box_{i+2}, \Box_{i+3})$ *is L-compatible.*
- *If* $s_1 = \Box_i$ *and* $s_2 = \Box_j$ *for* $i, j \geq 0$ *then* $(\Box_{i+1}, \Box_i, \Box_j, \Box_{i+j}, \Box_{i+j+1}, \Box_{i+j+2})$ *is L-compatible.*

Before showing that L-complete DFPTSs faithfully map Curry typing into Church typing, we need the following help lemma.

Lemma 38 (Convertible contexts)**.** *If* $\Gamma_1 =_r \Gamma_2$*,* $\Gamma_1 \vdash_r A : B$ *and* Γ_2 *is legal then* $\Gamma_2 \vdash_r A : B$*.*

Proof. By induction on the derivation $\Gamma_1 \vdash_r A : B$ using start and thinning lemmas for (start) and (conv). ⊠

The next lemma shows that L-complete DFPTSs with Label preserve types.

Lemma 39 (We can map Curry typing into Church typing)**.** *If* $\overline{\lambda}\mathcal{S}$ *is L-complete then:*

1. *If* $\Gamma \vdash^{\mathcal{S}}_{\beta} A : B$ *then* $[\Gamma]^L \vdash^{\mathcal{S}}_{\overline{\beta}} [A]^L : [B]^L$*.*

2. *If* $[\Gamma]^L \vdash^{\mathcal{S}}_{\overline{\beta}} [A]^L : B$ *then there is* B_1 *such that* $B =_{\overline{\beta}} [B_1]^L$ *and* $\Gamma \vdash^{\mathcal{S}}_{\beta} A : B_1$.
3. *If* $[\Gamma]^L \vdash^{\mathcal{S}}_{\overline{\beta}} B : s$ *then there is* B_1 *such that* $B =_{\overline{\beta}} [B_1]^L$ *and* $\Gamma \vdash^{\mathcal{S}}_{\beta} B_1 : s$.

Proof. 1. By induction on $\Gamma \vdash^{\mathcal{S}}_{\beta} A : B$. We only do the (λ) case. Assume $\Gamma \vdash^{\mathcal{S}}_{\beta} \lambda_{x:A}.b : \Pi_{x:A}.B$ comes from $\Gamma, x{:}A \vdash^{\mathcal{S}}_{\beta} b : B$ and $\Gamma \vdash^{\mathcal{S}}_{\beta} \Pi_{x:A}.B : s$. By IH, $[\Gamma]^L, x : [A]^L \vdash^{\mathcal{S}}_{\overline{\beta}} [b]^L : [B]^L$ and $[\Gamma]^L \vdash^{\mathcal{S}}_{\overline{\beta}} \Pi_{x:[A]^L}.[B]^L : s$ and by (λ_c), $[\Gamma]^L \vdash^{\mathcal{S}}_{\overline{\beta}} \lambda_x.[b]^L : \Pi_{x:[A]^L}.[B]^L$. By Generation, there is $(s_1, s_2, s) \in \boldsymbol{R}$ such that $[\Gamma]^L \vdash^{\mathcal{S}}_{\overline{\beta}} [A]^L : s_1$ and $[\Gamma]^L, x : [A]^L \vdash^{\mathcal{S}}_{\overline{\beta}} [B]^L : s_2$. Since $\overline{\lambda}\mathcal{S}$ is L-complete, (s_1, s_2) is L-complete. By Corollary 35, $[\Gamma]^L \vdash^{\mathcal{S}}_{\overline{\beta}} \mathsf{Label}[A]^L(\lambda_x.[b]^L) : \Pi_{x:[A]^L}.[B]^L$. Hence, $[\Gamma]^L \vdash^{\mathcal{S}}_{\overline{\beta}} [\lambda_x.b]^L : [\Pi_{x:A}.B]^L$.
2. and 3. simulatenously by induction on the derivation $[\Gamma]^L \vdash^{\mathcal{S}}_{\overline{\beta}} [A]^L : B$ or $[\Gamma]^L \vdash^{\mathcal{S}}_{\overline{\beta}} B : s$.

⊠

6. Interpreting Church's typing into Curry's typing without restricting the DFPTS

Section 5 showed that in order to translate Church's terms (terms of $\mathcal{T}$) into Curry's terms (terms of $\mathcal{T}_c$) without losing type information, we need to use DFPTSs that allow the typing of Label (the type saver). For these DFPTSs to behave well requires to be L-compatible and this condition can be restrictive. The reason behind this restrictiveness is the fact that for Label to be typed, sorts need to satisfy taxing conditions. Although L-complete DFPTSs exist (e.g., the DFPTS corresponding to ECC as shown in Example 37), ensuring the existence of L-compatible sorts $(s_1', s_1, s_2, s_3, s_4, s_5)$ can be quite restrictive. We require numerous conditions to hold and if the original PTS does not satisfy these conditions on sorts, then its corresponding one cannot handle Label and the type A of the variable x in $\lambda_{x:A}.B$ will be lost in the translations from a PTS to its corresponding DFPTS. For this reason, we presented the lPTSs which require no extra conditions to faithfully translate $\lambda_{x:A}.B$ from the PTS to the lPTS. The lPTSs approach is a more relaxed approach where the syntax

of terms is extended to include terms of the form lAB where l has similar behaviour to Label, but is primitive rather than defined and always comes with its two arguments. Hence, to type lAB we do not need the restrictions on sorts that we needed on DFPTSs for typing Label.

Compare the next lemma with Lemma 39 where we had to impose the L-complete condition on the DFPTS.

Lemma 40.

1. *If* $\Gamma \vdash^{\mathcal{S}}_{\beta} A : B$ *then* $[\Gamma]^l \vdash^{\mathcal{S}}_{\underline{\beta}} [A]^l : [B]^l$.
2. *If* $\Gamma \vdash^{\mathcal{S}}_{\underline{\beta}} A : B$ *then there are* Γ_1, A_1, B_1 *such that* $\Gamma =_{\underline{\beta}} [\Gamma_1]^l$, $B =_{\underline{\beta}} [B_1]^l$ *and* $\Gamma_1 \vdash^{\mathcal{S}}_{\beta} A_1 : B_1$ *and if* $A \equiv [A_2]^l$ *then* $A_1 \equiv A_2$ *else* $A =_{\underline{\beta}} [A_1]^l$.
3. *If* $\Gamma \vdash^{\mathcal{S}}_{\underline{\beta}} [A]^l : B$ *then there are* Γ_1, B_1 *such that* $\Gamma =_{\underline{\beta}} [\Gamma_1]^l$, $B =_{\underline{\beta}} [B_1]^l$ *and* $\Gamma_1 \vdash^{\mathcal{S}}_{\beta} A : B_1$.
4. *If* $[\Gamma]^l \vdash^{\mathcal{S}}_{\underline{\beta}} [A]^l : [B]^l$ *then* $\Gamma \vdash^{\mathcal{S}}_{\beta} A : B$.

Proof. 1. By induction on the derivation $\Gamma \vdash^{\mathcal{S}}_{\beta} A : B$.

2. By induction on the derivation $\Gamma \vdash^{\mathcal{S}}_{\underline{\beta}} [A]^l : B$.

3. This is a corollary of 2. above.

4. Similar to the proof given for the corresponding case of Lemma 39. ⊠

7. Interpretations in $\mathcal{T}_{l'}$

In Section 5 we showed that DFPTSs do not faithfully capture Church's typing and in Section 6 we showed that they could be made to capture Church's typing if $\mathsf{Label} \equiv \lambda_{uzx}.zx$ is typeable and a Church term $\lambda_{x:A}.b \in \mathcal{T}$ is translated as $[\lambda_{x:A}.b]^L \equiv \mathsf{Label}[A]^L(\lambda_x.[b]^L) \in \mathcal{T}_c$. This meant that we could only faithfully capture Church's typing inside L-complete DFPTSs which is too restrictive. In Section 6 we showed that Church's typing can be faithfully represented without any restrictions inside lPTSs, where new l-terms are added to $\mathcal{T}_c$. These terms are of the form $lA(\lambda_x.b)$ where l is a primitive built-in symbol that captures the meaning of Label but does need to be typed on its own. Here, l can be looked at as a parameterised constant which can only be used with its two arguments

A and $\lambda_x.b$. Since we don't need to type l or lA on their own, the rules and axioms needed to type $lA(\lambda_x.b)$ are exactly the same as those for typing $\lambda_x.b$ and so our lPTS is not restricted over the original PTS.

In this section we will check the mid-way where we add l'-terms to $\mathcal{T}_c$ and again use $l'A(\lambda_x.b)$ where l' is a primitive built-in symbol that captures the meaning of Label, but this time, $l'A$ is a term on its own and hence needs to be typed. In order to type $l'A$ we need to have at least $A \in Type^{s_1}$ and to find s_2 such that (s_1, s_2) are l'-complete. That is, we need to find s_3, s_4 such that $(s_1, s_4, s_2), (s_2, s_2, s_3) \in \boldsymbol{R}$. This is less restrictive than what Label demanded (we need two axioms less and 1 rule less).

In this section we establish the faithfulness of the interpretation of Church's typing in l'PTSs. First, we give the following definition:

Definition 41. [l'-complete l'PTS, l'-bachelor-free terms] We say that an l'PTS is l'-complete if every (s_1, s_2) for which there are Γ, A, B such that $\Gamma \vdash_{\beta'} A : s_1$ and $\Gamma \vdash_{\beta'} \Pi_{x:A}.B : s_2$, it holds that (s_1, s_2) is l'-complete.

We say that a term A is l'-bachelor-free if every occurrence of $l'B$ in A is immediately followed by a term of the form $(\lambda_x.b)$. We say that Γ is l'-bachelor-free if for any $y : A$ in Γ, A is l'-bachelor-free.

Lemma 42 (l'-complete l'PTSs preserve types)**.** *If $\tilde{\lambda}\mathcal{S}$ is l'-complete then:*

1. *If $\Gamma \vdash^{\mathcal{S}}_{\beta} A : B$ then $[\Gamma]' \vdash^{\mathcal{S}}_{\beta'} [A]' : [B]'$.*
2. *If $\Gamma \vdash^{\mathcal{S}}_{\beta'} A : B$ where Γ, A, B are all l'-bachelor-free, then there are Γ_1, A_1, B_1 such that $\Gamma =_{\beta'} [\Gamma_1]'$, $B =_{\beta'} [B_1]'$ and $\Gamma_1 \vdash^{\mathcal{S}}_{\beta} A_1 : B_1$ and if $A \equiv [A_2]'$ then $A_1 \equiv A_2$ else $A =_{\beta'} [A_1]'$.*
3. *If $\Gamma \vdash^{\mathcal{S}}_{\beta'} [A]' : B$ where Γ, B are all l'-bachelor-free, then there are Γ_1, B_1 such that $\Gamma =_{\beta'} [\Gamma_1]'$, $B =_{\beta'} [B_1]'$ and $\Gamma_1 \vdash^{\mathcal{S}}_{\beta} A : B_1$.*
4. *If $[\Gamma]' \vdash^{\mathcal{S}}_{\beta'} [A]' : [B]'$ then $\Gamma \vdash^{\mathcal{S}}_{\beta} A : B$.*

Proof. 1. By induction on the derivation $\Gamma \vdash^{\mathcal{S}}_{\beta} A : B$.

2. By induction on the derivation $\Gamma \vdash^{\mathcal{S}}_{\beta'} A : B$. We only do the case:

$\Gamma \vdash^{\mathcal{S}}_{\beta'} l'A(\lambda_x.b) : D[z := \lambda_x.b]$ comes from $\Gamma \vdash^{\mathcal{S}}_{\beta'} l'A : \Pi_{z:C}.D$ and $\Gamma \vdash^{\mathcal{S}}_{\beta'} \lambda_x.b : C$. Then by Generation, we have on the proof tree that for some B, s_1, s_2, s, E, F,

$\Gamma \vdash^{\mathcal{S}}_{\beta'} A : s_1$, $\Gamma \vdash^{\mathcal{S}}_{\beta'} \Pi_{x:A}.B : s_2$, $z \neq x$, $z \notin \text{DOM}(\Gamma)$, $\Gamma \vdash^{\mathcal{S}}_{\beta'} \Pi_{x:E}.F : s$, $\Gamma, x : E \vdash^{\mathcal{S}}_{\beta'} b : F$, $\Pi_{z:C}.D =_{\beta'} \Pi_{z:\Pi_{x:A}.B}.\Pi_{x:A}.B$ (hence $C =_{\beta'} \Pi_{x:A}.B =_{\beta'} D$) and $C =_{\beta'} \Pi_{x:E}.F$. Hence $A =_{\beta'} E$ and $B =_{\beta'} F$. By IH and Lemma 24, $\Gamma_1 \vdash^{\mathcal{S}}_{\beta} A_1 : s_1$, $\Gamma_2 \vdash^{\mathcal{S}}_{\beta} \Pi_{x:A_2}.B_2 : s_2$, $\Gamma_3, x : E_1 \vdash^{\mathcal{S}}_{\beta} b_1 : F_1$, where $[A_1]' =_{\beta'} [A_2]' =_{\beta'} [E_1]' =_{\beta'} A$ and $[b_1]' =_{\beta'} b$ and $[F_1]' =_{\beta'} [B_2]' =_{\beta'} B$. It is easy to show that $\Gamma_2, x : A_2 \vdash^{\mathcal{S}}_{\beta} b_1 : B_2$, and so, $\Gamma_2 \vdash^{\mathcal{S}}_{\beta} \lambda_{x:A_2}.b_1 : \Pi_{x:A_2}.B_2$ and we're done.

3. This is a corollary of 2. above. Note that $[A]'$ is l'-bachelor-free.

4. Similar to the proof given for the corresponding case of Lemma 39. Note that $[\Gamma]'$, $[A]'$ and $[B]'$ are all l'-bachelor-free. ⊠

8. Unicity of types, Classification and Consistency

In the previous sections we established desirable properties for all our type systems. Three properties have been left for this section: the unicity of types, classification and consistency lemmas. We discuss these properties in this section.

Lemma 43 (Unicity of Types for $\vdash_\beta$ and its failure for all other $\vdash_r$)**.**

1. *Let $\mathcal{S}$ be a functional specification. $\mathcal{S}$ satisfies the unicity of types with respect to $\vdash_\beta$ but fails to do so w.r.t. any other $\vdash_r$.*
2. *If $\mathcal{S}$ satisfies the unicity of types with respect to $\vdash_r$ then: If $\Gamma \vdash_r A_1 : B_1$ and $\Gamma \vdash_r A_2 : B_2$ and $A_1 =_r A_2$, then $B_1 =_r B_2$.*
3. *If $\mathcal{S}$ satisfies the unicity of types with respect to $\vdash_r$ then $\mathcal{S}$ preserves sorts w.r.t. $\vdash_r$.*
4. *If 2. holds for $\vdash_r$ then:*

   If $\Gamma \vdash_r B_1 : s$, $B_1 =_r B_2$ and $\Gamma \vdash_r A : B_2$ then $\Gamma \vdash_r B_2 : s$.

Proof.

1. For $\vdash_\beta$, use induction on the structure of A and the Generation Lemma. For the counterexample to all other relations, let $\Gamma = y : s_1, z : s_2$ and assume $(s_1, s_1, s_3) \in \boldsymbol{R}$ and $(s_2, s_2, s_4) \in \boldsymbol{R}$. Then:
   - Since $\Gamma \vdash_r \Pi_{x:y}.y : s_3$ and $\Gamma, x : y \vdash_r y : s_1$, we get $\Gamma \vdash_r \lambda_x.x : \Pi_{x:y}.y$.

- Since $\Gamma \vdash_r \Pi_{x:z}.z : s_4$ and $\Gamma, x : z \vdash_r z : s_2$, we get $\Gamma \vdash_r \lambda_x.x : \Pi_{x:z}.z$.

But $\Pi_{x:y}.y \neq_r \Pi_{x:z}.z$, hence contradicting 1.

2. This is a consequence of Church-Rosser of r-reduction, Subject Reduction of $\vdash_r$ and 1 above.
3. This is a direct consequence of 2 above.
4. Since $\Gamma \vdash_r A : B_2$ then by Correctness of Types for $\vdash_r$, either $B_2 \equiv s'$ or $\Gamma \vdash_r B_2 : s'$ for some sort s'.
   - If $B_2 \equiv s'$ then $B_1 \twoheadrightarrow_r B_2$ and by Subject Reduction of $\vdash_r$, $\Gamma \vdash_r B_2 : s$.
   - If $\Gamma \vdash_r B_2 : s'$ then by 2 above, $s =_r s'$ and hence $s \equiv s'$.

$\boxtimes$

In what follows, we use sorted variables. We divide $\mathcal{V}$ into countably infinite disjoint subsets $\mathcal{V}^s$ and use x^s, y^s, etc., to range over $\mathcal{V}^s$. With this partitioning of $\mathcal{V}$, variable renaming will respect sorts. That is for example, $\lambda_x.A$ is renamed to $\lambda_y.A[x := y]$ only if x and y belong to the same $\mathcal{V}^s$. We replace the two rules (start) and (weak) of Figure 2 by the rules in Figure 9. With these new rules of Figure 9, the second clause of the Generation Lemma changes to accommodate these sorts as shown in Lemma 44.

Lemma 44 (Revised Clauses of Generation)**.** *If (weak) and (start) of Figure 2 and (l′) of Figure 6 are replaced by (weak'), (start') and (l′') of Figure 9, then*

- *If* $\Gamma \vdash_r x : C$ *then for some* s, A, $x \equiv x^s$, $x : A \in \Gamma$, $C =_r A$, *and* $\Gamma \vdash_r C : s$.
- *If* $\Gamma \vdash_{\beta'} l'A : C$ *then there is* $(s_1, s_4, s_2), (s_2, s_2, s_3) \in \boldsymbol{R}$, *there is* $B, z \equiv z^{s_2}, x$ *where* $z \neq x, z \notin \text{DOM}(\Gamma)$, $\Gamma \vdash_r A : s_1$, $\Gamma, x : A \vdash_r B : s_4$, $C =_{\beta'} \Pi_{z:\Pi_{x:A}.B}.\Pi_{x:A}.B$ *and if* $C \not\equiv \Pi_{z:\Pi_{x:A}.B}.\Pi_{x:A}.B$ *then* $\Gamma \vdash_{\beta'} C : s'$ *for some* s'.

Proof. By induction on the derivation $\Gamma \vdash_r x : C$ $\boxtimes$

With the introduction of sorted variables, unicity of types still fails but the counterexample given in Lemma 43 needs to change to accommodate the new rules of Figure 9. This change dictates that $s_1 = s_2$. Hence, if $(s_1, s_4) \in \boldsymbol{A}$, $(s_1, s_1, s_3) \in \boldsymbol{R}$, $\Gamma = y : s_1, z : s_1$ and $x = x^{s_1}$, we can derive the following:
$\Gamma, x : y \vdash_r x : y$ and $\Gamma, x : z \vdash_r x : z$ by (start')

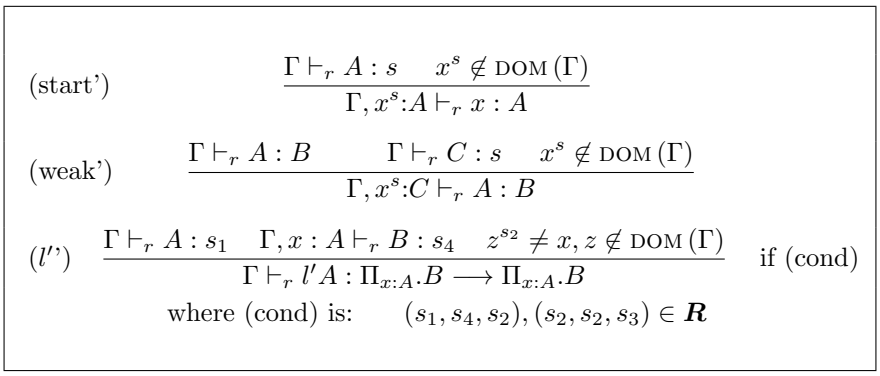

Figure 9: The start', weak' and l'' rules with sorted variables

$\Gamma \vdash_r \Pi_{x:y}.y : s_3$ and $\Gamma \vdash_r \Pi_{x:z}.z : s_3$ by (Π)

$\Gamma \vdash_r \lambda_x.x : \Pi_{x:y}.y$ and $\Gamma \vdash_r \lambda_x.x : \Pi_{x:z}.z$ by (λ_c).

Since $\Pi_{x:y}.y \neq_r \Pi_{x:z}.z$, unicity of types fails.

The counterexample given in Lemma 43 shows that not only unicity of types is lost but also unicity of sorts. The term $\lambda_x.x$ given in the proof of Lemma 43 belongs to $Term^{s_3} \cap Term^{s_4}$ and s_3 may be different from s_4. With our use of sorted variables we can rescue the unicity of sorts for injective specifications as we will see in Lemma 48. Note however that we cannot rescue unicity of types as we have just explained by the above example.

The next two lemmas are extensions of Lemmas in [3].

Lemma 45. *Let $\mathcal{S}$ be a specification. The following hold:*

1. *If $s' \in Type_r^s$ then $(s', s) \in \boldsymbol{A}$.*
2. *If $x \in Type_r^s$ then $x \in \mathcal{V}^{s'}$ where $(s, s') \in \boldsymbol{A}$.*
3. *$C \notin Type_r^s$ for $C \in \{\lambda_{x:A}.B, \lambda_x.B, lA(\lambda_x.B), l'A\}$.*
4. *If $\Pi_{x:A}.B \in Type_r^s$ then for some $(s_1, s_2, s) \in \boldsymbol{R}$, $A \in Type_r^{s_1}$ and $B \in Type_r^{s_2}$.*
5. *If $Fa \in Type_r^s$ then for some $(s_1, s_2, s_3) \in \boldsymbol{R}$, $(s, s_2) \in \boldsymbol{A}$, $F \in Term_r^{s_3}$ and $a \in Term_r^{s_1}$. Moreover, F cannot be of the form $l'A$ for some A.*

Proof.

1. By the Generation Lemma.
2. By the revised clause of Generation Lemma 44 using 1.

3. If $\Gamma \vdash_r C : s$ for $C \in \{\lambda_{x:A}.B, \lambda_x.B, lA(\lambda_x.B), l'A\}$, then by Generation Lemma 21, $s =_r \Pi_{z:A}.B$, contradicting Church-Rosser.
4. If $\Gamma \vdash_r \Pi_{x:A}.B : s$ then by Generation Lemma 44, for some $(s_1, s_2, s) \in \boldsymbol{R}$, $A \in Type_r^{s_1}$ and $B \in Type_r^{s_2}$.
5. If $\Gamma \vdash_r Fa : s$ then by Generation Lemma 44, there are z, H, G such that $\Gamma \vdash_r F : \Pi_{z:H}.G$, $\Gamma \vdash_r a : H$ and $s =_r G[z{:=}a]$. By correctness of Types, $\Gamma \vdash_r \Pi_{z:H}.G : s_3$ for some s_3. Hence $F \in Term_r^{s_3}$. By Generation (note that $=_r$ on sorts is $\equiv$), for some $(s_1, s_2, s_3) \in \boldsymbol{R}$, $\Gamma \vdash_r H : s_1$ (hence $a \in Term_r^{s_1}$), $\Gamma, z{:}H \vdash G : s_2$ and by Substitution Lemma $\Gamma \vdash_r G[z := a] : s_2$. By Subject Reduction, since $G[z{:=}a] \twoheadrightarrow_r s$, $\Gamma \vdash_r s : s_2$. By 1. above, $(s, s_2) \in \boldsymbol{A}$.
   If $F \equiv l'A$ then by Generation, there is B, $x \neq z$ (note that $z \notin \text{DOM}(\Gamma)$), such that $\Pi_{z:H}.G =_r \Pi_{z:\Pi_{x:A}.B}.\Pi_{x:A}.B$. Hence $G =_r \Pi_{x:A}.B$ and $G[z{:=}a] = \Pi_{x:A[z:=a]}.B[z{:=}a] =_r s$, absurd by Church-Rosser.

⊠

Lemma 46. *Let $\mathcal{S}$ preserve sorts w.r.t $\vdash_r$. The following hold:*

1. *If $s' \in Term_r^s$ then for some s'', $(s', s'') \in \boldsymbol{A}$ and $(s'', s) \in \boldsymbol{A}$.*
2. *If $x \in Term_r^s$ then $x \equiv x^s$.*
3. *If $M \in Term_r^s$ where $M \in \{\lambda_{x:A}.B, lA(\lambda_x.B)\}$ then $x \in \mathcal{V}^{s_1}$, $A \in Type_r^{s_1}$, and $B \in Term_r^{s_2}$ for some $(s_1, s_2, s) \in \boldsymbol{R}$.*
4. *If $\lambda_x.B \in Term_r^s$ then $x \in \mathcal{V}^{s_1}$, and $B \in Term_r^{s_2}$ for some $(s_1, s_2, s) \in \boldsymbol{R}$.*
5. *If $l'A \in Term_r^s$ then for some $(s_1, s_4, s_2) \in \boldsymbol{R}$, and $(s_2, s_2, s) \in \boldsymbol{R}$, $A \in Type_r^{s_1}$.*
6. *If $\Pi_{x:A}.B \in Term_r^s$ then for some $(s_1, s_2, s_3) \in \boldsymbol{R}$, $(s_3, s) \in \boldsymbol{A}$, $x \in \mathcal{V}^{s_1}$, $A \in Type_r^{s_1}$ and $B \in Type_r^{s_2}$.*
7. *If $Fa \in Term_r^s$ then for some $(s_1, s, s_3) \in \boldsymbol{R}$, $F \in Term_r^{s_3}$ and $a \in Term_r^{s_1}$.*

Proof.

1. By the Generation Lemma, Church-Rosser, Subject Reduction and Lemma 45.1.
2. By the revised clause of Generation Lemma 44 and Preservation of Sorts. Note that the $\mathcal{V}^{s_i}$s partition $\mathcal{V}$.
3. For $A \in Type_r^{s_1}$, and $B \in Term_r^{s_2}$ for some $(s_1, s_2, s) \in \boldsymbol{R}$, use Generation Lemma and Preservation of Sorts. For $x \in \mathcal{V}^{s_1}$, use Start Lemma and 2.

above.

4. Similar to 3. above.
5. By Generation Lemma and Preservation of Sorts.
6. If $\Gamma \vdash_r \Pi_{x:A}.B : C : s$ then by Generation, for some $(s_1, s_2, s_3) \in \boldsymbol{R}$, $\Gamma \vdash_r A : s_1$ and $\Gamma, x : A \vdash_r B : s_2$, $C =_r s_3$. Hence $A \in Type_r^{s_1}$ and $B \in Type_r^{s_2}$. By Church-Rosser and Subject Reduction, $\Gamma \vdash_r s_3 : s$ and by Lemma 45, $(s_3, s) \in \boldsymbol{A}$. Finally, by Start Lemma, $\Gamma \vdash_r x : A : s_1$ and hence by 2. above, $x \in \mathcal{V}^{s_1}$.
7. This is shown by Generation Lemma, Correctness of Types, Substitution Lemma and Preservation of Sorts.

$\boxtimes$

The following example shows that in general, the Classification Lemma fails for terms which contain l'.

Example 47. *Let $s_3 \neq s_3'$ and $s_4 \neq s_4'$ and let (s_1, s_4, s_2), (s_2, s_2, s_3), (s_1, s_4', s_2') and $(s_2', s_2', s_3') \in \boldsymbol{R}$.*
Let $\Gamma = y : s_1, y_1 : s_4, y_2 : s_4'$, and take $x \in \mathcal{V}^{s_1}$, $z_1 \in \mathcal{V}^{s_2}$ and $z_2 \in \mathcal{V}^{s_2'}$. Then, $\Gamma \vdash_r \Pi_{x:y}.y_1 : s_2$, $\Gamma \vdash_r \Pi_{x:y}.y_2 : s_2'$, $\Gamma \vdash_r l'y : \Pi_{z_1:\Pi_{x:y}.y_1}.\Pi_{x:y}.y_1 : s_3$ and $\Gamma \vdash_r l'y : \Pi_{z_2:\Pi_{x:y}.y_2}.\Pi_{x:y}.y_2 : s_3'$. Hence, $l'y \in Term_r^{s_3} \cap Term_r^{s_3'}$ with $s_3 \neq s_3'$.

If on the other hand we impose the condition that every term is l'-bachelor-free then the Unicity of Sorts will hold as we will see in the Classification Lemma 48. So, in the above example we would speak of $l'y(\lambda_x.b)$ which would belong to the same $Term^s$ as $\lambda_x.b$ and unicity of sorts for $\lambda_x.b$ would propagate to $l'y(\lambda_x.b)$.

Below, we prove the Classification Lemma for terms which are l'-bachelor-free.

Lemma 48 (Classification)**.** *Let $\mathcal{S}$ be an injective specification which preserves sorts. Assume M is l'-bachelor-free. The following hold:*

1. *If $M \in Term_r^s \cap Term_r^{s'}$ then $s = s'$.*
2. *If $M \in Type_r^s \cap Type_r^{s'}$ then $s = s'$.*

Proof. 1 and 2 are proved simultaneously by induction on M.

- If $s_1 \in Type_r^s \cap Type_r^{s'}$ then by Lemma 45, $(s_1, s), (s_1, s') \in \boldsymbol{A}$. By Start Lemma and Preservation of Sorts, $s = s'$.
- If $s_1 \in Term_r^s \cap Term_r^{s'}$ then by Lemma 46, there are s_2, s_3 such that $(s_1, s_2), (s_2, s), (s_1, s_3), (s_3, s') \in \boldsymbol{A}$. By Start Lemma and Preservation of Sorts, $s_2 = s_3$ and $s = s'$.
- If $x \in Type_r^s \cap Type_r^{s'}$ then by Lemma 45, $x \in \mathcal{V}^{s_1} \cap \mathcal{V}^{s_2}$, $(s, s_1), (s', s_2) \in \boldsymbol{A}$. Since the $\mathcal{V}^{s_i}$s partition $\mathcal{V}$, $s_1 = s_2$ and by injectivity, $s = s'$.
- If $x \in Term_r^s \cap Term_r^{s'}$ then by Lemma 46, $x \in \mathcal{V}^s \cap \mathcal{V}^{s'}$. Since the $\mathcal{V}^{s_i}$s partition $\mathcal{V}$, $s = s'$.
- If $\Pi_{x:A}.B \in Type_r^s \cap Type_r^{s'}$ then by Lemma 45, $(s_1, s_2, s), (s_1', s_2', s') \in \boldsymbol{R}$, $A \in Type_r^{s_1} \cap Type_r^{s_1'}$ and $B \in Type_r^{s_2} \cap Type_r^{s_2'}$. By IH, $s_1 = s_1'$ and $s_2 = s_2'$. Hence, $(s_1, s_2, s), (s_1, s_2, s') \in \boldsymbol{R}$. Since $\Gamma \vdash_r \Pi_{x:A}.B : s$ for some Γ, by Generation, $\Gamma \vdash_r A : s_3$, $\Gamma, x : A \vdash_r B : s_4$, hence $s_3 = s_1, s_4 = s_2$ and by (Π), $\Gamma \vdash_r \Pi_{x:A}.B : s$ and $\Gamma \vdash_r \Pi_{x:A}.B : s'$. By Preservation of Sorts, $s = s'$.
- If $\Pi_{x:A}.B \in Term_r^s \cap Term_r^{s'}$ then by Lemma 46, $(s_1, s_2, s_3), (s_1', s_2', s_3') \in \boldsymbol{R}$, $(s_3, s), (s_3', s') \in \boldsymbol{A}$, $x \in \mathcal{V}^{s_1} \cap \mathcal{V}^{s_1'}$, $A \in Type_r^{s_1} \cap Type_r^{s_1'}$ and $B \in Type_r^{s_2} \cap Type_r^{s_2'}$. By IH, $s_1 = s_1'$ and $s_2 = s_2'$. Hence, $(s_1, s_2, s_3), (s_1, s_2, s_3') \in \boldsymbol{R}$. Since $\Gamma \vdash_r \Pi_{x:A}.B : C$ for some Γ, by Generation, $\Gamma \vdash_r A : s_5$, $\Gamma, x : A \vdash_r B : s_4$, hence $s_5 = s_1, s_4 = s_2$ and by (Π), $\Gamma \vdash_r \Pi_{x:A}.B : s_3$ and $\Gamma \vdash_r \Pi_{x:A}.B : s_3'$. By Preservation of Sorts, $s_3 = s_3'$. By Start Lemma $\Gamma \vdash_r s_3 : s$ and $\Gamma \vdash_r s_3 : s'$. By Preservation of Sorts, $s = s'$.
- If $Fa \in Type_r^s \cap Type_r^{s'}$ then by Lemma 45, F is not of the form $l'A$, and $(s_1, s_2, s_3), (s_1', s_2', s_3') \in \boldsymbol{R}$, (s, s_2), $(s', s_2') \in \boldsymbol{A}$, $F \in Term_r^{s_3} \cap Term_r^{s_3'}$ and $a \in Term_r^{s_1} \cap Term_r^{s_1'}$. Use IH and injectivity twice.
- If $Fa \in Term_r^s \cap Term_r^{s'}$ then by Lemma 46, $(s_1, s, s_3), (s_1', s', s_3') \in \boldsymbol{R}$, $F \in Term_r^{s_3} \cap Term_r^{s_3'}$ and $a \in Term_r^{s_1} \cap Term_r^{s_1'}$.
  If $F \equiv l'A$ then $a \equiv \lambda_x.b$ and $\lambda_x.b \in Term_r^s \cap Term_r^{s'}$. Hence by IH, $s = s'$.
  If $F \not\equiv l'A$ for any A, then by IH on a, $s_1 = s_1'$ and by IH on F, $s_3 = s_3'$. By injectivity, $s = s'$.
- If $M \in \{\lambda_x.B, \lambda_{x:A}.B, lA(\lambda_x.B)\}$ then by Lemma 45, $M \notin Type_r^s \cap Type_r^{s'}$.

- If $M \in \{\lambda_x.B, \lambda_{x:A}.B, lA(\lambda_x.B)\}$ for $M \in Term_r^s \cap Term_r^{s'}$ then by Lemma 46, $x \in \mathcal{V}^{s_1} \cap \mathcal{V}^{s_1'}$, $A \in Type_r^{s_1} \cap Type_r^{s_1'}$, and $B \in Term_r^{s_2} \cap Term_r^{s_2'}$ for some $(s_1, s_2, s), (s_1', s_2', s') \in \boldsymbol{R}$. Use IH, Generation Lemma and Preservation of Sorts. ⊠

Corollary 49. *Let $\mathcal{S}$ be an injective specification which preserves sorts. Assume M is l'-bachelor-free and $\vdash_r$-legal. The following hold:*

1. *$M \in Term_r^s$ for some $s \in \boldsymbol{S}$; or*
2. *$M \in Type_r^s$ for some $s \in \boldsymbol{S}_T$; or*
3. *$M \equiv s$ for some $s \in \boldsymbol{S}_T$.*

Furthermore, 1..3 are mutually exclusive and s is unique.

Proof. The same proof as that of Corollary 25 of [3] applies here. ⊠

We say that $\lambda\mathcal{S}$ is top sort grounded if every top sort s is inhabited by another sort s' where $\vdash_\beta^{\mathcal{S}} s' : s$).

Next we move to consistency and show that if $\lambda\mathcal{S}$ is top sort grounded and consistent then $\overline{\lambda}\mathcal{S}$ (on $\mathcal{T}_c + \mathsf{Label}$), $\underline{\lambda}\mathcal{S}$ and $\tilde{\lambda}\mathcal{S}$ are consistent.

Lemma 50. *Let $\lambda\mathcal{S}$ be top sort grounded.*

1. *If $\lambda\mathcal{S}$ is L-complete and consistent then $\overline{\lambda}\mathcal{S}$ (on $\mathcal{T}_c + \mathsf{Label}$) is consistent.*
2. *If $\lambda\mathcal{S}$ is consistent then $\underline{\lambda}\mathcal{S}$ is consistent.*
3. *If $\lambda\mathcal{S}$ is l'-complete and consistent then $\tilde{\lambda}\mathcal{S}$ is consistent.*

Proof. All items are similar. We only do 2. Assume $\underline{\lambda}\mathcal{S}$ is inconsistent. Hence, every type is inhabited. In particular, for every $s \in \boldsymbol{S}$, $\Pi_{x:s}.x$ is inhabited. I.e., there is an $M \in \mathcal{T}_l$ such that $\vdash_{\underline{\beta}}^{\mathcal{S}} M : \Pi_{x:s}.x$. Hence by Lemma 39 and Reduction Preserves Types Lemma 24, there is $M_1 \in \mathcal{T}$ such that $[M_1]^l =_{\underline{\beta}} M$ and $\vdash_\beta^{\mathcal{S}} M_1 : \Pi_{x:s}.x$. Hence $y : s \vdash_\beta^{\mathcal{S}} M_1 y : y$ and for any C such that $\vdash_\beta^{\mathcal{S}} C : s$ we get by Substitution Lemma that $\vdash_\beta^{\mathcal{S}} M_1 C : C$. Hence, $\lambda\mathcal{S}$ is inconsistent. ⊠

9. Curry Style to Church Style

An interpretation in the reverse direction requires the availability of a type which can be used as the domain in the Church-style syntax which interprets

$$(\mathsf{A}\lambda) \quad \frac{\Gamma \vdash_\beta \lambda_{x:B}.M : \Pi_{x:B}.C}{\Gamma \vdash_\beta \lambda_{x:\mathsf{A}}.M : \Pi_{x:B}.C}$$

Figure 10: The λ-rule for Church's typing

the Curry-style abstract $\lambda_x.M$. For this purpose, let us introduce a new atomic constant A as a new dummy type, so that the interpretation of $\lambda_x.M$ in the Church-style system will be $\lambda_{x:\mathsf{A}}.M$. This, of course, requires a change in the definition of "type". We will also need to add the following rule to the system: See Fig. 10.

This rule corresponds to the inference in the Curry-style system from

$$\Gamma \vdash \mathsf{Label}B(\lambda_x.M) : \Pi_{x:B}.C$$

to

$$\Gamma \vdash \lambda_x.M : \Pi_{x:B}.C$$

since $[\lambda_{x:B}.M]^L$ is $\mathsf{Label}B(\lambda_x.M)$, and is necessary for the proof of the Theorem 54 below.

We can now define the mapping from the Curry-style syntax to the Church-style syntax:

Definition 51. The function $-^{\mathrm{Ch}}$ from the Curry-style syntax ($\mathcal{T}_c$, $\mathcal{T}_l$ and $\mathcal{T}'$) to the Church-style syntax $\mathcal{T}$ is defined as follows:

$x^{\mathrm{Ch}} \equiv x$ $\qquad$ $c^{\mathrm{Ch}} \equiv c$

$(\lambda_x.M)^{\mathrm{Ch}} \equiv \lambda_{x:\mathsf{A}}.M^{\mathrm{Ch}}$ $\qquad$ $(lB(\lambda_x.C))^{\mathrm{Ch}} \equiv (l'B(\lambda_x.C))^{\mathrm{Ch}} \equiv \lambda_{x:\mathsf{A}}.C^{\mathrm{Ch}}$

$(MN)^{\mathrm{Ch}} \equiv M^{\mathrm{Ch}}N^{\mathrm{Ch}}$ $\qquad$ $(\Pi_{x:B}.C)^{\mathrm{Ch}} \equiv \Pi_{x:B^{\mathrm{Ch}}}.C^{\mathrm{Ch}}$

We define Γ^{Ch} in the obvious way.

Note that $-^{\mathrm{Ch}}$ is partial on $\mathcal{T}'$ since we only define it for l'-bachelor-free terms.

The following lemma shows that the function $-^{\mathrm{Ch}}$ is closed under free variables, substitution and reduction.

Lemma 52. *Let $r \in \{\overline{\beta}, \underline{\beta}, \beta'\}$. When r is β', we assume that we are only working with l'-bachelor-free terms.*

1. *If M is in either $\mathcal{T}_c$, $\mathcal{T}_l$ or $\mathcal{T}'$, then* $\mathsf{FV}(M^{\mathrm{Ch}}) = \mathsf{FV}(M)$.
2. $(M[x := N])^{\mathrm{Ch}} \equiv M^{\mathrm{Ch}}[x := N^{\mathrm{Ch}}]$.
3. *If* $M \twoheadrightarrow_r N$ *then* $M^{\mathrm{Ch}} \twoheadrightarrow_\beta N^{\mathrm{Ch}}$.
4. *If* $M =_r N$ *then* $M^{\mathrm{Ch}} =_\beta N^{\mathrm{Ch}}$.
5. *If* $M^{\mathrm{Ch}} \twoheadrightarrow_\beta N$ *then* $N \equiv P^{\mathrm{Ch}}$ *and* $M \twoheadrightarrow_r P$.
6. *If* $M, N \in \mathcal{T}_c$ *and* $M^{\mathrm{Ch}} =_\beta N^{\mathrm{Ch}}$ *then* $M =_{\overline{\beta}} N$.

Proof. 1. and 2. By induction on M.

3. First show by induction on the derivation of $M \to_r N$ that if $M \to_r N$ then $M^{\mathrm{Ch}} \to_\beta N^{\mathrm{Ch}}$, and then we show the lemma by induction on the length of the derivation $M \twoheadrightarrow_r N$.

4. Use 3 and CR of r.

5. First show by induction on $M \in \mathcal{T}_c$ that if $M^{\mathrm{Ch}} \to_\beta N$ then $N \equiv P^{\mathrm{Ch}}$ and $M \to_r P$. Then show the lemma by induction on the length of the derivation $M^{\mathrm{Ch}} \twoheadrightarrow_\beta N$.

6. By CR, $M^{\mathrm{Ch}} \twoheadrightarrow_\beta P \ {}_\beta\!\!\twoheadleftarrow N^{\mathrm{Ch}}$. By 5, $P \equiv Q_1^{\mathrm{Ch}} \equiv Q_2^{\mathrm{Ch}}$ where $M \twoheadrightarrow_{\overline{\beta}} Q_1$ and $N \twoheadrightarrow_{\overline{\beta}} Q_2$. Since $Q_1, Q_2 \in \mathcal{T}_c$ and $Q_1^{\mathrm{Ch}} \equiv Q_2^{\mathrm{Ch}}$, we can show by induction on Q_1 that $Q_1 \equiv Q_2$. Hence, $M \twoheadrightarrow_{\overline{\beta}} Q_1 \ {}_{\overline{\beta}}\!\!\twoheadleftarrow N$. Hence, $M =_{\overline{\beta}} N$. ⊠

Definition 53. We define $\vdash_{\beta\mathsf{A}}$ to be the typing relation given by the rules of Figures 2 and 3 (that is $\vdash_\beta$) and ($\mathsf{A}\lambda$) given above.
When $\lambda\mathcal{S} = (\mathcal{T}, \beta, \vdash^{\mathcal{S}}_{\beta})$ is the PTS induced by $\mathcal{S}$, we take $\lambda\mathcal{S}_{\mathsf{A}}$ to be the tuple $(\mathcal{T}, \beta, \vdash^{\mathcal{S}}_{\beta\mathsf{A}})$.

Note that the system $\lambda\mathcal{S}_{\mathsf{A}}$ is not a PTS. Note also that there are no postulates which make it possible to deduce $\vdash \mathsf{A} : s$ for any sort s. This means that in $\lambda\mathcal{S}_{\mathsf{A}}$, the only place in which A can occur is in the domain of an abstraction. In particular, in $\lambda\mathcal{S}_{\mathsf{A}}$, we cannot introduce assumptions of the form $x : \mathsf{A}$ into any legal context and we cannot prove a result of the form $\Gamma \vdash M : \mathsf{A}$.

Theorem 54. *In $\mathcal{T}_c$ the following holds:*
1. *If* $\Gamma \vdash^{\mathcal{S}}_{\overline{\beta}} M : B$ *then* $\Gamma^{\mathrm{Ch}} \vdash^{\mathcal{S}}_{\beta\mathsf{A}} M^{\mathrm{Ch}} : B^{\mathrm{Ch}}$.

2. *If* $\Gamma \vdash^{\mathcal{S}}_{\beta\mathsf{A}} M : B$ *then there are* Γ_1, M_1, B_1 *such that* $\Gamma_1^{\mathrm{Ch}} =_\beta \Gamma$, $B_1^{\mathrm{Ch}} =_\beta B$, $\Gamma_1 \vdash^{\mathcal{S}}_{\overline{\beta}} M_1 : B_1$ *and if* $M \equiv M_2^{\mathrm{Ch}}$ *then* $M_1 \equiv M_2$ *else* $M_1^{\mathrm{Ch}} =_\beta M$.
3. *If* $\Gamma^{\mathrm{Ch}} \vdash^{\mathcal{S}}_{\beta\mathsf{A}} M^{\mathrm{Ch}} : B^{\mathrm{Ch}}$ *then* $\Gamma \vdash^{\mathcal{S}}_{\overline{\beta}} M : B$.

Proof. 1. By induction on the derivation $\Gamma \vdash^{\mathcal{S}}_{\overline{\beta}} M : B$.

2. By induction on the derivation $\Gamma \vdash^{\mathcal{S}}_{\beta\mathsf{A}} M : B$.

3. By 2, $\Gamma_1 \vdash^{\mathcal{S}}_{\overline{\beta}} M : B_1$ where $\Gamma_1^{\mathrm{Ch}} =_\beta \Gamma^{\mathrm{Ch}}$ and $B_1^{\mathrm{Ch}} =_\beta B^{\mathrm{Ch}}$. By Lemma 52, $\Gamma_1 =_{\overline{\beta}} \Gamma$ and $B_1 =_{\overline{\beta}} B$. By Lemma 38, $\Gamma \vdash^{\mathcal{S}}_{\overline{\beta}} M : B_1$ (it is easy to show that Γ is $\vdash_{\overline{\beta}}$-legal). Now, by correctness of types we have:

- Either $B^{\mathrm{Ch}} \equiv s$ then $B \equiv s$ and $B_1 =_{\overline{\beta}} s$ hence $B_1 \twoheadrightarrow_{\overline{\beta}} s$ and by reduction preserves types lemma, $\Gamma \vdash^{\mathcal{S}}_{\overline{\beta}} M : s \equiv B$.
- Or $\Gamma^{\mathrm{Ch}} \vdash^{\mathcal{S}}_{\beta\mathsf{A}} B^{\mathrm{Ch}} : s$ and by 2, $\Gamma_2 \vdash^{\mathcal{S}}_{\overline{\beta}} B : C$ where $\Gamma_2^{\mathrm{Ch}} =_\beta \Gamma^{\mathrm{Ch}}$ and $C^{\mathrm{Ch}} =_\beta s^{\mathrm{Ch}}$. From this we can deduce $\Gamma \vdash^{\mathcal{S}}_{\overline{\beta}} B : s$ and by $(\mathrm{conv}_{\overline{\beta}})$, $\Gamma \vdash^{\mathcal{S}}_{\overline{\beta}} M : B$. ⊠

The proof of the theorem does not depend on how the type A is interpreted, but it might be worth considering how that interpretation should be carried out. Since $(\lambda_{x:\mathsf{A}}.M)$ is the way the Curry-style abstraction $(\lambda_x.M)$ is interpreted, this suggests that we want to interpret $(\lambda_{x:B}.M)$ for B not convertible to A, as a restriction of $(\lambda_{x:\mathsf{A}}.M)$. With this idea for an interpretation, we might think of rule $(\mathsf{A}\lambda)$ as making the system, which is based on the Church-style syntax, more like a system based on the Curry-style syntax. This idea for an interpretation suggests that our intended interpretation of A is as a type including all terms in all other types. The system $\lambda\mathcal{S}_{\mathsf{A}}$ as defined above does not include any postulates to formalize this interpretation: all we have in $\lambda\mathcal{S}_{\mathsf{A}}$ is an alternative to adding domain-free abstraction terms to the Church-style syntax. If we wanted to add such a postulate, we might consider adding the following rule:

$$(\mathsf{AI}) \qquad \frac{\Gamma \vdash M : B}{\Gamma \vdash M : \mathsf{A}}$$

It might appear that this significantly strengthens the system. However, unless we add an axiom of the form $\mathsf{A} : s$ for some sort s, it is impossible to prove that a type of the form $\Pi_{x:\mathsf{A}}.B$ has any sort as its type, so a type

of this form cannot be the type of a conclusion of (λ). Hence, it does not appear that adding rule (AI) would have that much effect on the system. In particular, it appears that adding rules (AI) and $(\mathsf{A}\lambda)$ to a system satisfying strong normalization would preserve strong normalization.

Another possibility would be to interpret A as the type ω that is the type of every term (or pseudoterm in a Church-style syntax, but with ω present as a type, every pseudoterm in a Church-style system is a legal term), which is the way this type is used in intersection type systems. Then instead of rule (AI), we would add the rule

$$(\omega\mathrm{I}) \qquad \frac{}{\Gamma \vdash M : \omega}$$

This would automatically give a type to every pseudoterm of the Church-style semantics. Since the Curry-style system with which we are starting is a PTS, whose postulates exclude a rule like $(\omega\mathrm{I})$, this really would strengthen the system. And this rule clearly does not satisfy strong normalization.

Note that adding postulates to interpret A as a type is not necessary to the interpretation of the Curry-style PTS in a Church-style system.

10. Conclusion and future work

In this paper we showed how to interpret in a Curry style system every Church style pure type system without losing any typing information. We gave a number of interpretations together with conservative extensions that preserve consistency and showed how the reverse interpretations can be constructed. The three new interpretations are summarised in Figure 1.

The following remarks are in order:

Remark 55. [Subtyping] If subtyping is present, $\mathsf{Label}A(\lambda_x.M)$, $lA(\lambda_x.M)$ and $l'A(\lambda_x.M)$ are in a sense restrictions of $\lambda_x.M$ to the type A as domain, and $\lambda_x.M$ is a kind of universal function. Since $\mathsf{Label}A(\lambda_x.M)$, $lA(\lambda_x.M)$ and $l'A(\lambda_x.M)$ are typeable under the same conditions that type $\lambda_x.M$ (and vice versa), Curry style identifies functions with their restrictions. This is not surprising since

λ-calculus involves uniform definitions of functions as rules. In Church-style typing, the domain in an abstraction is given explicitly in the abstraction term; it is the type A in $\lambda_{x:A}.M$. However, in Curry-style typing, no such domain is given in $\lambda x.M$. In a Church-style PTS with subtyping, the specification of the domain A in $(\lambda_{x:A}.M)$ can represent a restriction of a function as a distinct term not convertible to $(\lambda_{x:B}.M)$ for a type B which is not convertible to A, but there is no way to use the Curry-style syntax to represent a restriction this way.

Remark 56. [η-reduction] If we could find a way to add η-reduction to a Church-style system with subtyping without losing the Church-Rosser Theorem, then we would have a Church-style system in which functions could not be distinguished from their restrictions; see [9, Remark 13.77]. For suppose B is a subtype of C, and suppose that within a certain context, $M : C \to D$. Then the subtyping relation gives us

$$(\lambda_{u:B}.u) : B \to C$$

so that if $x \notin \mathsf{FV}(M)$,

$$(\lambda_{x:B}.M((\lambda_{u:B}.u)x))$$

represents the restriction of M to domain type B. Now we have by (β_{Ch}),

$$(\lambda_{x:B}.M((\lambda_{u:B}.u)x)) \twoheadrightarrow (\lambda_{x:B}.Mx),$$

so $(\lambda_{x:B}.Mx)$ also represents the restriction of M to domain type B. But if we then apply a contraction by (η_{Ch}), we contract $(\lambda_{x:B}.Mx)$ to M, thus identifying M with its restriction. Thus, to have a system with subtyping in which functions can be distinguished from their restrictions, it is necessary to use a Church-style syntax and use only β-reduction.

This seems to show the importance of the distinction between β-reduction and $\beta\eta$-reduction for type theory.

Remark 57. [multivariate λ-calculus] In his paper [17], Garrel Pottinger introduces a variety of the λ-calculus, which he calls the *multivariate λ-calculus*,

in which a term of the form $\lambda_{x_1x_2\ldots x_n}.M$ is not an abbreviation for repeated abstraction, but is a term which can only become the head of a redex if it is followed by n arguments. Thus in that calculus, $(\lambda_{xyz}.xz(yz))MN$ is not a redex, but $(\lambda_{xyz}.xz(yz))MNP$ is. If the multivariate λ-calculus is used, then Label would be a term which requires three arguments to make a redex.

In the multivariate λ-calculus, the reduction is β-reduction, not $\beta\eta$-reduction. The reason for this is that η-reduction collapses multivariate abstractions to regular abstractions, since for a multivariate abstract

$$(\lambda_{x_1x_2\ldots x_n}.M)$$

and variables $u_1, u_2, \ldots, u_n$ which do not occur bound or free in our term, we have

$$\begin{aligned}
&\lambda_{u_1}.\lambda_{u_2}.\ldots.\lambda_{u_n}.(\lambda_{x_1x_2\ldots x_n}.M)u_1u_2\ldots u_n\\
&\quad\twoheadrightarrow_\beta \quad \lambda_{u_1}.\lambda_{u_2}.\ldots.\lambda_{u_n}.[x_1 := u_1, x_2 := u_2, \ldots, x_n := u_n]M\\
&\quad\twoheadrightarrow_\alpha \quad \lambda_{x_1}.\lambda_{x_2}.\ldots.\lambda_{x_n}.M
\end{aligned}$$

and

$$\begin{aligned}
&\lambda_{u_1}.\lambda u_2.\ldots.\lambda_{u_n}.(\lambda_{x_1x_2\ldots x_n}.M)u_1u_2\ldots u_n\\
&\quad\twoheadrightarrow_\eta \quad \lambda_{u_1}.\lambda_{u_2}.\ldots.\lambda_{u_{n-1}}.(\lambda_{x_1x_2\ldots x_n}.M)u_1u_2\ldots u_{n-1}\\
&\quad\vdots\\
&\quad\twoheadrightarrow_\eta \quad (\lambda_{x_1x_2\ldots x_n}.M).
\end{aligned}$$

This tells us that if η-reduction is introduced into the multivariate λ-calculus in connection with the use with Label in the previous paragraph, then the distinction between functions and their restrictions would be lost, just as it is with η in the Church-style syntax as shown in Remark 55.

10.1. Future Work/Open Questions

The following questions arise naturally from the above interpretations, but are so far unanswered:

1. In the Church-style syntax, we might consider adding η-contractions and also the contraction scheme

$$\lambda_{x:B}.M \twoheadrightarrow \lambda_{x:\mathsf{A}}.M. \tag{1}$$

   This is roughly equivalent to adding Curry-style abstractions to the Church-style syntax and then adding the contraction scheme

$$\lambda_{x:B}.M \twoheadrightarrow \lambda_x.M.$$

   In conjunction with this extended reduction, the typing rule ($\mathsf{A}\lambda$) would preserve the Subject Reduction Lemma. Furthermore, the contraction scheme (1) is the analogue for the Church-style syntax of a valid reduction in the Curry-style syntax under the interpretation of this paper. But would the Church-Rosser property hold for this reduction?
2. If we interpret A as the type ω as suggested above, then the normal form theorem fails. In intersection type systems with this type, it can be proved that any term that has a type in which ω does not occur has a normal form. Would this be true here?
3. Suppose that instead of PTSs we consider the more liberal versions of PTSs of [4]. How are the results of this paper affected?

Acknowledgements

We would like to thank Martin Bunder, Roger Hindley, Garrel Pottinger and the anonymous referees for their helpful comments and suggestions. Seldin was supported by the Natural Sciences and Engineering Research Council of Canada as well as the Scottish Informatics and Computer Science Alliance (SICSA). Kamareddine was supported by the Pacific Institute of Mathematical Sciences (PIMS).

References

[1] Barendregt, H.P. *The Lambda Calculus: its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics 103. North-Holland, 1984.

[2] Barendregt, H.P. Lambda calculi with types. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum (editors), *Handbook of Logic in Computer Science, Volume 2*, pp. 117–309. Oxford University Press, 1992.

[3] Barthe and Sorensen. Domain Free Pure Type Systems. *Journal of Functional Programming 10(5)*, pp. 417-452, Cambridge University Press, 2000.

[4] M. Bunder and W. Dekkers. Pure type systems with more liberal rules. *Journal of Symbolic Logic*, 66:1561–1580, 2001.

[5] Church, A. A formulation of the simple theory of types. *The Journal of Symbolic Logic 5*, pp. 56–68, 1940.

[6] Coquand, T. and Huet, G. The calculus of constructions. *Information and Computation 76*, pp. 95–120, 1988.

[7] H. Curry, J. Hindley, and J. Seldin. *Combinatory Logic*, volume 2. North-Holland Publishing Company, Amsterdam and London, 1972.

[8] Girard, J.-Y. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris 7, 1972.

[9] J. Hindley and J. Seldin. *Lambda-Calculus and Combinators, an Introduction*. Cambridge University Press, 2008.

[10] Howard. W.A. The formulas-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*, pp. 479–490, Academic Press, New York, 1980.

[11] Fairouz Kamareddine, Twan Laan and Rob Nederpelt. Refining the Barendregt Cube using Parameters. Fifth International Symposium on Functional and Logic Programming, FLOPS 2001, Lecture Notes in Computer Science 2024, Pages 3750–389. Springer 2001.

[12] Fairouz Kamareddine, Twan Laan and Rob Nederpelt *A Modern Perspective on Type Theory From its Origins Until Today*. Kluwer Academic Publishers, Applied Logic Series, Volume 29. 357 pages. May 2004.

[13] Marc Bezem, J.W. Klop and Roel de Vrijer, *Term rewriting systems*. Cambridge University Press 2003.

[14] Z. Luo. ECC, an extended calculus of constructions. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science, June 1989, Asilomar, California, U.S.A.*, 1989.

[15] Z. Luo. *An Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, 1990.

[16] G. Pottinger. Ulysses: Logical and computational foundations of the primitive inference engine. Technical Report TR 11-8, ORA Corporation, January 1988.

[17] G. Pottinger. A tour of the multivariate lambda calculus. In J. M. Dunn and A. Gupta, editors, *Truth or Consequences: Essays in Honor of Nuel Belnap*, pages 209–229. Kluwer Academic Publishers, Dordrecht, Boston, and London, 1990.

[18] G. Pottinger and J. P. Seldin. Interpreting Church-style typed λ-calculus in Curry-style type assignment. Former title, "Note on η-Reduction and Labelling Bound Variables in Typed λ-Calculus." Unpublished.

[19] Reynolds, J.C. Towards a theory of type structure. *Lecture Notes in Computer Science. 19*, pp. 408–425, Springer-Verlag, 1974.

[20] J. Seldin. Progress report on generalized functionality. *Annals of Mathematical Logic*, 17:29–59, 1979.

[21] J. Seldin. On the relation between Church-style typing and Curry-style typing. Unpublished.

[22] S. van Bakel, L. Liquori, S. R. della Rocca, and P. Urzyczyn. Comparing cubes of typed and type assignment systems. *Annals of Pure and Applied Logic*, 86(3):267–303, 1997.