Bridging Curry and Church’s typing style

Fairouz Kamareddine\textsuperscript{1}, Jonathan P. Seldin\textsuperscript{2} and J.B. Wells\textsuperscript{1}

\textsuperscript{1} School of Maths and Computer Sc., Heriot-Watt Univ., Edinburgh, UK
\textsuperscript{2} Maths and Computer Sc., Univ. of lethbridge, Canada

Abstract. There are two versions of type assignment in the $\lambda$-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, $\lambda x:A.M$ 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, $\lambda x.M$ 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

1 Introduction

There are two main styles of type theory in $\lambda$-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 \textit{domain-full} and the \textit{domain-free} styles respectively. These styles are compared and discussed in \cite{3}.

Remark 1 Barthe and Sorensen \cite{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 $\lambda$-calculus. In this they are following Barendregt \cite[Definition 4.1.7]{2}, who identifies as the Curry-version of $\lambda 2$ a system in which the rules for $\forall$ are given as follows:
(∀-elimination) \[ \frac{\Gamma \vdash M : (\forall \alpha . \sigma)}{\Gamma \vdash M : [\tau/\alpha]\sigma} \]

(∀-introduction) \[ \frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : (\forall \alpha . \sigma)} \quad \alpha \notin \text{FV}(\Gamma) \]

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 \( \lambda \)-calculus and \( \text{I} \) in a system of combinators can have any type of the form \( \alpha \to \alpha \), which Curry wrote \( \text{F}_\alpha \lambda \). This is what Curry called functionality. He also suggested what he called generalized functionality, in which the constant \( \text{F} \) was replaced by \( \text{G} \), where \( \text{G}_\alpha \beta \) 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 \( \lambda_x . M \). 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 \( (\lambda_x : A . B) \). 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 \(\lambda\)-abstraction is not lost. This interpretation extends previous work with Garrel Pottinger [18],\(^3\) which carried through the interpretation for three systems from the Barendregt cube: \(\lambda \rightarrow [5], \lambda 2 [8, 19], \lambda C [6]\), 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 \((\lambda x:A.M)\). 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 \(\Gamma \vdash (\lambda x:A.M) : (\Pi x:B.C)\). This rule corresponds in a sense to the fact that in the Curry-style system, the term interpreting \((\lambda x:B.M)\) \(\beta\)-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 \(\lambda\)-calculus in both Church’s and Curry’s notations (systems \(T\) and \(T_c\)) and we introduce 3 extra systems (\(T_c\) + Label, \(T_l\) and \(T_l'\)) that will be used to study the interpretations between \(T\) and \(T_c\) 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, typability of subterms and preservation of types. The systems introduced include PTSs [2], DFPTSSs [3] which cannot support faithful translations between Church’s and Curry’s notation, and 3 systems that support faithful translations: L-complete DFPTSSs, IPTSSs and l’PTSs.

– In Section 4 we show that DFPTSSs 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 \(T_c +\) Label. We find that these DFPTSSs 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 IPTSSs (which interpret \(T_l\)) 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 DFPTSSs.

– 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.

\(^3\) But the reader will not need knowledge of [18] to understand the present paper.
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 \( \text{Label} \) so that in the Curry-style syntax

\[
(\lambda x:A.M)
\]

is an abbreviation for

\[
\text{Label}A(\lambda x.M).
\]

(Pottinger used “\( \phi \)” for “\( \text{Label} \).”) By the analogy with \( \beta \)-reduction, we will want

\[
\text{Label}A(\lambda x.M)N \rightarrow M[x := N].
\]

This suggests that \( \text{Label} \) should have the reduction rule

\[
\text{Label}XYZ \rightarrow YZ,
\]

as proposed by Pottinger in [16, §9]. This would suggest, in turn, that we define \( \text{Label} \) as follows:

\[
\text{Label} \equiv \lambda xyz.yz.
\]

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

The next definition introduces a number of sets of terms. \( T \) is the set of terms written à la Church while \( T_c \) is the set of terms written à la Curry. Terms in \( T_l \) and \( T_r \) are also terms written à la Curry, but instead of using \( \text{Label} \) to save the type 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 1 hole which will be used in the proofs.

**Definition 2** [Terms and translations]

1. We define the set of terms \( T \) by:

\[
T ::= S | V | \lambda y:T.T | \Pi y:T.T | TT.
\]

We define the set \( C \) of \( T \)-terms with 1 hole by:

\[
C ::= \boxempty | \lambda y:C.T | \lambda y:T.C | \Pi y:C.T | \Pi y:T.C | CT | TC.
\]

2. We define the set of terms \( T_c \) by:

\[
T_c ::= S | V | \lambda y:T_c.T | \Pi y:T_c.T | T_c.T_c | T_cC_c.
\]

We denote the term \( \lambda x:Ax.y \) of \( T_c \) by \( \text{Label} \).

We define the set \( C_c \) of \( T_c \)-terms with 1 hole by:

\[
C_c ::= \boxempty | \lambda y:C_c.T_c | \Pi y:C_c.T_c | \Pi y:T_c.C_c | C_cT_c | T_cC_c.
\]

3. We define the set of terms \( T_l \) by:

\[
T_l ::= S | V | \lambda y:T_l.T | T_lT_l | T_l.T_l.
\]

We define the set \( C_l \) of \( T_l \)-terms with 1 hole by:

\[
C_l ::= \boxempty | \lambda y:C_l.T_l | \Pi y:T_l.C_l | C_lT_l | T_lC_l | lT_l(\lambda y:C_l).
\]

4. We define the set of terms \( T' \) by:

\[
T' ::= S | V | \lambda y:T'.T' | T'T' | l'T'.
\]

We define the set \( C' \) of \( T' \)-terms with 1 hole by:

\[
C' ::= \boxempty | \lambda y:C'.T' | \Pi y:T'.C' | C'T' | T'C' | l'C'.
\]

5. For \( A \in T \), we define:

- \( \overline{A} \in T_c \) by: \( \overline{s} \equiv s, \overline{x} \equiv x, \overline{AB} \equiv \overline{A}\overline{B}, \overline{\Pi x:A.B} \equiv \Pi x:A.\overline{B}, \overline{\lambda x:A.B} \equiv \lambda x.\overline{B}.
- \( \overline{A} \in T_l \) by: \( \overline{s} \equiv s, \overline{x} \equiv x, \overline{AB} \equiv \overline{A}\overline{B}, \overline{\Pi x:A.B} \equiv \Pi x:A.\overline{B}, \overline{\lambda x:A.B} \equiv \lambda x.\overline{B}.
- \( \overline{\lambda x:A.B} \equiv \text{Label}(\overline{A}^{l}(\overline{x}^{l}.\overline{B}^{l})). \)
5

\[ A \in T \text{ if } s = s, x = x, AB = AB, \Pi_{x:A:B} = \Pi_{x:A:B}, \lambda x.A.B = \lambda x.B. \]
\[ A \in T' \text{ if } s = s, \hat{x} = x, AB = A \hat{B}, \Pi_{x:A:B} = \Pi_{x:A:B}, \lambda x.A.B = \lambda x.B. \]

6. For \( A \in T \), we define \( A^w \in T' \) by:
\[ s^w = s, x^w = x, (A^w)^w = A^w, (\Pi_{x:A:B}^w)^w = \Pi_{x:A^{w}B^{w}}, (\lambda x.B)^w = \lambda x.B^w, \]
\[ (lA(\lambda x.B))^w = lA(\lambda x.B^w). \]

7. For \( C \in C \), we define \( \overline{C}, \overline{C'}, \overline{C} \) and \( \overline{\hat{C}} \) in the obvious way. Similarly, if \( C \in C_t \), we define \( C^w \) in the obvious way.

8. Let \( f \in \{., c, l, \} \). If \( A \) is a \( T_f \)-term with 1 hole and \( a \in T_f \), we define \( C[a] \) to be the term resulting from replacing \( \overline{x} \) by \( a \) in \( C \).

Note again in the translations given above, the \( \overline{\_} \) translation \( \lambda x.B' \) of the Church term term \( \lambda x.A.B \) loses the type \( A \). The \( \overline{-} \) translation keeps the type since \( \lambda x.A.B \equiv \text{Label}A\lambda x.B \). Similarly, the translations \( \overline{\_} \) and \( \overline{\_} \) keep the type of the variable \( x \).

**Notation 3** We let \( s, s', s_1, \) etc. range over the sorts. We take \( V \) to be a set of variables over which, \( x, y, z, x_1, u, v, \) etc. range. We assume that \( S \cap V = \emptyset \).

We take \( A, A_1, A_2, B, a, b, t, M, \) etc. to range over \( T, T_\epsilon, T_i \) and \( T' \). We take \( C, C_1, C' \) etc. to range over \( C, C_\epsilon, C_t \) and \( C' \). We use \( \text{FV}(A) \) to denote the free variables of \( A \), and \( A[x := B] \) to denote the substitution of all the free occurrences of \( x \) in \( A \) by \( B \). In particular, \( \text{FV}(lAB) = \{ \text{FV}(A) \cup \text{FV}(B) \} \) and \( lAC[x := B] \equiv lA[x := B][C[x := B]] \).

We assume familiarity with the notion of compatibility. 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 \( \Gamma \). We define subterms in the usual way. For \( \pi \in \{\lambda, \Pi\} \), we write \( \pi_{x_m:A_m} \ldots \pi_{x_n:A_n}.A \) as \( \pi_{x_1^{m..n}} \cdot A \). We also write \( \lambda x_m \ldots \lambda x_n.A \) as \( \lambda x_1^{m..n}.A \).

The next lemma connects terms and their translations.

**Lemma 4**
1. Let \( A \in T \). We have: \( \text{FV}(\overline{A}) \subseteq \text{FV}(A) = \text{FV}(\overline{A'}) = \text{FV}(A) = \text{FV}(\hat{A}) \).
2. Let \( A \in T_i \). We have: \( \text{FV}(\overline{A}) = \text{FV}(A^w) \).
3. Let \( A, B \in T \). We have: \( \overline{A[x := B]} \equiv \overline{A[x := B]} \), \( \overline{A'(x := B')} \equiv \overline{A'(x := B')} \), \( A[x := B] \equiv A[x := B] \) and \( \overline{A[x := B]} \equiv A[x := B] \).
4. Let \( A, B \in T_i \). We have: \( A[x := B]^w \equiv A^w[x := B^w] \). If \( \overline{A'} \equiv B^w \) or \( \overline{A} \equiv B \) or \( \overline{A} \equiv B \) then \( A \equiv B \).
5. It is not the case that \( \overline{A} \equiv B \).
6. Let \( C \in C \) and \( A \in T \). We have: \( C[A] \equiv C[A], \overline{C[A]} \equiv \overline{C[A]}, \overline{C}[\overline{A}] \equiv \overline{C} \cdot \overline{A'}, \overline{C}[\overline{A}] \equiv \overline{C[A]}, \overline{C}[\overline{A}] \equiv \overline{C[A]}, \) and \( \overline{C[A]} \equiv \overline{C[A]} \).
8. Let $C \in C_1$ and $A \in T_1$. We have: $C[A]^\circ = C^\circ[A^\circ]$.

9. If $\overline{C^L} \equiv \overline{C_2^L}$ or $C^L \equiv C_2^L$ or $\tilde{C} \equiv C_2$ then $C \equiv C_2$.

10. It is not the case that $\overline{C_1} \equiv \overline{C_2}$ implies $C_1 \equiv C_2$.

Proof. All of 1, 2, 3, 4 and 5 are by induction on the structure of $A$. We only do the case $A \equiv \lambda x . C \cdot D$ of $\overline{A^L} \equiv \overline{B^L}$ of 5. Since $A \equiv \lambda x . C \cdot D$ then $\overline{A^L} \equiv \overline{\lambda x . \overline{C^L}(\lambda x . T^L)} \equiv \overline{B^L}$, hence $B \equiv \lambda x . E \cdot F$ where $\overline{E^L} \equiv \overline{C^L}$ and $\overline{F^L} \equiv \overline{D^L}$. By IH, $B \equiv \lambda x . C \cdot D \equiv A$. As for 6., let $A \equiv \lambda x y . x$ and $B \equiv \lambda x z . x$ where $y \neq z$. It is obvious that $\overline{A} \equiv \lambda y . x \equiv \overline{B}$ but $A \neq B$. 7, 8 and 9 are by induction on $C$. For 10, use a similar counterexample to that of 6.

\[ \Box \]

**Definition 5** [Reductions]

- \( \beta \)-reduction \( \rightarrow_{\beta} \) is the compatible closure of \( (\lambda x . A) \cdot B \rightarrow_{\beta} B[x := C] \).
- \( \overline{\beta} \)-reduction \( \rightarrow_{\overline{\beta}} \) is the compatible closure of \( (\lambda x . B) \cdot C \rightarrow_{\overline{\beta}} B[x := C] \).
- \( l \)-reduction \( \rightarrow_l \) is the compatible closure of \( lA(\lambda x . b) \rightarrow_l \lambda x . b \).
- \( l' \)-reduction \( \rightarrow_{l'} \) is the compatible closure of \( l'A(\lambda x . b) \rightarrow_{l'} \lambda x . b \).

We define the union of reduction relations as usual. For example, \( \beta, \cdot l \)-reduction \( \rightarrow_{\beta l} \) is the union of \( \rightarrow_{\beta} \) and \( \rightarrow_l \). We speak of \( \beta, \cdot l \)-reduction and use \( \rightarrow_{\beta} \) to denote \( \rightarrow_{\beta l} \). We also speak of \( \overline{\beta}, \cdot l \)-reduction and use \( \rightarrow_{\overline{\beta}} \) to denote \( \rightarrow_{\overline{\beta} l} \). Note that \( \beta \) is defined on \( T \); \( \overline{\beta} \) is defined on \( T_2 \), \( T_1 \) and \( T' \); \( l \) and \( \beta \) are defined on \( T_1 \), and \( l' \) and \( \beta' \) are defined on \( T' \).

- We write \( A \Rightarrow_{\overline{\beta}} B \) when \( A \equiv C[lE(\lambda x . b)[a] \rightarrow_l C[(\lambda x . b)[a]] \rightarrow_{\overline{\beta}} C[b[x := a]] \equiv B \).
- We write \( A \Rightarrow_{\overline{\beta}} B \) when \( A \equiv C[l'E(\lambda x . b)[a] \rightarrow_l C[(\lambda x . b)[a]] \rightarrow_{\overline{\beta}} C[b[x := a]] \equiv B \).

- Let \( r \in \{ \beta, \overline{\beta}, l, l', \beta, \beta' \} \). We define \( r \)-redexes in the usual way. Moreover:
  - \( \rightarrow_r \) is the reflexive transitive closure of \( \rightarrow_r \) and \( =_r \) is the equivalence closure of \( \rightarrow_r \). We write \( \overline{\rightarrow_r} \) to denote one or more steps of \( r \)-reduction.
  - We write \( \overline{\rightarrow_r} \) (resp. \( \overline{\rightarrow_r} \)) to denote at most (resp. exactly) \( n \) steps of \( r \)-reduction.
  - If \( A \rightarrow_r B \) (resp. \( A \overline{\rightarrow}_r B \)), we also write \( B \rightarrow_r A \) (resp. \( B \overline{\rightarrow}_r A \)).
  - We say that \( A \) is strongly normalising with respect to \( \rightarrow_r \) (we use the notation \( SN_{\rightarrow_r}(A) \)) if there are no infinite \( \rightarrow_r \)-reductions starting at \( A \).
  - We say that \( A \) is in \( r \)-normal form, notation \( NF_{\rightarrow_r}(A) \), if there is no \( B \) such that \( A \rightarrow_r B \).
  - We use \( nf_r(A) \) to refer to the \( r \)-normal form of \( A \) if it exists.
  - We say that \( A \) is \( r \)-weakly normalising, notation \( WN_{\rightarrow_r}(A) \), if \( A \rightarrow_r B \) where \( NF_{\rightarrow_r}(B) \).

**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 \( P_{y . C} \cdot D \) into a Curry expression. Recall that \( \lambda y : C . d^L \equiv LabelA(\lambda y . b) \) 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 \( LabelA(\lambda y . b) \). Similarly, \( l \) and \( l' \) save the type of a variable in a Church expression: \( \lambda y : C \cdot d \equiv lC(\lambda y . d) \) and \( \lambda y : C \cdot d \equiv l'C(\lambda y . d) \).
Remark 7 [LabelA(λy,b) →βλy,b even without η-reduction] In our definition of reduction we have not assumed the η-rule: λx.Ax →η A if x /∈ FV(A). Without this rule, we cannot always show that LabelAB →β B. We can however show this property when B ≡ λy,b as follows: LabelA(λy,b) ≡ (λux.x)A(λy,b) →β (λux.x)y A(λx,y)x →β λx.b[y := x] ≡ λy,b. On the other hand, lA(λx,b) →β λy,b and l′A(λx,b) →β′ λy,b.

The next lemma shows that β-reduction and l-reduction commute.

Lemma 8
1. Let r ∈ {l,l′}. If A →r B then
2. If A →β B1 and A →l B2 then ∃C such that B1 →l C and B2 →β C.
3. If A →β B1 and A →l B2 then ∃C such that B1 →l C and B2 →β C.
4. If A →β B1 and A →l B2 then ∃C such that B1 →l C and B2 →β C.

Proof. 1a): By induction on the structure of A.
1b): By induction on the structure of C.
2: By induction on the structure of A.
4: By induction on A →β B1 using 3.

The next lemma shows that in (Tl,β), l-reductions can be carried out first and β-reductions can be postponed.

Lemma 9
1. If A →β A1 →l A2 then ∃A3 such that A →l A3 →β A2.
2. If A →β A1 →l A2 then ∃A3 such that A →l A3 →β A2.
3. If A →β A1 →l A2 then ∃A3 such that A →l A3 →β A2.
4. If A →β B then there is C such that A →l C →β B.

Proof.
1. By induction on A →β A1. The compatibility cases are easy:
   - BC →β B′C →l B″C use IH.
   - BC →β B′C →l B″C then BC →l BC′ →β B′C′.
   - BC →β BC′ →l BC″ use IH.
   - BC →β BC′ →l BC″ then BC →l B′C →β B′C′.
   - Pi.v.B.C →β Pi.v.B′.C →l Pi.v.B″.C use IH.
   - Pi.v.B.C →β Pi.v.B′.C →l Pi.v.B″.C then Pi.v.B.C →l Pi.v.B′C →β Pi.v.B″.C′.
   - Pi.v.B.C →β Pi.v.B″.C′ →l Pi.v.B″.C′ then Pi.v.B.C →l Pi.v.B″.C →β Pi.v.B″.C′.
   - lBC →β lB′C →l lB″C use IH.
   - lBC →β lB′C →l lB″C then lBC →l lBC′ →β lB″C.
   - lBC →β lB″C →l lB″C use IH.
   - lBC →β lBC′ →l lBC″ then lBC →l lBC →β lBC″.
   - (λv.B)C →β B[v := C] →l D then there are two cases:
• If the l-redex is in \( C \) then \( v \in \text{fv}(B) \). Write \( B \equiv B_{v_1, \ldots, v_n} \) where \( v_1, \ldots, v_n \) are the \( n \) free occurrences of \( v \) in \( B \). Since the l-redex is in \( C \) where \( C \to_{l} C' \) and \( B[v := C] \to_{l} D \), let \( B[v := C] \equiv B_{v_1 := C_1, \ldots, v_n := C_n} \) and suppose \( B[v := C] \to_{l} B_{v_1 := C_1, \ldots, v_n := C_n} \equiv B \). Hence, \( (\lambda \alpha. B) C \to_{l} (\lambda \alpha. B_{v_1 := C_1, \ldots, v_n := C_n}) C \). 

• If the l-redex is in \( B \), say \( B \equiv B_{lEF} \) where \( (\lambda \alpha. B_{lEF}) C \to_{l} B_{lEF}[v := C] \) then \( (\lambda \alpha. B_{lEF}) C \to_{l} (\lambda \alpha. B_{lEF}) C \equiv B_{lEF}[v := C] \).

2. By induction on \( A \to_{l} A_1 \) using 1.
3. By induction on \( A \to_{l} B \) using 2.
4. By induction on \( A \to_{l} B \) using 2.

The next lemma establishes the metasubstitution property for all our systems and shows that \( \equiv_{r} \) is closed under substitution.

**Lemma 10** Let \( (\mathcal{E}, r) \in \{(\mathcal{E}, \beta), (\mathcal{T}, \beta), (\mathcal{T}, \beta'), (\mathcal{T}', \beta')\} \). In \( (\mathcal{E}, r) \) we have:
1. \( A[x := B][y := C] \equiv A[y := C][x := B[y := C]] \).
2. If \( B \equiv_{r} C \) then \( A[x := B] \equiv_{r} A[x := C] \).
3. If \( A \equiv_{r} B \) and \( C \equiv_{r} D \) then \( A[x := C] \equiv_{r} B[x := D] \).

**Proof.** 1 and 2 are by induction on the structure of \( A \). 3 is by induction on \( A \equiv_{r} B \) using 1 and 2.

**Theorem 11** (Church-Rosser (CR) for \( \rightarrow_{\beta/l/l'} \beta \)) Let \( r \in \{\beta, \beta', l, l', \beta \} \). If \( B_1 \equiv_{r} A \rightarrow_{r} B_2 \) then there exists \( C \) such that \( B_1 \rightarrow_{r} C \equiv_{r} B_2 \).

![Fig. 1. Church-Rosser proof of \( \beta \)](image)

**Proof.** For \( \beta \) and \( \beta' \), see [2]. For \( l \) and \( l' \), note that \( (\mathcal{T}, \rightarrow_{l}) \) and \( (\mathcal{T}', \rightarrow_{l'}) \) are orthogonal term rewriting systems (i.e. left linear since no variable occurs twice
on the lefthand side in the $l$ and $l'$ rules, and there are no critical pairs) and hence $l$ and $l'$ are Church-Rosser (see [13]). For $\beta$, use Figure 1.

In order to prove Church-Rosser of $\beta'$, we establish a back translation from $T'$ into $\mathcal{B}_f$. For this, we will need to introduce markings into $\mathcal{T}_f$ so that the translation of $l'A$ which is not part of an $l'$-redex (call it bachelor) into $\mathcal{T}_f$ is a marked translation of $A$. This is to save the information that there was an $l'$ on the left of $A$ which was lost during the translation into $\mathcal{T}_f$ but will be needed when returning back to $T'$.

**Definition 12**

- For $A \in T'$, we define $A^\circ$, a translation of $A$ into marked terms of $\mathcal{T}_f$ as follows:
  
  $s^\circ \equiv s, \ x^\circ \equiv x, (\Pi_{x:A}.B)^\circ \equiv \Pi_{x:A^\circ}.B^\circ, (\lambda_x:B)^\circ \equiv \lambda_x.B^\circ$, $(l'A)^\circ \equiv A^\circ$,
  
  $(l'AB)^\circ \equiv lA^\circ B^\circ$ and $(CB)^\circ \equiv C^\circ B^\circ$ (if $C \neq l'A$).

- If $A$ is a marked term in $\mathcal{T}_f$, we construct $A^\triangleleft \in T'$ by writing every $lAB$ into $l'A^\circ B^\triangleleft$ and also writing every $\overline{A}$ into $l'A$ hence obtaining a term (unmarked) in $T'$. That is:
  
  $\overline{A} \equiv l'A^\triangleleft, \ s^\triangleleft \equiv s, \ x^\triangleleft \equiv x, (\Pi_{x:A}.B)^\triangleleft \equiv \Pi_{x:A^\triangleleft}.B^\triangleleft, (\lambda_x:B)^\triangleleft \equiv \lambda_x.B^\triangleleft,
  
  (l'AB)^\triangleleft \equiv lA^\triangleleft B^\triangleleft$ and $(CB)^\triangleleft \equiv A^\triangleleft B^\triangleleft$.

In marked $\mathcal{T}_f$, we will work as usual. We simply leave the marks untouched. Hence, $\overline{A} \ [x := b] \equiv A[x := b]$, if $A \rightarrow^\beta B$ then $\overline{A} \overline{\rightarrow^\beta B}$, if $\overline{A} \overline{\rightarrow^\beta B}$ then $B$ must be marked (i.e., $B \equiv \overline{\overline{C}}$).

The next lemma is needed to show Church-Rosser of $\beta'$.

**Lemma 13**

1. $A[x := b]^\circ \equiv A^\circ[x := b^\circ]$.
2. For $A \in T'$, $(A)^\triangleleft \equiv A$. For $A \in T$, $(\hat{A})^\circ \equiv A$ and $(A)^\circ \equiv \hat{A}$.
3. If $A \rightarrow^\beta B$ then $A^\circ \rightarrow^\beta B^\circ$.
4. If $A \rightarrow^\beta B$ then $A^\circ \overline{\rightarrow^\beta B^}\circ$.
5. If $A \equiv B$ then $A^\circ \equiv B^\circ$.
6. Let $A \in T'$. If $A^\circ \overline{\rightarrow^\beta D}$ then $D \equiv C^\circ$ and $A \rightarrow^\beta C$.
7. Let $A \in T'$. If $A^\circ \rightarrow^\beta D$ then $D \equiv C^\circ$ and $A \rightarrow^\beta C$.

**Proof.** 1. and 2. By induction on $A$ in the relevant set. 3. By induction on the derivation $A \rightarrow^\beta B$. 4. By induction on the length of the derivation $A \rightarrow^\beta B$.
5. By induction on the length of the derivation $A \equiv B$. 6. By induction on the derivation $A^\circ \rightarrow^\beta D$. We only do the following cases:

- If $(l'A)^\circ \equiv A^\circ \rightarrow^\beta D \equiv \overline{C}$, then $A^\circ \rightarrow^\beta C$ and by IH, $C \equiv E^\circ$ and $A \rightarrow^\beta E$.

  Hence, $l'A \rightarrow^\beta l'E$ and $(l'E)^\circ \equiv E^\circ \equiv \overline{\overline{C}} \equiv D$.

- If $(l'A(\lambda_x.b))^\circ \equiv A^\circ(\lambda_x.b^\circ) \rightarrow^\beta \lambda_x.b^\circ$ then $\lambda_x.b^\circ \equiv (\lambda_x.b)^\circ$ and $l'A(\lambda_x.b) \rightarrow^\beta \lambda_x.b$.

- If $((\lambda_x.A)b)^\circ \equiv (\lambda_x.A^\circ)b^\circ \rightarrow^\beta A^\circ[x := b^\circ]$ then $A^\circ[x := b^\circ] \equiv A[x := b] ^\circ$ and $(\lambda_x.A)b \rightarrow^\beta A[x := b]$. 

7. By induction on the length of the derivation \( A^o \rightarrow \beta D \).

**Theorem 14 (Church-Rosser for \( \rightarrow_{\beta'} \))** If \( B_1 \beta' \leftrightarrow A \rightarrow_{\beta'} B_2 \) then there exists \( C \) such that \( B_1 \rightarrow_{\beta'} C \beta' \leftarrow B_2 \).

*Proof.* If \( B_1 \beta' \leftrightarrow A \rightarrow_{\beta'} B_2 \) then by Lemma 13, \( B_1^o \beta' \leftrightarrow A^o \rightarrow_{\beta'} B_2^o \) and by CR of \( \beta' \) (which is not affected by marking), \( \exists C \) such that \( B_1^o \rightarrow_{\beta'} C \beta' \leftarrow B_2^o \).

By Lemma 13, \( C \equiv D^o \) and \( B_1 \rightarrow_{\beta'} D \beta' \leftarrow B_2 \).

**Corollary 15** For \( r \in \{ \beta, \bar{\beta}, l', \bar{\beta}, \beta' \} \), \( r \)-normal forms are unique.

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

**Lemma 16**

1. If \( A \rightarrow_{\beta} B \) then:
   
   (a) \( \underbrace{A}_{\beta} \rightarrow L B \) and \( \overbrace{A}^{L \beta} \rightarrow L B \).

   (b) \( A \rightarrow L B \) and hence \( \underbrace{A}_{\beta} \rightarrow L B \).

   (c) \( \underbrace{A}_{\beta} \rightarrow L B \) and hence \( \overbrace{A}^{L \beta} \rightarrow L B \).

2. If \( A =_{\beta} B \) then \( \underbrace{A}_{\beta} \rightarrow L B \), \( \overbrace{A}^{L \beta} \rightarrow L B \), \( A =_{\beta} B \) and \( \underbrace{A}_{\beta} \rightarrow L \). Then by Lemma 13.2, \( \underbrace{A}_{\beta} \rightarrow L B \).

3. If \( A =_{\beta} B \) then \( A =_{\beta} B \).

4. If \( A =_{\beta} B \) then \( A =_{\beta} B \).

5. \( \underbrace{A}_{\beta} \rightarrow L B \) iff \( A =_{\beta} B \).

6. If \( A \rightarrow L B \) then \( A = C[(\lambda x:E.F)a] \rightarrow_{\beta} C[F[x := a]] \) and \( B = C[F[x := a]] \).

*Proof.* 1 (a), (b) and (c) are by induction on the derivation \( A \rightarrow_{\beta} B \) using lemma 4 (and remark 7 for (a)).

2. is by Church-Rosser of \( \beta \) and 1 (a), (b) and (c) above.

3. First show by induction on \( A \rightarrow_{\beta} B \) that \( A \rightarrow_{\beta} B \) gives \( A^o =_{\beta} B^o \), then use induction on the length of the derivation \( A \rightarrow_{\beta} B \).

4. is by Church-Rosser of \( \beta \) and 3 above.

5. If \( A =_{\beta} B \) then by 4 above, \( (A)^o =_{\beta'} (B)^o \) and by Lemma 13.2, \( \underbrace{A}_{\beta} =_{\beta'} B \). If \( \underbrace{A}_{\beta} =_{\beta'} B \) then by CR of \( \beta' \), \( \exists C \) such that \( A \rightarrow_{\beta'} C \beta' \leftarrow B \). By Lemma 13, \( (\underbrace{A}_{\beta})^o =_{\beta'} (B)^o \) and hence \( (\underbrace{A}_{\beta})^o =_{\beta'} (B)^o \) and by Lemma 13.2, \( A =_{\beta} B \).

6. is by induction on \( A \).

In order to prove that \( \underbrace{A}_{\beta} =_{\beta'} B \) implies \( A =_{\beta} B \), we will give the following definition and lemma.

**Definition 17**

- Let \( \beta^*_r \) be the least compatible relation on \( T \) closed under \( \langle IA(\lambda x.B)C \rightarrow_{\beta^*_r} B[x := C] \rangle \), and define \( \rightarrow_{\beta^*_r} \) and \( =_{\beta^*_r} \) as usual.

- Let \( \underbrace{A}_{\beta} \in T \) \( \subset T \).
Lemma 18

1. If $A \rightarrow \beta_* B$ then $B \equiv C$ where $A \rightarrow \beta C$. Hence, if $A \rightarrow \beta_* B$ then $A \rightarrow \beta B$.
2. If $A \rightarrow \beta B$ then $A \rightarrow \beta_* B$.
3. $\beta^*$ is CR on $T$.
4. $A \equiv B \iff A \rightarrow \beta_* B$.
5. If $A \rightarrow \beta B$ then $A \rightarrow \beta_* B$.
6. $A \equiv \beta B \iff A \rightarrow \beta_* B$.
7. $A \equiv B$ if $A \equiv B$.
8. $A \equiv \beta B \iff A \equiv \beta B$.
9. It is not the case that $A \equiv B$ implies $A \equiv B$.

Proof. 1. First show by induction on the derivation of $A \rightarrow \beta_* B$ that if $A \rightarrow \beta_* B$ then $B \equiv C$ where $A \rightarrow \beta C$. Then, show the lemma by induction on the length of the derivation $A \rightarrow \beta_* B$.
2. First show by induction on the derivation of $A \rightarrow \beta B$ that if $A \rightarrow \beta B$ then $A \rightarrow \beta_* B$. Then, show the lemma by induction on the length of the derivation $A \rightarrow \beta B$.
3. If $B_1 \beta_* \equiv A \rightarrow \beta_* B_2$ then by 1, $B_1 \equiv C_1$, $B_2 \equiv C_2$, and $C_1 \beta_* \equiv A \rightarrow \beta C_2$.
4. Hence by CR ($\beta$), $\exists C$ such that $C_1 \equiv \beta C \rightarrow \beta C_2$ and by 2, $B_1 \equiv \beta_* C \rightarrow \beta_* B_2$.
5. If $A \equiv B$ then by CR ($\beta$), $\exists C$ such that $A \equiv \beta C \rightarrow \beta B$ and by 2, $A \equiv \beta_* C \rightarrow \beta_* B$.
6. Note that $A \rightarrow \beta B$ is not possible because otherwise, $A \equiv C[\lambda x.B]$, $B \equiv C[\lambda x.D]$ and hence $B \not\in T$.

Iff $A \rightarrow \beta B$ then $A \equiv C[\lambda x.B]$, $B \equiv C[\lambda x.D]$, and $C[\lambda x.D] \rightarrow t C[(\lambda x.D) : = \beta x]$. Hence $A \rightarrow \beta B$.

If $A \rightarrow \beta B$ where $n > 2$ then by Lemma 9, $A \rightarrow t C \rightarrow \beta B$. Let $C^{marked}$ be $C$ where each $\lambda x.F$ is labeled with the term $E$ where $I E(\lambda x.-)$. Along the path was $l$-reduced to $\lambda x.-$. Note that all $\lambda x$ in $C^{marked}$ are labeled because they all come from an $l$-redex from $A \rightarrow t C$. Since $C \rightarrow \beta B$, we can mimic all the $\beta$-steps into $\beta_*$ from $C^{marked}$ to $B$, where every time we hit $(\lambda x.E) a$, we treat it as $(I E(\lambda x.D)) a$ and we use $\beta$-reduction. Since every $\beta$ has to be proceeded somewhere by an $l$-step and since no $l$-steps are done if they don’t also have a following $\beta$-step (otherwise, $B$ would not be a translation), $A \rightarrow \beta B$ is transformed into $A \rightarrow \beta_* B$.

6. If $A \equiv \beta_* B$ then by CR of $\beta_*$, $A \equiv \beta_* C \rightarrow \beta_* B$ and by 1, $A \equiv \beta B$ and hence $A \equiv \beta B$.

If $A \equiv \beta B$ then by CR of $\beta$, there is a $C$ such that $A \equiv \beta C \rightarrow \beta B$. By Lemma 9, $A \rightarrow t A_1 \rightarrow \beta C$ and $B \rightarrow t B_1 \rightarrow \beta C$. Note that every $\beta$-step has somewhere before, an associated $l$-step. Let us collect the $l$-steps from $A \rightarrow t A_1$ that have not been used for a $\beta$-step from $A_1 \rightarrow \beta C$, and let $C_1$ be where these unused steps are undone. Let us do the same with the unused $l$-steps from $B \rightarrow t B_1$ and let us call the corresponding term $C_2$. Note that $C_1, C_2 \in T$ and that one
can show $C_1 = C_2$. Hence, $A \rightarrow_\beta C_1 \equiv C_2 \leftarrow_\beta B$ and $C_1, C_2 \in \Gamma$. By 5 above, $A \rightarrow_\beta C_1 \equiv C_2 \leftarrow_\beta B$. Hence, $A =_\beta B$.


8. Since by Lemma 16.4, $A =_\beta B$ iff $A =_\beta B$ and since by Lemma 17.8 $A =_\beta B$ iff $A =_\beta B$, we are done.

9. let $A = \lambda_x.y.x$ and $B = \lambda_x.z.x$ where $y \neq z$. It is obvious that $A \equiv \lambda_x.y.x = B$, $\lambda_x.x \equiv B$ but $A \neq B$.

Lemma 19  If $A \equiv \overrightarrow{B}$ then $A =_\beta B$.

Proof. Left to the reader.

Corollary 20  $A =_\beta B$ iff $A \equiv \overrightarrow{B}$ iff $A =_\beta B$.

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 IPTSs and the lPTs. For all these systems, we establish the necessary properties.

Definition 21 [Declarations, contexts, $\subseteq$]

1. A declaration $d$ is of the form $x : A$. We define $\text{var}(d) \equiv x$, $\text{type}(d) \equiv A$, and $\text{fv}(d) = \text{fv}(A)$. We let $d, d', d_1, \ldots$ range over declarations.

2. A context $\Gamma$ is a concatenation of declarations $d_1, d_2, \ldots, d_n$ such that if $i \neq j$ then $\text{var}(d_i) \neq \text{var}(d_j)$. We define $\text{dom} (\Gamma) = \{ \text{var}(d) \mid d \in \Gamma \}$ and use $\emptyset$ to denote the empty context. We let $\Gamma, \Delta, \Gamma', \Gamma_1, \ldots$ range over contexts.

3. Assume $\Gamma$ is a context such that $x \not\in \text{dom}(\Gamma)$. We define the substitution of $A$ for $x$ on $\Gamma$, denoted $\Gamma[x := A]$, inductively as follows:

\[
\emptyset[x := A] = \emptyset, \quad (\Gamma', y : B)[x := A] \equiv \Gamma'[x := A], \quad y : B[x := A].
\]

4. We define $\subseteq$ between contexts as the least reflexive transitive relation closed under: $\Gamma, \Delta \subseteq \Gamma, \Delta, \Delta$.

For $r \in \{\beta, \beta, l, \beta, l', \beta'\}$, we extend $r$-reduction to contexts in the usual way.

Similarly, we extend the translations in Definition 2 to contexts as follows:

\[
\emptyset \equiv \emptyset, \quad T, x : A \equiv T, x : A, \quad \emptyset \equiv \emptyset, \quad \Gamma, x : A \equiv \Gamma, x : A.
\]

\[
\emptyset \equiv \emptyset, \quad (\Gamma, x : A) \equiv \Gamma, x : A, \quad \emptyset \equiv \emptyset.
\]

Definition 22 [Type Systems]

- A specification $S$ is a triple $(S, A, R)$ such that $S$ is a set of sorts, $A \subseteq S \times S$ is a set of axioms and $R \subseteq S \times S \times S$ is a set of rules. When no confusion occurs with an axiom, a rule of the form $(s_1, s_2, s_3)$ is written as $(s_1, s_2)$.

- A sort $s$ is said to be a top sort if there is no $(s, s') \in A$. The set of top sorts is denoted by $S_T$. 


• \((s_1, s_2)\) is \(\beta\)-complete if \(\exists s_3, s_4\) such that \((s_1, s_4, s_2), (s_2, s_3, s_3) \in R\).

• \((s_1', s_1, s_2, s_3, s_4, s_5, s_6')\) is L-compatible if: 
\[(s_1, s_2, s_3), (s_3, s_4, s_5), (s_1', s_4, s_5) \in R\] and \((s_1, s_1'), (s_2, s_2') \in A\).

• \((s_1, s_2)\) is L-complete if \(\exists s_1', s_3, s_4, s_5, s_6'\) such that \((s_1', s_1, s_2, s_3, s_4, s_5, s_6')\) is L-compatible.

• A specification \(S\) is said to be functional (also singly-sorted) if:
  1. If \((s, s') \in A\) and \((s, s'') \in A\) then \(s' = s''\).
  2. If \((s_1, s_2, s_3) \in R\) and \((s_1, s_2, s_3') \in R\) then \(s_3 = s_3'\).

A specification \(S\) is said to be injective if:
  1. If \((s', s) \in A\) and \((s'', s) \in A\) then \(s' = s''\).
  2. If \((s_1, s_2, s_3) \in R\) and \((s_1, s_2, s_3) \in R\) then \(s_2 = s_2'\).

Let \(S = (S, A, R)\) be a specification. We define:

• \(\vdash_{\beta}\) to be the type derivation relation given by the rules of Figures 2 and 3.

• \(\vdash_{\gamma}\) to be the type derivation given by the rules of Figures 2 and 4.

• \(\vdash_{\beta'}\) to be the type derivation given by the rules of Figures 2, 4 and 5.

• \(\vdash_{\beta''}\) to be the type derivation given by the rules of Figures 2, 4 and 6.

• When \(\vdash_r\) is a type derivation relation on a specification \(S\), we write \(\vdash^S_r\) to emphasize the dependability of type derivation on \(S\).

The Pure Type System (PTS) induced by \(S\) is the tuple \(\lambda S = (T, \beta, \vdash^S_\beta)\).

The Domain Free Pure Type System (DFPTS) induced by \(S\) is the tuple \(\lambda S = (T, \beta, \vdash^S_\beta)\).

The \(l\)-Labeled Pure Type System (lPTS) induced by \(S\) is the tuple \(\lambda S = (T', \beta', \vdash^{lS}_{\beta'})\).

The \(l'\)-Labeled Pure Type System (l'PTS) induced by \(S\) is the tuple \(\lambda S = (T', \beta', \vdash^{lS}_{\beta'})\).

\[
\begin{array}{ll}
\text{axiom} & \langle \rangle \vdash_r s_1 : s_2 \quad (s_1, s_2) \in A \\
\text{start} & \quad \Gamma \vdash_r A : s \quad x \notin \text{DOM}(\Gamma) \\
\text{weak} & \quad \Gamma \vdash_r A : B \quad \Gamma, x : A \vdash_r x : A \\
\text{II} & \quad \Gamma \vdash_r A : s_1 \quad \Gamma, x : A \vdash_r B : s_2 \quad (s_1, s_2, s_3) \in R \\
\text{conv}_{r} & \quad \Gamma \vdash_r A : B \quad \Gamma \vdash_r B' : s \quad B =_r B' \\
\text{appl} & \quad \Gamma \vdash_r F : \Pi_{s_2, B} \quad \Gamma, a : A \\
\end{array}
\]

\[\Gamma \vdash_r Fa : B[x:=a] \]

Fig. 2. The common \(\vdash_r\) typing rules
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 the strongest system is the Calculus of Constructions $\lambda P \omega$ [6]. The second order $\lambda$-calculus [8, 19] figures on the β-cube between $\lambda \rightarrow$ and $\lambda P \omega$ (cf. Figure 7). 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., $(*:\Box) \in A$). 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 $R$ of triple of sorts $(s_1, s_2, s_3)$ allowed in the type-formation or Π-formation rule, $(\Pi)$.

Fig. 3. The $\lambda$-rule for Church’s typing

\[
\text{(λ)} \quad \frac{\Gamma, x:A \vdash b : B}{\Gamma \vdash \lambda x:A.b : B} \quad \frac{\Gamma \vdash \Pi x:A.B : s}{\Gamma \vdash \Pi x:A.B : s}
\]

Fig. 4. The $\lambda_c$-rule for Curry’s typing

\[
\text{(λc)} \quad \frac{\Gamma, x:A \vdash b : B}{\Gamma \vdash \lambda x:A.b : B} \quad \frac{\Gamma \vdash \Pi x:A.B : s}{\Gamma \vdash \Pi x:A.B : s}
\]

The β-cube is a collection of 8 special systems where each system has its own set $R$ such that $(*,*) \in R \subseteq \{(*,*),(*,\Box),(\Box,*),(\Box,\Box)\}$. With rule $(\Pi)$, 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. $\lambda \Box$ is the weakest system on the β-cube that features this rule.
– $(*,\Box)$ takes care of term dependent types. $\lambda P$ is the weakest system on the β-cube that features this rule.

We refer to each system of Figure 7 according to the kind of PTSs we are in. So, $\lambda P \omega$, (resp. $\lambda \Box P \omega$, resp. $\lambda 2 P \omega$, resp. $\lambda P \omega$) is the PTS (resp. DFPTS, resp. lPTS, resp. l’PTS) calculus of constructions. Now we give basic notions of type systems:

Fig. 5. The $l$-label-rule
2. If \( x \neq z \notin \text{dom}(\Gamma) \) then \( \Gamma \vdash_{\ell} \Pi_{\check{z}}: B \) is \( \ell \)-complete.

\[
\begin{array}{c}
\Gamma \vdash_x A : s_1 & \Gamma \vdash_{\ell} B : s_2 & \frac{z \neq x, z \notin \text{dom}(\Gamma)}{\Gamma \vdash_{\ell} \Pi_{\check{z}}: B} \quad \text{(s, s) \( \ell \)-complete}
\end{array}
\]

Fig. 6. The \( \ell' \)-rule

\[
\begin{array}{c}
\begin{array}{c}
\text{Definition 23 \cite{Legal-Contexts, Judgements, Types, Terms}}
\end{array}
\end{array}
\]

Let \((\mathcal{E}, r) \in \{(\mathcal{T}, \beta), (\mathcal{I}, \beta), (\mathcal{P}, \beta), (\mathcal{I'}, \beta')\}\) and let \( \mathcal{S} \) be a specification on \((\mathcal{E}, r)\).

1. \( \Gamma \vdash_{\ell} A : B \) is a judgement which states that \( A \) has type \( B \) in context \( \Gamma \).

2. \( \Gamma \) is \( \ell \)-legal (or simply legal) if there exist \( A, B \) where \( \Gamma \vdash_{\ell} A : B \).

3. \( A \) is \( \ell \)-legal (or simply legal) if there exist \( B, \Gamma \) where \( \Gamma \vdash_{\ell} A : B \lor \Gamma \vdash_{\ell} B : A \).

4. \( A \) is \( \ell \)-legal (or simply \( \ell \)-legal) if there exists \( B \) where \( \Gamma \vdash_{\ell} A : B \lor \Gamma \vdash_{\ell} B : A \).

5. If \( \Gamma \vdash_{\ell} x : A \) for every \( x : A \in \Delta \), we write \( \Gamma \vdash_{\ell} \Delta \).

6. We define \( \text{Type}_{\ell} = \bigcup_{\mathcal{S}} \text{Type}_{\mathcal{S}}^\ell \) and \( \text{Term}_{\ell} = \bigcup_{\mathcal{S}} \text{Term}_{\mathcal{S}}^\ell \) where:
   - \( \text{Type}_{\mathcal{S}}^\ell = \{ M \in \mathcal{E} \mid \Gamma \vdash_{\ell} M : s \text{ for some } \Gamma \} \),
   - \( \text{Term}_{\mathcal{S}}^\ell = \{ M \in \mathcal{E} \mid \Gamma \vdash_{\ell} M : A : s \text{ for some } \Gamma \text{ and } A \} \).

7. We say that \( \mathcal{S} \) satisfies unicity of types \( \ell \)-w.r.t \( \vdash_{\ell} \) if whenever \( \Gamma \vdash_{\ell} A : B_1 \) and \( \Gamma \vdash_{\ell} A : B_2 \), then \( B_1 =_\ell B_2 \).

8. We say that \( \mathcal{S} \) preserves sorts \( \ell \)-w.r.t \( \vdash_{\ell} \), if whenever \( \Gamma \vdash_{\ell} B_3 : s_1 \), \( \Gamma \vdash_{\ell} B_2 : s_2 \), and \( B_1 =_\ell B_2 \) then \( s_1 = s_2 \).

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 irrespective of any interpretation. 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 24 (Free Variable Lemma for \( \vdash_{\ell} \) and \( \rightarrow_{\ell} \))

1. If \( x : A \) and \( y : B \) are different elements in a legal context \( \Gamma \), then \( x \neq y \).

2. If \( \Gamma_1, x : A, \Gamma_2 \vdash_{\ell} B : C \) 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 \( \Gamma_1 \vdash_{\ell} D : s \) and \( D =_\ell A \) then \( \Gamma_1, x : D, \Gamma_2 \vdash_{\ell} B : C \).

Proof.

1. Since \( \Gamma \) is legal, assume \( \Gamma \vdash_{\ell} C : D \). The proof is by induction on the derivation \( \Gamma \vdash_{\ell} C : D \).
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 $\Gamma_1, x : A \vdash_r D : s$ and $D =_r A$ then by (start) $\Gamma_1, x : D \vdash_r x : D$ and by (weak) $\Gamma_1, x : D \vdash_r x : A$. Hence by (conv) $\Gamma_1, x : D \vdash_r x : A$.

Lemma 25 (Start/Context Lemma for $\vdash_r$ and $\vdash_r$)

If $\Gamma$ is $\vdash_r$-legal then $\Gamma \vdash_r s_1 : s_2$ for every $(s_1, s_2) \in A$ and for all $x : A \in \Gamma$, $\Gamma \vdash_r x : A$ and $\Gamma \vdash_r A : s$.

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

Lemma 26 (Transitivity Lemma)

If $\Delta$ 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 $\Gamma \vdash_r A : B$.

Lemma 27 (Thinning Lemma for $\vdash_r$ and $\vdash_r$)

If $\Delta$ is $\vdash_r$-legal, $\Gamma \subseteq \Delta$, and $\Gamma \vdash_r A : B$ then $\Delta \vdash_r A : B$.

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

Lemma 28 (Substitution Lemma for $\vdash_r$ and $\vdash_r$)

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 $\Gamma, x : A, \Delta \vdash_r B : C$.

Lemma 29 (Generation Lemma for $\vdash_r$ and $\vdash_r$)

1. If $\Gamma \vdash_r s : C$ then $\exists s'$ such that $(s, s') \in A, C =_r s'$ and if $C \neq s'$ 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 $\Gamma \vdash_r \lambda x : A, B : C$ then for some $D, s$, $\Gamma \vdash_r \Pi x : A, D : s$; $\Gamma, x : A \vdash_r B : D$; $\Pi x : A, D =_r C$ and if $\Pi x : A, D \neq C$ then $\Gamma \vdash_r C : s'$ for some sort $s'$.

4. If $\Gamma \vdash_r \lambda A(x : B) : C$ then for some $D, s$, $\Gamma \vdash_r \Pi x : A, D : s$; $\Gamma, x : A \vdash_r B : D$; $\Pi x : A, D =_r C$ and if $\Pi x : A, D \neq C$ then $\Gamma \vdash_r C : s'$ for some sort $s'$.

5. If $\Gamma \vdash_r v \overline{A} : C$ then there is $(s_1, s_2) \vdash_r$-complete, there is $B, z, x$ where $z \neq x, z \notin \text{dom}(\Gamma)$, $\Gamma \vdash_r A : s_1$, $\Gamma \vdash_r \Pi x : A, B : s_2$, $C =_r \Pi z : \Pi x : A, B \cdot \Pi x : A, B$ and if $C \neq \Pi z : \Pi x : A, B \cdot \Pi x : A, B$ then $\Gamma \vdash_r C : s$ for some $s$.

6. If $\Gamma \vdash_r \lambda x : A, B : C$ then for some $A, D, s$, $\Gamma \vdash_r \Pi x : A, D : s$; $\Gamma, x : A \vdash_r B : D$; $\Pi x : A, D =_r C$ and if $\Pi x : A, D \neq C$ then $\Gamma \vdash_r C : s'$ for some sort $s'$.

7. If $\Gamma \vdash_r \Pi x : A, B : C$ then there is $(s_1, s_2, s_3) \in R$ such that $\Gamma \vdash_r A : s_1$, $\Gamma, x : A \vdash_r B : s_2$, $C =_r s_3$ and if $C \neq s_3$ then $\Gamma \vdash_r C : s$ for some sort $s$. 

8. If \( \Gamma \vdash \Gamma \) if \( \Gamma \vdash_F \Pi_{\varepsilon \vdash A} B, \) \( \Gamma \vdash a : A \) and \( C = B[x := a] \) and if \( C \neq B[x := a] \) then \( \Gamma \vdash \Gamma \vdash C : s \) for some \( s. \)

**Proof.** By induction on the derivation \( \Gamma \vdash M : C \) where \( M \) is of the right form.

**Lemma 30 (Correctness of types for \( \vdash \) and \( \rightarrow \))**

If \( \Gamma \vdash A : B \) then \( (B \in S \) or \( \Gamma \vdash B : s \) for some sort \( s. \))

**Proof.** By induction on the derivation \( \Gamma \vdash A : B \) using generation and substitution lemmas for the (appl) case.

**Lemma 31 (Subject Reduction for \( \vdash \) and \( \rightarrow \))**

If \( \Gamma \vdash A : B \) and \( A \rightarrow A' \) then \( \Gamma \vdash A' : B. \)

**Proof.** First we prove by simultaneous induction on the derivation \( \Gamma \vdash A : B \) the following (use correctness of types for the (\( \lambda \)), (l), (l') and (appl) cases and also generation and substitution for the (appl) case):
- If \( \Gamma \vdash A : B \) and \( A \rightarrow A' \) then \( \Gamma \vdash A' : B. \)
- If \( \Gamma \vdash A : B \) and \( \Gamma \rightarrow \Gamma \) then \( \Gamma' \vdash A : B. \)

Then we prove the lemma by induction on the derivation \( A \rightarrow A'. \)

We only show the case \( \Gamma \vdash \Pi_{\varepsilon \vdash A} D[z := a] \) comes from \( \Gamma \vdash \Pi_{\varepsilon \vdash A} D \) and \( \Gamma \vdash a : C \) where \( \varepsilon \vdash a \rightarrow C. \) By generation, \( \Pi_{\varepsilon \vdash A} D = \Pi_{\varepsilon \vdash A} B \) where \( z \neq x. \) Hence \( C \vdash \Pi_{\varepsilon \vdash A} B \) and \( z \notin \text{fv}(D), \) so \( D[z := a] = D[z := a] \). By correctness of types, \( \Gamma \vdash \Pi_{\varepsilon \vdash A} D : s \) for some \( s \) and by generation \( \Gamma \vdash D : s' \) for some \( s'. \) By substitution lemma, \( \Gamma \vdash D[z := a] : s' \) and so \( \Gamma \vdash D : s'. \) Hence by (conv\( \varepsilon \)), \( \Gamma \vdash a : D \) and \( \Gamma \vdash a : D[z := a] \) and we are done.

**Lemma 32 (Reduction preserves types for \( \vdash \) and \( \rightarrow \))**

If \( \Gamma \vdash A : B \) and \( B \rightarrow B' \) then \( \Gamma \vdash A : B'. \)

**Proof.** By correctness of types lemma 30, \((B \in S \) or \( \Gamma \vdash B : s \) for some sort \( s. \))

If \( B \in S \) then \( B' \equiv B \) and we are done. Else, by subject reduction, \( \Gamma \vdash B' : s. \)

Since \( B' = B, \) use (conv\( \rightarrow \)).

**Lemma 33 (Typability of subterms for \( \vdash \) and \( \rightarrow \))**

If \( A \) is \( \vdash, \) legal and \( B \) is a subterm of \( A, \) then \( B \) is \( \vdash, \) legal.

**Proof.** By correctness of types, it is enough to prove that if \( \Gamma \vdash A : B \) and \( A' \) is a subterm of \( A \) then there exists \( \Gamma' \) and \( B' \) such that \( \Gamma' \vdash A' : B'. \) We do this by induction on the generation \( \Gamma \vdash A : B. \) Note that for the cases (\( \lambda \)) and (l) the induction hypothesis is used on a higher branch of the derivation tree. Note also that if \( \Gamma \vdash A : B \) and \( B \) is a top sort then \( B \) is legal but not typable.
4 DFPTs do not faithfully capture Church’s typing

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

**Lemma 34 (DFPTs interpret Church’s typing, but do not preserve types)**

1. It is not the case that $T \vdash \frac{S}{\beta} \ A : B$ implies $\Gamma \vdash \frac{S}{\beta} A : B$.
2. It is not the case that $\Gamma \vdash \frac{S}{\beta} \ A : B$ implies there are $\Gamma_1$ and $\Gamma_2$ such that $T_1 = \frac{\pi}{\beta} \Gamma$, $B_1 = \frac{\pi}{\beta} B$ and $\Gamma_1 \vdash \frac{S}{\beta} A : B_1$.
3. If $\Gamma \vdash \frac{S}{\beta} A : B$ then $T \vdash \frac{S}{\beta} A : B$.

**Proof.** 1. Assume $\exists S$ such that $(s_1, s_1, s_3) \in R$. Note that $\lambda x.x \equiv \lambda x.x$ and $y : s_1, z : s_2 \vdash \frac{S}{\beta} \lambda x.x : \Pi x.y.y$. Even if also $(s_2, s_2, s_4) \in R$, and even if $\exists S$ is L-complete, we still cannot show that $y : s_1, z : s_2 \vdash \frac{S}{\beta} \lambda x.x : \Pi x.y.y$. Otherwise, by generation $\Pi x.y.y = \beta \Pi x.y.y$, and hence by CR, $y = \beta z$ absurd.

2. Take the same example as 1. above. We cannot find $\Gamma_1$ and $\Gamma_2$ such that $T_1 = \frac{\pi}{\beta} y : s_1, z : s_2$, $B_1 = \frac{\pi}{\beta} \Pi x.y.y$ and $\Gamma_1 \vdash \frac{S}{\beta} \lambda x.x : B_1$. Otherwise, since $\Pi x.y.y = \beta \Pi x.y.y$, by Corollary 20, $B_1 = \beta \Pi x.y.y$ and we can find $\Gamma_2$, $s$ such that $\Gamma_2 \vdash \frac{S}{\beta} \Pi x.y.y : s$ and $\Gamma_2 \vdash \frac{S}{\beta} \lambda x.x : B_1$ (if $\Gamma_1 \vdash \frac{S}{\beta} \lambda x.x : B_1$) then by $(\text{conv}_\beta)$, $\Gamma_2 \vdash \frac{S}{\beta} \lambda x.x : \Pi x.y.y$ which is absurd for the same reasons as 1. above.

3. By induction on $\Gamma \vdash \frac{S}{\beta} A : B$, using corollary 20.

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

Recall Remark 6 where we discussed how we intend LabelA($\lambda_x.b$) 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 Remark 7 where we showed that LabelA($\lambda_y.b$) $\rightarrow_{\pi} \lambda_y.b$.

Hence, we would expect LabelA($\lambda_y.b$) to have the same type as $\lambda_y.b$. In order to type LabelA($\lambda_y.b$) in a DFPTS, Label and LabelA must also be typeable. Let us start by discussing when Label is typeable.

**Example 35 (Type checking Label $\equiv \lambda_uxx.xx$ in a DFPTS)** Let us see under what circumstances Label $\equiv \lambda_uxx.xx$ can be typechecked in $T_c$ with $\pi$. We will attempt to solve the equation $\pi \vdash \frac{\pi}{\beta} \lambda_uxx.xx$?

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

1. $\pi \vdash \frac{\pi}{\beta} \lambda_uxx.xx : \Pi x.t_1, \Pi x.t_2, \Pi x.t_3, t_4$ where 2. $\pi \vdash \frac{\pi}{\beta} \Pi x.t_1, \Pi x.t_2, \Pi x.t_3, t_4 : s_5$
2. $\pi \vdash \frac{\pi}{\beta} \lambda_uxx.xx : \Pi x.t_2, \Pi x.t_3, t_4$ where 4. $\pi \vdash \frac{\pi}{\beta} \Pi x.t_2, \Pi x.t_3, t_4 : s_4$
3. $\pi \vdash \frac{\pi}{\beta} \lambda_uxx.xx : \Pi x.t_3, t_4$ where 5. $\pi \vdash \frac{\pi}{\beta} \Pi x.t_3, t_4 : s_3$
4. $\pi \vdash \frac{\pi}{\beta} \lambda_uxx.xx : t_3$ where 6. $\pi \vdash \frac{\pi}{\beta} \Pi x.t_3, t_4 : s_3$
5. $\pi \vdash \frac{\pi}{\beta} \lambda_uxx.xx : t_2$ where 7. $\pi \vdash \frac{\pi}{\beta} \Pi x.t_3, t_4 : s_3$
6. $\pi \vdash \frac{\pi}{\beta} \lambda_uxx.xx : t_1$
7. $\pi \vdash \frac{\pi}{\beta} \lambda_uxx.xx$.

Now we turn to (app) on $\pi$ 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. \( \Gamma, u : t_1, z : t_2 \vdash \Pi_{x : t_3, t_4} : s_3 \) it is enough by weakening to derive \( \Gamma, u : t_1 \vdash \Pi_{x : t_3, t_4} : s_3 \). For this we need \( s_1, s_2 \) such that:

10. \( \Gamma, u : t_1 \vdash \Pi_{x : t_3} : s_1 \). \( \Pi, \Gamma, u : t_1, x : t_3 \vdash \Pi_{x : t_4} : s_2 \) and 12. \((s_1, s_2, s_3) \in R\).

To derive 4. \( \Gamma, u : t_1 \vdash \Pi_{x : t_3, t_4} : s_4 \) we need: 13. \((s_3, s_4, s_5) \in R\).

To derive 2. \( \Gamma \vdash \Pi_{u : t_1}, \Pi_{x : t_3, t_4}, \Pi_{x : t_5} : s_5 \) we need \( s' \) such that:

14. \( \Gamma \vdash t_1 : s'_1 \) and \((s'_1, s_4, s_5) \in R\).

To summarise, for \( \Gamma \vdash \Pi_{u : t_1}, \Pi_{x : t_3, t_4}, \Pi_{x : t_5} : s_5 \) we need \( z \not\in \text{FV}(\Pi_{x : t_3, t_4}) \).

Since \( \Gamma \vdash \Pi_{u : t_1}, \Pi_{x : t_3, t_4}, \Pi_{x : t_5} : s_5 \) we need \( \Pi \vdash t_1 : s'_1 \), \( \Gamma, u : t_1 \vdash \Pi_{x : t_4} : s_2 \) and \( \Pi, \Gamma, u : t_1, x : t_3 \vdash \Pi_{x : t_4} : s_2 \).

Since Label saves the type of the abstracted variable in its first argument, assuming that \( \Gamma \vdash \lambda_x. b : \Pi_{x : A, B} \), we would expect \( \Gamma \vdash \text{Label}(\lambda_x. b) : \Pi_{x : A, B} \).

Let us use Example 35 to see under which circumstances \( \text{Label}A \) is typeable and when is \( \Gamma \vdash \text{Label}(\lambda_x. b) : \Pi_{x : A, B} \) considering that \( \Gamma \vdash \lambda_x. b : \Pi_{x : A, B} \).

**Example 36 (When can we have \( \Gamma \vdash \Pi_{x : A, B} \)?)** In order to have \( \Gamma \vdash \Pi_{x : A, B} \) in a DFPTS, we need to type \( \text{Label}A \) as well as \( \text{Label}A \). Looking back at Example 35, we see that we need to take \( t_3 \) to be \( u \) in the typing of \( \text{Label}A \) and to take \( t_1 \) to be \( s_1 \). We also take \( t_4 \) to be \( v \) and add the condition \( v : s_2 \). Since \( v : s_2 \) will be added to the context, by the DFPTS rules, \( s_2 \) must have a sort \( s'_2 \) as a type. i.e., we must have \((s_2, s'_2) \in A\).

In this way, the way to type \( \text{Label}A \) which behaves as type saver for the variable \( x \) are as follows:

1. \( z, v, u \) and \( x \) are mutually distinct.

2. \((s_1, s_2, s_3), (s_3, s_4, s_5) \in R, (s_1, s'_1), (s_2, s'_2) \in A\).

3. \( v : s_2 \vdash \Pi_{u : s_1}, \Pi_{x : s_3, u : v : \Pi_{x : s_1}} \). Since we also need to type \( \text{Label}A \), we need \( \Gamma \vdash A : s_1 \).

This way by \((\text{app})\), \( \Gamma, v : s_2 \vdash \Pi_{x : A, v : \Pi_{x : A, v}} \). Furthermore, for any \( B \) such that \( \Gamma \vdash \Pi_{x : A, B} : s_2 \), we can use substitution lemma and obtain \( \Gamma \vdash \Pi_{x : A, B} \). Hence if \( \Gamma \vdash \lambda_x. b : \Pi_{x : A, B} \) we get by \((\text{app})\) \( \Gamma \vdash \Pi_{x : A, B} \).

To summarise all these findings, in order to type \( \text{Label}A \equiv \lambda_{u : x, z : x} \) in a DFPTS and to obtain the expected behaviour that: \( \Gamma \vdash \Pi_{x : A, B} \) whenever \( \Gamma \vdash \lambda_x. b : \Pi_{x : A, B} \) we need the following conditions:

- The DFPTS has \((s_1, s_2, s_3), (s_3, s_4, s_5) \in R, (s_1, s'_1), (s_2, s'_2) \in A\).

- \( \Gamma \vdash A : s_1 \).

- \( \Gamma, x : A \vdash \Pi_{x : B} : s_2 \).

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

**Definition 37** [L-compatible and L-complete sorts] We call a tuple of sorts \((s'_1, s_1, s_2, s_3, s_4, s_5, s'_2)\) L-compatible if \((s_1, s_2, s_3), (s_3, s_4, s_5), (s'_1, s_4, s_5) \in R\) and \((s'_1, s'_2) \in A\). We call \((s_1, s_2)\) L-complete if \( \exists s'_1, s_3, s_4, s_5, s'_2\) such that \((s'_1, s_1, s_2, s_3, s_4, s_5, s'_2)\) is L-compatible.
Example 38 Let us see which corresponding systems of the cube have L-compatible tuple \((s'_1, s_1, s_2, s_3, s_4, s_5, s'_2)\). Since \((s_1, s'_1), (s_2, s'_2) \in A\), \(s_1 = \ast, s_2 = \ast, s'_1 = \square\) and \(s'_2 = \square\). We also need \((\ast, s_3), (s_3, s_4), (\square, s_4, s_3) \in R\) and hence \(* = s_3 = s_4 = s_5\). Hence, \((\square, \ast, \ast, \ast, \ast, \ast, \square)\) is the only L-compatible tuple and it holds in \(\overline{\nu_2}, \overline{\nu_2}, \overline{\nu_2}, \overline{\nu_2}\). In these systems, the only L-complete pair of sorts is \((\ast, \ast)\).

If we take a DFPTS where \(S = \{\ast, \square, \triangle\}, A = \{(\ast, \ast), (\square, \triangle)\} \text{ and } R = \{(\ast, \ast, \ast, \ast, \ast, \ast, \ast)\} \text{ and } (\square, \ast, \ast, \ast, \ast, \ast, \square)\) are L-compatible. Hence, \((\ast, \ast)\) and \((\ast, \square)\) are L-complete.

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

Lemma 39 Assume a DFPTS with L-compatible sorts \((s'_1, s_1, s_2, s_3, s_4, s_5, s'_2)\).

For \(z, x, u \text{ and } v \text{ mutually distinct}, v : s_2 \vdash^\neg \text{Label} : \Pi_{u : s_3}, \Pi_{x : u \cdot v}, \Pi_{x : u \cdot v} : s_5\).

Proof. Since \(0 \cdot \vdash^\neg v : s_2 \text{ and } 0' \vdash^\neg s_2, u : s_1, x : u \vdash^\neg \beta v : s_2 \text{ and } 0' \vdash^\neg s_2, u : s_1 \vdash^\neg s_1 : s_1\):

1. \(v : s_2, u : s_1, \Pi_{x : u \cdot v} : s_3 \text{ by } 0', (II) \text{ and } (s_1, s_2, s_3) \in R\).
2. \(v : s_2, u : s_1, z : \Pi_{x : u \cdot v} : s_3 \text{ by } 1, \text{ (weak), for } z \text{ fresh}.
3. \(v : s_2, u : s_1, \Pi_{x : u \cdot v} : s_4 \text{ by } 1, 2, (II) \text{ and } (s_1, s_3, s_4) \in R\).
4. \(v : s_2 \vdash^\neg s_2, s'_1 : s'_1 \text{ by start lemma since } (s_1, s'_1) \in A\).
5. \(v : s_2 \vdash^\neg \Pi_{u : s_3}, \Pi_{x : u \cdot v} : s_5 \text{ by } 3, 4, (II) \text{ and } (s'_1, s_4, s_5) \in R\).
6. \(v : s_2, u : s_1, z : \Pi_{x : u \cdot v} : x \text{ is legal by } 2, \text{ and generation}.
7. \(v : s_2, u : s_1, z : \Pi_{x : u \cdot v} : x \text{ is legal by } 2, \text{ and generation}.
8. \(v : s_2, u : s_1, z : \Pi_{x : u \cdot v} : x \text{ is legal by } 6, \text{ and start lemma}.
9. \(v : s_2, u : s_1, z : \Pi_{x : u \cdot v} : x \text{ is legal by } 6, \text{ and start lemma}.
10. \(v : s_2, u : s_1, \lambda x : \Pi_{x : u \cdot v} : s_5 \text{ by } 9, 2, \text{ and } (\lambda c)\).
11. \(v : s_2, u : s_1, \Pi_{x : u \cdot v} : s_5 \text{ by } 10, 3, \text{ and } (\lambda c)\).
12. \(v : s_2 \vdash^\neg \lambda x : \Pi_{x : u \cdot v} : s_5 \text{ by } 11, 5, \text{ and } (\lambda c)\).

Example 40

- Since \((\square, \ast, \ast, \ast, \ast, \ast, \square)\) is L-compatible in \(\overline{\nu_2}, \overline{\nu_2}, \overline{\nu_2}, \overline{\nu_2}, \overline{\nu_2}, \overline{\nu_2}\), we have in these systems \(v : s_2 \vdash^\neg \text{Label} : \Pi_{u : s_3}, \Pi_{x : u \cdot v} : s_5\).
- If we take the DFPTS where \(S = \{\ast, \square, \triangle\}, A = \{(\ast, \ast), (\square, \triangle)\} \text{ and } R = \{(\ast, \ast, \ast, \ast, \ast, \ast, \ast)\} \text{ and } (\square, \ast, \ast, \ast, \ast, \ast, \ast)\) are L-compatible. We have:

\[ v : \ast \vdash^\neg \text{Label} : \Pi_{u : s_3}, \Pi_{x : u \cdot v} : s_5 \]
\[ v : \square \vdash^\neg \text{Label} : \Pi_{u : s_3}, \Pi_{x : u \cdot v} : s_5 \]

By example 40 we see that Label \(\in \text{Term}^* \cup \text{Term}^\square\) and hence Label is not uniquely sorted. The next lemma and corollary show that in DFPTSs that preserve sorts, if Label is typeable then it is uniquely sorted.

Lemma 41 Assume a DFPTS \(\overline{\nu_2}S\) which preserves sorts such that

\[ v : s_2 \vdash^\neg \Pi_{u : s_3}, \Pi_{x : u \cdot v} : s_5 \text{. There is a unique L-compatible tuple } (s'_1, s_1, s_2, s_3, s_4, s_5, s'_2) \text{.} \]
Proof. By generation, \( \exists s_1', s_4 \) such that \( v : s_2, u : s_1 \vdash \Pi z: \Pi w: v. \Pi x: u. v : s_4 \) and \( v : s_2 \vdash \Pi z: s_4 \). By start lemma, \( (s_1, s_1'), (s_2, s_2') \in A \) for some \( s_2' \). By generation, \( \exists s_3, s_4 \) such that \( v : s_2, u : s_1, z : \Pi x: u. v \vdash \Pi z: \Pi x: u. v : s_3 \) and \( v : s_2, u : s_1 \vdash \Pi z: \Pi x: u. v : s_3 \) where \( (s_3, s_3', s_4) \in R \). By weakening and preservation of sorts, \( s_3 = s_3' \).

Since \( v : s_2, u : s_1 \vdash \Pi z: \Pi x: u. v : s_3 \), by preservation of sorts and generation, \( v : s_2, u : s_1 \vdash \Pi z: u : s_1 \) and \( v : s_2, u : s_1, x : u \vdash \Pi v : s_2 \) and \( (s_1, s_2, s_3) \in R \).

Hence, \( (s_1, s_1, s_2, s_3, s_4, s_5, s_2') \) is L-compatible. Take \( (s_1', s_1, s_2, s_3, s_4, s_5, s_2') \) L-compatible. As \( (s_1, s_1'), (s_2, s_2') \), \( (s_2, s_2') \in A \), then \( s_1 = s_1' \) and \( s_2 = s_2' \) by preservation of sorts and start lemma.

Since \( v : s_2, u : s_1, x : u \vdash \Pi v : s_2 \) and \( v : s_2, u : s_1 \vdash \Pi z: \Pi x: u. v : s_3 \) and hence by preservation of sorts, \( s_3 = s_3' \).

Since \( v : s_2, u : s_1 \vdash \Pi z: \Pi x: u. v : s_3 \), by weakening \( v : s_2, u : s_1, z : \Pi x: u. v \vdash \Pi z: \Pi x: u. v : s_3 \) and by (II) and \( (s_3, s_3, s_4') \) we have \( v : s_2, u : s_1 \vdash \Pi z: \Pi w: v. \Pi x: u. v : s_4' \) and hence by preservation of sorts, \( s_4 = s_4' \).

Corollary 42 Assume a DFPTS \( L \) which preserves sorts. Then we have:
1. \( v : s_2 \vdash \Pi Label : \Pi u: s_1. \Pi z: \Pi w: v. \Pi x: u. v : s_5 \) iff there is a unique L-compatible tuple \( (s_1', s_1, s_2, s_3, s_4, s_5, s_4') \).
2. If \( Label \) is typeable then there is a unique sort \( s \) such that \( Label \in Term^s \).

Proof. 1. is a corollary of Lemmas 39 and 41.
2. If \( Label \) is typeable then by Example 36, for \( z, v, u \) and \( x \) are mutually distinct and for \( (s_1, s_2, s_3), (s_3, s_4, s_5), (s_1', s_1), (s_2, s_2') \in A \) we have \( \vdash v : s_2 \vdash \Pi Label : \Pi u: s_1. \Pi z: \Pi w: v. \Pi x: u. v \). Hence, for some L-compatible tuple \( (s_1', s_1, s_2, s_3, s_4, s_5, s_2') \) \( \vdash \Pi Label : \Pi u: s_1. \Pi z: \Pi w: v. \Pi x: u. v \) and \( s_5 \) is unique and \( Label \in Term^s \).

Starting from \( \vdash \Pi A : s_1 \) and \( \vdash \Pi x: A. B : s_3 \) (needed for correctness of types and hence subject reduction to hold), we need to ensure that for every \( b \) where \( \vdash \Pi A : s_1 \) and \( \vdash \Pi x: A. B : s_2 \), and hence by \( (\lambda x) \vdash \Pi x: A. B : s_3 \), we have \( \vdash \Pi \lambda x: A. B : s_3 \).

Lemma 43 Assume a DFPTS with L-complete \( (s_1, s_2) \).
1. There are mutually distinct \( u, v, x, z \) and there is \( s_5 \) such that:
\( \vdash v : s_2 \vdash \Pi Label : \Pi u: s_1. \Pi z: \Pi w: v. \Pi x: u. v : s_5 \).
2. For any \( \Gamma, A, B \) such that \( \Gamma \vdash \Pi A : s_1 \), \( \Gamma, x : A \vdash \Pi B : s_2 \), there are \( s_3, s_4 \) such that \( \forall \Pi A : s_1 \), \( \Pi x: A. B : s_4 \) and \( \Gamma \vdash \Pi LabelA : \Pi x: A. B : s_4 \).
3. For every \( b \) where \( \vdash \Pi A : s_1 \) and \( \vdash \Pi x: A. B : s_2 \), we have \( \vdash \Pi \lambda x: A. B : s_3 \).

Proof. 1. By definition of L-complete \( (s_1, s_2) \), there is an L-compatible tuple of sorts \( (s_1', s_1, s_2, s_3, s_4, s_5, s_4') \). By Lemma 39, for \( z, x, u \) and \( v \) mutually distinct, \( v : s_2 \vdash \Pi Label : \Pi u: s_1. \Pi z: \Pi w: v. \Pi x: u. v : s_5 \).
2. \( \vdash \Pi A : s_1 \) is by (II) and \( (s_1, s_2, s_3) \in R \).

By weakening lemma and (app), \( \vdash \Pi LabelA : \Pi x: A. B : s_3 \). Hence by substitution lemma, \( \vdash \Pi \lambda x: A. B : s_3 \).
\[ \Gamma \vdash \Pi_{z:A,B} \Pi_{x:A} B : s_4 \text{ is by weakening, } (\Pi) \text{ and } (s_1, s_3, s_4) \in R. \]

3. By \((\lambda x), \Gamma \vdash \lambda x.b : \Pi_{x:A,B}\). By \((\text{app}) \) \(\Gamma \vdash LabelA(\lambda x.b) : \Pi_{x:A} B : s_3.\) ☒

**Corollary 44** In a DFPTS with \(L\)-complete \((s_1, s_2)\), where \(\Gamma \vdash A : s_1 \) and \(\Gamma, x : A \vdash A : s_2\), we have: \(\Gamma \vdash LabelA(\lambda x.b) : \Pi_{x:A} B\) iff \(\Gamma \vdash LabelA(\lambda x.b) : \Pi_{x:A} B\).

Proof. By lemma 43, \(\Gamma \vdash LabelA : \Pi_{z:A,B} \Pi_{x:A} B\). If \(\Gamma \vdash LabelA(\lambda x.b) : \Pi_{x:A} B\) then by \((\text{app})\), \(\Gamma \vdash LabelA(\lambda x.b) : \Pi_{x:A} B\). If \(\Gamma \vdash LabelA(\lambda x.b) : \Pi_{x:A} B\) then by subject reduction \(\Gamma \vdash LabelA : \Pi_{x:A} B\). ☒

**Definition 45** \([L\text{-complete DFPTSs}]\) We say that a DFPTS is \(L\)-complete if every \((s_1, s_2)\) for which there are \(\Gamma, A, B\) such that \(\Gamma \vdash A : s_1 \) and \(\Gamma, x : A \vdash A : s_2\), it holds that \((s_1, s_2)\) is \(L\)-complete.

**Example 46** Recall that in the extended calculus of constructions ECC we have:

\[ S = \{ * \} \cup \{ \sqcap_n \text{ where } n \text{ is a nonnegative integer} \} \]
\[ A = \{ * \} \cup \{ \sqcap_n \text{ where } n \text{ is a nonnegative integer} \} \]
\[ R = \{ \{ *, *, *, *, *, \} \text{ where } 0 \leq n \leq m \} \]
\[ \cup \{ \{ *, \sqcap_n, \sqcap_m \text{ where } n \leq m \} \cup \{ \{ \sqcap_n, \sqcap_m, \sqcap_r \text{ where } 0 \leq n \leq r \text{ and } 0 \leq m \leq r \} \} \]

Let us show that the DFPTS corresponding ECC is \(L\)-complete. Assume \(\Gamma \vdash A : s_1 \) and \(\Gamma, x : A \vdash B : s_2\). We will show that \((s_1, s_2)\) is \(L\)-complete:

- If \(s_1 = s_2 = *\) then \((\sqcap_0, *, *, *, *, \sqcap_0)\) is \(L\)-compatible.
- If \(s_1 = *\) and \(s_2 = \sqcap_i \text{ for } i \geq 0\) then \((\sqcap_0, *, *, *, *, \sqcap_{i+1}, \sqcap_i, \sqcap_{i+2}, \sqcap_{i+3}, \sqcap_{i+1})\) is \(L\)-compatible.
- If \(s_1 = \sqcap_i\) and \(s_2 = * \text{ for } i \geq 0\) then \((\sqcap_{i+1}, \sqcap_i, *, \sqcap_{i+1}, \sqcap_{i+2}, \sqcap_{i+3}, \sqcap_0)\) is \(L\)-compatible.
- If \(s_1 = \sqcap_i\) and \(s_2 = \sqcap_j \text{ for } i, j \geq 0\) then \((\sqcap_{i+1}, \sqcap_i, \sqcap_{i+j}, \sqcap_{i+j+1}, \sqcap_{i+j+2}, \sqcap_{i+1})\) is \(L\)-compatible.

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

**Lemma 47 (Convertible contexts)** If \(\Gamma_1 \vdash \Gamma_2, \Gamma_1 \vdash A : B \) and \(\Gamma_2 \vdash A : B\) is legal then \(\Gamma_2 \vdash \Gamma_1 \vdash A : B\).

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

**Lemma 48 (We can map Curry typing into Church typing)** If \(\exists S\) is \(L\)-complete then: if \(\Gamma \vdash S A : B\) then \(\exists \Gamma_1, A_1, B_1\) such that \(\Gamma \vdash \Gamma_1 \vdash \Gamma, B_1 \vdash B\)

and \(\Gamma_1 \vdash S A_1 : B_1\) and if \(A \equiv A_2 \) then \(A_1 \equiv A_2\) else \(\equiv \Gamma_1 \vdash A_1 \equiv A_2\). ☒

Proof. By induction on the derivation \(\Gamma \vdash S A : B\) using reduction preserves types lemma 32, convertible contexts lemma 47 and Corollary 20. We only do one case:
If $\Gamma \vdash^L_\beta \lambda x. b : \Pi_{x:A} B$ comes from $\Gamma, x:A \vdash^L_\beta b : B$ and $\Gamma \vdash^L_\beta \Pi_{x:A} B : s$ then somewhere on the proof tree we have $\Gamma \vdash^L_\beta A : s_1$, $\Gamma, x:A \vdash^L_\beta B : s_2$ for $(s_1,s_2,s) \in R$. By IH and Lemmas 32 and 47, $\Gamma_1 \vdash^L_\beta A_1 : s_1$ and $\Gamma_1, x : A_1 \vdash^L_\beta B_1 : s_2$ hence $\Gamma_1 \vdash^L_\beta \Pi_{x:A_1} B_1 : s$ where $\Gamma_1 = \Gamma$, $A_1 = \Gamma$ and $B_1 = B$. Also by IH, $\Gamma_2, x : A_2 \vdash^L_\beta b_1 : B_2$ for $\Gamma_2 = \Gamma$, $A_2 = \Gamma$, $B_1 = \Gamma$ and $B_2 = B$ and $b_1 = b$. By Lemma 47 and (conv$\beta$), we have $\Gamma_1, x : A_1 \vdash^L_\beta b_1$ and hence $\Gamma_1 \vdash^L_\beta \lambda x. B_1 : \Pi_{x:A_1} B_1$. Note that $\lambda x. b_1 = \lambda x. b$ and $\Pi_{x:A_1} B_1 = \Pi_{x:A} B$.

The next lemma shows that L-complete DFPTSs preserve types in the labeled interpretation.

**Lemma 49 (L-complete DFPTSs with Label preserve types)** If $\bar{x}S$ is L-complete then:

1. If $\Gamma \vdash^L_\beta A : B$ then $\bar{\Gamma}^L_\beta \vdash^L_\beta \bar{A}^L_\beta : \bar{B}^L_\beta$.

2. If $\Gamma \vdash^L_\beta \bar{A}^L_\beta : B$ then there are $\Gamma_1, B_1$ such that $\Gamma = \bar{\Gamma}_1^L$, $B = \bar{B}_1^L$ and $\Gamma_1 \vdash^L_\beta A : B_1$.

3. If $\bar{T}_1^L \vdash^L_\beta \bar{A}^L_\beta : \bar{B}^L_\beta$ then $\Gamma \vdash^L_\beta \bar{T}_1^L$.

**Proof.**

1. By induction on $\Gamma \vdash^L_\beta A : B$. We only do the ($\lambda$) case. Assume $\Gamma \vdash^L_\beta \lambda x.A : B$. $\Gamma \vdash^L_\beta A$ and $\Gamma \vdash^L_\beta B$. By IH, $\bar{\Gamma}_1^L, x \vdash^L_\beta \bar{A}^L_\beta : \bar{B}^L_\beta$ and $\bar{\Gamma}_1^L \vdash^L_\beta \Pi_{x:A} \bar{B}^L_\beta : s$ and by ($\lambda$), $\bar{\Gamma}_1^L \vdash^L_\beta \lambda x. \bar{b}^L_\beta : \Pi_{x:A} \bar{B}^L_\beta$. By construction, there is $(s_1,s_2,s) \in R$ such that $\bar{\Gamma}_1^L \vdash^L_\beta \bar{A}^L_\beta : s_1$ and $\bar{\Gamma}_1^L, x \vdash^L_\beta \bar{A}^L_\beta : s_2$. Since $\bar{x}S$ is L-complete, $(s_1,s_2)$ is L-complete.

2. This is a corollary of Lemma 48.

3. By correctness of types lemma either $\bar{B}^L_\beta \equiv s$ or $\bar{T}_1^L \vdash^L_\beta \bar{B}^L_\beta : s$.

If $\bar{B}^L_\beta \equiv s$ then $B \equiv s$ and $\bar{\Gamma}_1^L \vdash^L_\beta \bar{A}^L_\beta : s$. By Lemma 48, $\exists \Gamma_1, B_1$ such that $\bar{\Gamma}_1^L = \bar{\Gamma}_1^L, B_1 = \bar{B}_1^L$ and $\Gamma_1 \vdash^L_\beta A : B_1$. Hence, $B_1 = \beta s$ and hence $B_1 \vdash^L_\beta s$ and so by Lemma 32 $\Gamma_1 \vdash^L_\beta A : s$ and by Lemma 47 $\Gamma \vdash^L_\beta A : s$.

If $\bar{T}_1^L \vdash^L_\beta \bar{B}^L_\beta : s$ then by Lemma 48, $\exists \Gamma_1, C$ such that $\bar{\Gamma}_1^L = \bar{\Gamma}_1^L, C = \bar{C}^L_\beta$ and $\Gamma_1 \vdash^L_\beta B : C$. Hence, $C \vdash^L_\beta s$ and by Lemma 32, $\Gamma_1 \vdash^L_\beta B : s$. Furthermore, by Lemma 48, $\exists \Gamma_2, B_1$ such that $\bar{T}_2^L = \bar{\Gamma}_1^L, B_1 = \bar{B}_1^L$ and $\Gamma_2 \vdash^L_\beta A : B_1$. By Lemma 47 and (conv$\beta$), $\Gamma \vdash^L_\beta A : B$. $\Box$
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). These DFPTSs to behave well require top 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 46), ensuring the existence of L-compatible sorts ($s'_1, s_1, s_2, s_3, s_4, s_5, s'_2$) can be quite restrictive. We require numerous conditions to hold and if the original PTS does not have 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 come 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 49 where we had to impose the L-complete condition on the DFPTS.

**Lemma 50**

1. If $\Gamma \vdash^S \beta A : B$ then $\Gamma \vdash^S \beta A : B$.
2. If $\Gamma \vdash^S \beta A : B$ then $\exists \Gamma_1, A_1, B_1$ such that $\Gamma = \beta \Gamma_1$, $B = \beta B_1$ and $\Gamma_1 \vdash^S \beta A_1 : B_1$ and if $A \equiv A_2$ then $A_1 \equiv A_2$ else $A = \beta A_1$.
3. This is a corollary of 2. above.
4. This is similar to the proof given for the corresponding case of Lemma 49.

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 Label $\equiv \lambda x x . z x$ is typeable and a Church term $\lambda x . b \in \mathcal{T}$ is translated as $\lambda x . b^L \equiv \text{Label}(\lambda x . b^L) \in \mathcal{T}_c$. 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 IPTSs, where new $l'$-terms are added to $T_c$. 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 not 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 IPTS is not restricted over the original PTS.

In this section we will check the mid-way where we add $l'$-terms to $T_c$ and again use $lA(\lambda_x.b)$ 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 $A \in \text{Type}^+$: 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 $(s_1, s_4, s_2, s_2, s_3) \in R$. 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 51** [\(l'$-complete \)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 $\Gamma, A, B$ 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 $\Gamma$ is $l'$-bachelor-free if for any $y : A$ in $\Gamma$, $A$ is $l'$-bachelor-free.

**Lemma 52** ($l'$-complete \(l'$PTSs preserve types) If $\lambda S$ is $l'$-complete then:
1. If $\Gamma \vdash_{\beta'}^S A : B$ then $\Gamma \vdash_{\beta'}^S \tilde{A} : \tilde{B}$.
2. If $\Gamma \vdash_{\beta'}^S A : B$ where $\Gamma, A, B$ are all $l'$-bachelor-free, then there are $\Gamma_1, A_1, B_1$ such that $\Gamma =_{\beta'} \Gamma_1, B =_{\beta'} \tilde{B}_1$ and $\Gamma_1 \vdash_{\beta}^S A_1 : B_1$ and if $A \equiv A_2$ then $A_1 \equiv A_2$ else $A =_{\beta'} \tilde{A}_1$.
3. If $\Gamma \vdash_{\beta'}^S A : B$ where $\Gamma, B$ are all $l'$-bachelor-free, then there are $\Gamma_1, B_1$ such that $\Gamma =_{\beta'} \tilde{\Gamma}_1, B =_{\beta'} \tilde{B}_1$ and $\Gamma_1 \vdash_{\beta}^S A : B_1$.
4. If $\Gamma \vdash_{\beta'}^S \tilde{A} : \tilde{B}$ then $\Gamma \vdash_{\beta}^S A : B$.

**Proof.** 1. By induction on the derivation $\Gamma \vdash_{\beta'}^S A : B$.
2. By induction on the derivation $\Gamma \vdash_{\beta'}^S A : B$. We only do the case: $\Gamma \vdash_{\beta'}^S \tilde{A}(\lambda_x.b) : D[z := \lambda_x.b]$ comes from $\Gamma \vdash_{\beta'}^S \tilde{A}' : \Pi_x : C.D$ and $\Gamma \vdash_{\beta'}^S \lambda_x : C$. Then by generation, we have on the proof tree that for some $B, s_1, s_2, s, E, F$, $\Gamma \vdash_{\beta'}^S A : s_1, \Gamma \vdash_{\beta'}^S \Pi_{x:A}.B : s_2, z \neq x, z \notin \text{DOM}(\Gamma)$, $\Gamma \vdash_{\beta'}^S \Pi_{x:E}.F : s$, $\Gamma, x : E \vdash_{\beta'}^S b : F, \Pi_{x:C}.D \equiv_{\beta'} \Pi_{x : \Pi_{x:A}.B}.\Pi_{x:A}.B$ (hence $C =_{\beta'} \Pi_{x:A}.B$) and $C =_{\beta'} \Pi_{x:E}.F$. Hence $A =_{\beta'} E$ and $B =_{\beta'} F$. By IH and Lemma 32, $\Gamma_1 \vdash_{\beta}^S A_1 : s_1, \Gamma_2 \vdash_{\beta}^S \Pi_{x:A}.B_2 : s_2, \Gamma_3, x : E_1 \vdash_{\beta}^S b_1 : F_1$, where $A_1 =_{\beta'} \tilde{A}_2 =_{\beta'} \tilde{E}_1 =_{\beta'} A$ and $b_1 =_{\beta'} b$ and $\tilde{E}_1 =_{\beta'} \tilde{B}_2 =_{\beta'} B$. It is easy to show that $\Gamma_2, x : A_2 \vdash_{\beta}^S b_1 : B_2$, and so, $\Gamma_2 \vdash_{\beta}^S \lambda_x : A_2, b_1 : \Pi_{x:A_2}.B_2$ and we’re done.
3. This is a corollary of 2. above. Note that \( \hat{A} \) is \( \lambda \)-bachelor-free.
4. Similar to the proof given for the corresponding case of Lemma 49. Note that \( \Gamma, \hat{A} \) and \( \hat{B} \) are all \( \lambda \)-bachelor-free. \( \Box \)

8 Unicity of types, Classification and Consistency

In the previous section 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 53 (Unicity of Types for \( \vdash_{\beta} \) and its failure for all other \( \vdash_r \))**

1. Let \( S \) be a functional specification. \( S \) satisfies the unicity of types with respect to \( \vdash_{\beta} \), but fails to do so w.r.t. any other \( \vdash_r \).
2. If \( S \) 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 \( A_1 \equiv A_2 \), then \( B_1 \equiv B_2 \).
3. If \( S \) satisfies the unicity of types with respect to \( \vdash_r \) then \( S \) preserves sorts w.r.t. \( \vdash_r \).
4. If 2. holds for \( \vdash_r \) then:
   - If \( \Gamma \vdash_r B_1 : s, B_1 \equiv B_2 \) 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 \((s_1, s_1, s_1) \in R \) and \((s_2, s_2, s_4) \in R \). Then:
   - Since \( \Gamma \vdash_r \Pi_{x,y} : s_3 \) and \( \Gamma, x : y \vdash_r y : s_1 \), we get \( \Gamma \vdash_r \lambda_{x,x} : \Pi_{x,y,y} \).
   - Since \( \Gamma \vdash_r \Pi_{x,z} : s_4 \) and \( \Gamma, x : z \vdash_r z : s_2 \), we get \( \Gamma \vdash_r \lambda_{x,x} : \Pi_{x,z,z} \).
   But \( \Pi_{x,y,y} \neq \Pi_{x,z,z} \), 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 \( \Gamma \vdash_r B_2 \) and by Subject reduction of \( \vdash_r \), \( \Gamma \vdash_r B_2 : s \).
   - If \( \Gamma \vdash_r B_2 : s' \) then by 2 above, \( s \equiv s' \) and hence \( s \equiv s' \). \( \Box \)

In what follows, we use sorted variables. We divide \( V \) into countably infinite disjoint subsets \( V^\alpha \) and use \( x^\alpha, y^\alpha, \) etc., to range over \( V^\alpha \). With this partitioning of \( V \), variable renaming will respect sorts. That is for example, \( \lambda_{x,}A[x := y] \) only if \( x \) and \( y \) belong to the same \( V^\alpha \). We replace the two rules (start) and (weak) of Figure 2 by the rules in Figure 8. With these new rules of Figure 8, the second clause of the generation lemma changes to accommodate these sorts as shown in lemma 54.

**Lemma 54 (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 8, then
If \( \Gamma \vdash r \quad x : C \) then for some \( s, A, x \equiv x^s, x : A \in \Gamma, C =_r A \), and \( \Gamma \vdash r \quad C : s \).

If \( \Gamma \vdash \beta \quad l'A : C \) then there is \((s_1, s_4, s_2), (s_2, s_2, s_3) \in R\), there is \( B, z \equiv z^{s_2}, x \) where \( z \not= x, z \not\in \text{DOM}(\Gamma) \), \( \Gamma \vdash r \quad A : s_1, \Gamma, x : A \vdash_r B : s_4, C =_r \Pi_{z;\Pi_{z,A}B,\Pi_{x,A}B} \) and if \( C \not= \Pi_{z;\Pi_{z,A}B,\Pi_{x,A}B} \) then \( \Gamma \vdash \beta' \quad C : s' \) for some \( s' \).

Proof. By induction on the derivation \( \Gamma \vdash r \quad x : C \) \( \Box \)

\[
\begin{array}{c}
\text{(start')} \quad \Gamma \vdash r \quad A : s \quad x^s \not\in \text{DOM}(\Gamma) \\
\hline
\Gamma, x^s : A \vdash r \quad x : A
\end{array}
\]

\[
\begin{array}{c}
\text{(weak')} \quad \Gamma \vdash r \quad A : B \\
\hline
\Gamma, x^s : C \vdash r \quad B
\end{array}
\]

\[
\begin{array}{c}
\text{(I')} \quad \Gamma \vdash r \quad A : s_1 \\
\hline
\Gamma, x : A \vdash r \quad B : s_4 \\
\hline
\Gamma, x^s \not\in \text{DOM}(\Gamma) \\
\hline
\Gamma, \Gamma' : \Pi_{x,y:B,\Pi_{x,A}B} \\
\hline
\Gamma \vdash r \quad \Gamma' A : \Pi_{z;\Pi_{z,A}B,\Pi_{x,A}B}
\end{array}
\]

Fig. 8. The start, weak and \( l'' \) rules with sorted variables

With the introduction of sorted variables, unicity of types still fails but the counterexample given in Lemma 53 needs to change to accommodate the new rules of Figure 8. This change dictates that \( s_1 = s_2 \). Hence, if \((s_1, s_4) \in A, (s_1, s_1, s_3) \in R, \Gamma = y : s_1, z : s_1 \) and \( x = x^{s_1} \), we can derive the following:

\( \Gamma, x : y \vdash r \quad x : y \) and \( \Gamma, x : z \vdash r \quad x : z \) by (start')

\( \Gamma \vdash r \quad \Pi_{x,y:B} : s_3 \) and \( \Gamma \vdash r \quad \Pi_{x,z:z} : s_3 \) by (I)

\( \Gamma \vdash r \quad \lambda_x : \Pi_{x,y:B} \) and \( \Gamma \vdash r \quad \lambda_x : \Pi_{x,z:z} \) by \((\lambda_c)\).

Since \( \Pi_{x,y:B} \not=_{r} \Pi_{x,z:z} \), unicity of types fails.

The counterexample given in lemma 53 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 53 belongs to \( \text{Term}^{s_3} \cap \text{Term}^{s_4} \) 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 58. 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 55** Let \( S \) be a specification. The following hold:
1. If \( s' \in \text{Type}_r^{s_1} \) then \((s', s) \in A\).
2. If \( x \in \text{Type}_r^{s_1} \) then \( x \in \mathcal{V}^s \) where \((s, s') \in A\).
3. \( C \not\in \text{Type}_r^{s_1} \) for \( C \in \{\lambda_x : A, B, \lambda_x : B, lA(\lambda_x : B), l'A\} \).
4. If \( \Pi_{x:A}B \in \text{Type}_r^{s_1} \) then for some \((s_1, s_2, s) \in R, A \in \text{Type}_r^{s_1} \) and \( B \in \text{Type}_r^{s_2} \).
5. If \( F a \in \text{Type}_r^{s_1} \) then for some \((s_1, s_2, s_3) \in R,(s, s_2) \in A, F \in \text{Term}_r^{s_2} \) and \( a \in \text{Term}_r^{s_1} \). 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 54 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 29, $s =_r \Pi x : A.B$, contradicting Church-Rosser.
4. If $\Gamma \vdash_r \Pi x : A.B : s$ then by generation lemma 54, for some $(s_1, s_2, s) \in \mathbb{R}$, $A \in \text{Type}^{s_1}_r$ and $B \in \text{Type}^{s_2}_r$.
5. If $\Gamma \vdash_r F a : s$ then by generation lemma 54, 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 $F \in \text{Term}^{s_3}_r$. By generation (note that $=_{=}$ on sorts is $\equiv$), for some $(s_1, s_2, s_3) \in \mathbb{R}$, $\Gamma \vdash_r H : s_1$ (hence $a \in \text{Term}^{s_1}_r$), $\Gamma, z : H \vdash G : s_2$ and by substitution lemma $\Gamma \vdash_r G[z := a] : s_2$. By subject reduction, since $G[z:=a] \rightarrow_r s$, $\Gamma \vdash_r s : s_2$. By 1. above, $(s, s_2) \in A$. If $F \equiv l'A$ then by generation, there is $B$, $x \neq z$ (note that $z \notin \text{Dom}(\Gamma)$), such that $\Pi z : H.G =_{=} \Pi z : lB.A.B.\Pi x : A.B$. Hence $G =_{=} \Pi x : A.B$ and $G[z:=a] =_{=} \Pi x : A[z:=a].B[z:=a] =_r s$, absurd by Church-Rosser.

**Lemma 56** Let $S$ preserve sorts w.r.t $\vdash_r$. The following hold:
1. If $s' \in \text{Term}^{s}_r$ then for some $s'', (s', s'') \in A$ and $(s'', s) \in A$.
2. If $x \in \text{Term}^{s}_r$ then $x =_{=} x^*$.
3. If $M \in \text{Term}^{s}_r$ where $M \in \{\lambda x : A.B, lA(\lambda x.B)\}$ then $x \in \mathcal{V}^{s_1}$, $A \in \text{Type}^{s_1}_r$, and $B \in \text{Term}^{s_2}_r$ for some $(s_1, s_2, s) \in \mathbb{R}$.
4. If $\lambda x.B \in \text{Term}^{s}_r$ then $x \in \mathcal{V}^{s_1}$, and $B \in \text{Term}^{s_2}_r$ for some $(s_1, s_2, s) \in \mathbb{R}$.
5. If $l'A \in \text{Term}^{s}_r$ then for some $(s_1, s_4, s_2) \in \mathbb{R}$, and $(s_2, s_3, s) \in \mathbb{R}$, $A \in \text{Type}^{s_1}_r$.
6. If $\Pi x : A.B \in \text{Term}^{s}_r$ then for some $(s_1, s_2, s_3) \in \mathbb{R}$, $(s_3, s) \in A$, $x \in \mathcal{V}^{s_1}$, $A \in \text{Type}^{s_2}_r$ and $B \in \text{Type}^{s_3}_r$.
7. If $Fa \in \text{Term}^{s}_r$ then for some $(s_1, s_3, s) \in \mathbb{R}$, $F \in \text{Term}^{s_3}_r$ and $a \in \text{Term}^{s_1}_r$.

*Proof.*
1. By the generation lemma, Church-Rosser and Subject reduction and Lemma 55.1.
2. By the revised clause of generation lemma 54 and preservation of sorts. Note that the $\mathcal{V}^{s_1}$s partition $\mathcal{V}$.
3. For $A \in \text{Type}^{s_1}_r$, and $B \in \text{Term}^{s_2}_r$ for some $(s_1, s_2, s) \in \mathbb{R}$, 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 $(s_1, s_2, s_3) \in \mathbb{R}$, $\Gamma \vdash_r A : s_1$ and $\Gamma, x : A \vdash_r B : s_2$, $C =_{=} s_3$. Hence $A \in \text{Type}^{s_1}_r$ and $B \in \text{Type}^{s_2}_r$. By Church-Rosser and Subject reduction, $\Gamma \vdash_r s_3 : s$ and by lemma 55, $(s_3, s) \in A$. 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.

The following example shows that in general, the classification lemma fails for terms which contain $l'$. 

}\end{quote}
Example 57 Let $s_2 \neq s'_2$, $s_3 \neq s'_3$ and $s_4 \neq s'_4$ and let $(s_1, s_4, s_2)$, $(s_2, s_1, s_3)$, $(s_1, s'_4, s'_2)$ and $(s'_2, s'_3, s'_1) \in R$.

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, 
\begin{align*}
\Gamma \vdash \Pi_{x:y_1,y_2} : s_2, \Gamma \vdash \Pi_{x:y_2} : s'_2, \Gamma \vdash l'y : \Pi_{x:y_1,y_2} : s_3 \text{ and } \\
\Gamma \vdash l'y : \Pi_{x:y_2} : s'_3. \text{ Hence, } l'y \in \text{Term}^s \cap \text{Term}^{s'} \text{ with } s_3 \neq s'_3.
\end{align*}

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 58.

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 58 (Classification) Let $S$ be an injective specification which preserves sorts. Assume $M$ is l'-bachelor-free. The following hold:

1. If $M \in \text{Term}^s \cap \text{Term}^{s'}$ then $s = s'$.
2. If $M \in \text{Type}^s \cap \text{Type}^{s'}$ then $s = s'$.

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

- If $s_1 \in \text{Type}^s \cap \text{Type}^{s'}$ then by Lemma 55, $(s_1, s), (s_1, s') \in A$. By start lemma and preservation of sorts, $s = s'$.

- If $s_1 \in \text{Term}^s \cap \text{Term}^{s'}$ then by Lemma 56, there are $s_2, s_3$ such that 

$(s_1, s_2), (s_2, s), (s_1, s_3), (s_3, s') \in A$. By start lemma and preservation of sorts, 

$s_2 = s_3$ and $s = s'$.

- If $x \in \text{Type}^s \cap \text{Type}^{s'}$ then by Lemma 55, $x \in \mathcal{V}^{s_1} \cap \mathcal{V}^{s_2}, (s, s_1), (s, s_2) \in A$.

Since the $\mathcal{V}^{s_1}$ partition $\mathcal{V}$, $s_1 = s_2$ and by injectivity, $s = s'$.

- If $x \in \text{Term}^s \cap \text{Term}^{s'}$ then by Lemma 56, $x \in \mathcal{V}^s \cap \mathcal{V}^{s'}$. Since the $\mathcal{V}^{s_1}$ partition $\mathcal{V}$, $s = s'$.

- If $\Pi_{x:A} : B \in \text{Type}^s \cap \text{Type}^{s'}$ then by Lemma 55, $(s_1, s_2, s), (s'_1, s'_2, s') \in R$, $A \in \text{Type}^s \cap \text{Type}^{s'}$ and $B \in \text{Type}^s \cap \text{Type}^{s'}$. By IH, $s_1 = s'_1$ and $s_2 = s'_2$.

Hence, $(s_1, s_2, s), (s_1, s_2, s') \in R$. Since $\Gamma \vdash \Pi_{x:A} : B : s$ for some $\Gamma$, by generation, $\Gamma \vdash A : s_3, \Gamma x : A \vdash B : s_4$, hence $s_3 = s_1, s_4 = s_2$ and by (II), $\Gamma \vdash \Pi_{x:A} : B : s$ and $\Gamma \vdash \Pi_{x:A} : B : s'$. By preservation of sorts, $s = s'$.

- If $\Pi_{x:A} : B \in \text{Term}^s \cap \text{Term}^{s'}$ then by Lemma 56, $(s_1, s_2, s_3), (s'_1, s'_2, s'_3) \in R$, $(s_3, s), (s'_3, s') \in A$, $x \in \mathcal{V}^s \cap \mathcal{V}^{s'}$, $A \in \text{Type}^s \cap \text{Type}^{s'}$ and $B \in \text{Type}^s \cap \text{Type}^{s'}$. By IH, $s_1 = s'_1$ and $s_2 = s'_2$.

Hence, $(s_1, s_2, s_3), (s_1, s_2, s'_3) \in R$. Since $\Gamma \vdash \Pi_{x:A} : C : s$ for some $\Gamma$, by generation, $\Gamma \vdash A : s_5, \Gamma x : A \vdash B : s_4$, hence $s_5 = s_1, s_4 = s_2$ and by (II), $\Gamma \vdash \Pi_{x:A} : s_3$ and $\Gamma \vdash \Pi_{x:A} : s'_3$. By preservation of sorts, $s = s'_3$. By start lemma $\Gamma \vdash s : s$ and $\Gamma \vdash s : s'$. By preservation of sorts, $s = s'$.

- If $Fa \in \text{Type}^s \cap \text{Type}^{s'}$ then by Lemma 55, $F$ is not of the form $l'A$, and $(s_1, s_2, s_3), (s'_1, s'_2, s'_3) \in R$, $(s, s_2), (s', s'_2) \in A$, $F \in \text{Term}^s \cap \text{Term}^{s'}$ and $a \in \text{Term}^s \cap \text{Term}^{s'}$. Use IH and injectivity twice.

- If $Fa \in \text{Term}^s \cap \text{Term}^{s'}$ then by Lemma 56, $(s_1, s_2, s_3), (s'_1, s'_2, s'_3) \in R$, $F \in \text{Term}^s \cap \text{Term}^{s'}$ and $a \in \text{Term}^s \cap \text{Term}^{s'}$. Hence by IH, $s = s'$. 

If $F \neq l'A$ 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 55, $M \notin \text{Type}^s \cap \text{Type}^t$.
- If $M \in \{\lambda x.B, \lambda x:A.B, lA(\lambda x.B)\}$ where $M \in \text{Term}^s \cap \text{Term}^t$ then by Lemma 56, $x \in \mathcal{V}^s \cap \mathcal{V}^t$, $A \in \text{Type}^s \cap \text{Type}^t$, and $B \in \text{Term}^s \cap \text{Term}^t$ for some $(s_1, s_2, s), (s'_1, s'_2, s') \in \mathcal{R}$. Use IH, generation lemma and preservation of sorts.

**Corollary 59** Let $\mathcal{S}$ be an injective specification which preserves sorts. Assume $M$ is $l'$-bachelor-free and $\vdash_{-l}$-legal. The following hold:

1. $M \in \text{Term}^s$ for some $s \in \mathcal{S}$; or
2. $M \in \text{Type}^s$ for some $s \in \mathcal{S}_T$; or
3. $M \equiv s$ for some $s \in \mathcal{S}_T$.

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

**Proof.** The same proof as that of Corollary 25 of [3] applies here. $\Box$

Next we move to consistency and show that if $\lambda \mathcal{S}$ is topsort grounded and consistent then $\lambda \mathcal{S}$ (on $\mathcal{T} + \text{Label}$), $\Delta \mathcal{S}$ and $\lambda \mathcal{S}$ are consistent.

**Lemma 60** Let $\lambda \mathcal{S}$ be topsort grounded (i.e., every topsort $s$ is inhabited by another sort $s'$ where $\vdash \lambda \mathcal{S} \; s' : s$).

1. If $\lambda \mathcal{S}$ is $L$-complete and consistent then $\lambda \mathcal{S}$ (on $\mathcal{T} + \text{Label}$) is consistent.
2. If $\lambda \mathcal{S}$ is consistent then $\Delta \mathcal{S}$ is consistent.
3. If $\lambda \mathcal{S}$ is $l'$-complete and consistent then $\lambda \mathcal{S}$ is consistent.

**Proof.** All items are similar. We only do 2. Assume $\Delta \mathcal{S}$ is inconsistent. Hence, every type is inhabited. In particular, for every $s \in \mathcal{S}$, $\Pi_{x:s}.x$ is inhabited. I.e., there is an $M \in \mathcal{T}$ such that $\vdash \lambda \mathcal{S} \; M : \Pi_{x:s}.x$. Hence by lemma 49 and reduction preserves types lemma 32, there is $M_1 \in \mathcal{T}$ such that $M_1 \equiv_\beta M$ and $\vdash \lambda \mathcal{S} \; M_1 : \Pi_{x:s}.x$. Hence $y : s \vdash \lambda \mathcal{S} \; M_1 y : y$ and for any $C$ such that $\vdash \lambda \mathcal{S} \; C : s$ we get by substitution lemma that $\vdash \lambda \mathcal{S} \; M_1 C : C$. Hence, $\lambda \mathcal{S}$ is inconsistent. $\Box$

### 9 Curry Style to Church Style

An interpretation in the reverse direction requires a type to use as the domain in the Church-style syntax to interpret the Curry-style abstract $\lambda x.M$. For this purpose, let us introduce a new atomic constant $A$, so that the interpretation of $\lambda x.M$ in the Church-style system will be $\lambda x:A.M$. Then we will need to add the following rule to the system:

$$
(A\lambda) \quad \Gamma \vdash \beta \lambda x:B.M : \Pi_{x:B}.C \\
\Gamma \vdash \beta \lambda x:A.M : \Pi_{x:B}.C
$$

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

$$
\Gamma \vdash \text{Label}B(\lambda x.M) : \Pi_{x:B}.C
$$
to

\[ \Gamma \vdash \lambda x. M : \Pi x : B. C \]

provided that we think of the Church-style term \( \lambda x : B. M \) as being interpreted by the translation \( \rightarrow_L \), and is necessary for the proof of the Theorem 64 below.

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

**Definition 61** The function \( \overline{\text{Ch}} \) from the Curry-style syntax \( (T_c, T_l, T') \) to the Church-style syntax \( T \) is defined as follows:

\[
x^{\text{Ch}} \equiv x \quad (\lambda x. M)^{\text{Ch}} \equiv \lambda x^{\text{Ch}} A^{\text{Ch}} M^{\text{Ch}}
\]

\[
(\text{IB}(\lambda x. C))^\text{Ch} \equiv (\text{IB}(\lambda x. C))^{\text{Ch}}
\]

\[
(l B (\lambda x. C))^\text{Ch} \equiv \lambda x A \cdot C^{\text{Ch}}
\]

\[
(\Pi x : B. C)^\text{Ch} \equiv \Pi x : B \cdot C^{\text{Ch}}
\]

We define \( I^{\text{Ch}} \) in the obvious way.

Note that \( \overline{\text{Ch}} \) is partial on \( T' \) since we only define it for \( l' \)-bachelor-free terms.

The following lemma shows that the function \( \overline{\text{Ch}} \) is closed under free variables, substitution and reduction.

**Lemma 62** Let \( r \in \{\overline{\beta}, \beta, \beta'\} \). When \( r \) is \( \beta' \), we assume that we are only working with \( l' \)-bachelor-free terms.

1. If \( M \) is in either \( T_c, T_l \) or \( T' \), then \( \text{FV}(M^{\text{Ch}}) = \text{FV}(M) \).
2. \( (M[x := N])^{\text{Ch}} \equiv M^{\text{Ch}}[x := N^{\text{Ch}}] \).
3. If \( M \rightarrow_r N \) then \( M^{\text{Ch}} \rightarrow_{\beta} N^{\text{Ch}} \).
4. If \( M =_{=_{\beta}} N \) then \( M^{\text{Ch}} =_{=_{\beta}} N^{\text{Ch}} \).
5. If \( M^{\text{Ch}} \rightarrow_{\beta} N \) then \( N \equiv P^{\text{Ch}} \) and \( M \rightarrow_r P \).
6. If \( M^{\text{Ch}} =_{\beta} N^{\text{Ch}} \) then \( M =_{\beta} N \).

**Proof.** 1. and 2. By induction on \( M \).
3. First show by induction on the derivation of \( M \rightarrow_r N \) that if \( M \rightarrow_r N \) then \( M^{\text{Ch}} \rightarrow_{\beta} N^{\text{Ch}} \), and then we show the lemma by induction on the length of the derivation \( M \rightarrow_r N \).
4. Use 3 and CR of \( r \).
5. First show by induction on \( M \in T_c \) that if \( M^{\text{Ch}} \rightarrow_{\beta} N \) then \( N \equiv P^{\text{Ch}} \) and \( M \rightarrow_r P \). Then show the lemma by induction on the length of the derivation \( M^{\text{Ch}} \rightarrow_{\beta} N \).
6. By CR, \( M^{\text{Ch}} \rightarrow_{\beta} P \). By 5, \( P \equiv Q^{\text{Ch}} \) and \( M \rightarrow_r Q \). Hence, \( M =_{\beta} N \).

**Definition 63** We define \( \vdash_{\beta} A \) to be the type derivation relation given by the rules of Figures 2 and 3 (that is \( \vdash_{\beta} \)) and (AA) given above.

When \( \lambda S = (T, \beta, \vdash_{\beta} S) \) is the PTS induced by \( S \), we take \( \lambda S_A \) to be the tuple \( (T, \beta, \vdash_{\beta} S_A) \).

Note that the system \( \lambda S_A \) is not a PTS. Note also that there are no postulates which make it possible to deduce \( \vdash A : s \) for any sort \( s \). This means that in \( \lambda S_A \), the only place in which \( A \) can occur is in the domain of an abstraction. In particular, in \( \lambda S_A \), we cannot introduce assumptions of the form \( x : A \) into any legal context and we cannot prove a result of the form \( \Gamma \vdash M : A \).
Theorem 64 Let \( r \in \{\beta\beta, \beta', \beta''\} \). When \( r \) is \( \beta' \), we assume that we are only working with \( \Gamma \)-bachelor-free terms.

1. If \( \Gamma \vdash_{\beta}^{S} M : B \) then \( \Gamma^{\text{Ch}} \vdash_{\beta}^{S} M^{\text{Ch}} : B^{\text{Ch}} \).

2. If \( \Gamma \vdash_{\beta}^{S} \beta A : B \) then there are \( \Gamma_1, M_1, B_1 \) such that \( \Gamma^{\text{Ch}}_1 =_{\beta} \Gamma_1, B^{\text{Ch}}_1 =_{\beta} B_1 \),
   \[ \Gamma_1 \vdash_{\beta}^{S} M_1 : B_1 \text{ and if } M \equiv M_2^{\text{Ch}} \text{ then } M_1 \equiv M_2 \text{ else } M_1^{\text{Ch}} =_{\beta} M. \]

3. If \( \Gamma^{\text{Ch}} \vdash_{\beta}^{S} M^{\text{Ch}} : B^{\text{Ch}} \) then \( \Gamma \vdash_{\beta}^{S} M : B \).

Proof. 1. By induction on the derivation \( \Gamma \vdash_{\beta}^{S} M : B \).
2. By induction on the derivation \( \Gamma \vdash_{\beta}^{S} \beta A : B \).
3. By 2, \( \Gamma_1 \vdash_{\beta}^{S} M : B_1 \) where \( \Gamma^{\text{Ch}}_1 =_{\beta} \Gamma^{\text{Ch}}_1 \) and \( B^{\text{Ch}}_1 =_{\beta} B^{\text{Ch}}. \) By Lemma 62, \( \Gamma_1 =_{r} \Gamma \) and \( B_1 =_{r} B. \) By Lemma 47, \( \Gamma \vdash_{r}^{S} M : B_1 \) (it is easy to show that \( \Gamma \) is \( \vdash_{r}, \text{-legal} \)). Now, by correctness of types we have:
   - Either \( B^{\text{Ch}} \equiv s \text{ then } B \equiv s \text{ and } B_1 =_{r} s \text{ hence } B_1 \rightarrow_{r} s \text{ and by reduction preserves types lemma, } \Gamma \vdash_{r}^{S} M : s \equiv B. \)
   - Or \( \Gamma^{\text{Ch}} \vdash_{\beta}^{S} B^{\text{Ch}} : s \) and by 2, \( \Gamma_2 \vdash_{\beta}^{S} B : C \) where \( \Gamma^{\text{Ch}}_2 =_{\beta} \Gamma^{\text{Ch}} \) and \( C^{\text{Ch}} =_{\beta} s^{\text{Ch}}. \) From this we can deduce \( \Gamma \vdash_{r}^{S} B : s \) and by \( (\text{conv}_{r}), \Gamma \vdash_{r}^{S} M : B. \) \( \Box \)

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 \( (\lambda_x A, M) \) 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 \( (\lambda_x A, M) \). With this idea for an interpretation, we might think of rule \((A\lambda)\) 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 \( \lambda S A \) as defined above does not include any postulates to formalize this interpretation: all we have in \( \lambda S A \) 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:

\[
(\text{AI}) \quad \frac{\Gamma \vdash M : B}{\Gamma \vdash M : A}
\]

It might appear that this significantly strengthens the system. However, unless we add an axiom of the form \( A : s \) for some sort \( s \), it is impossible to prove that a type of the form \( \Pi x A.B \) has any sort as its type, so a type of this form cannot be the type of a conclusion of \( (\lambda) \). Hence, it does not appear that adding rule \((\text{AI})\) would have that much effect on the system. In particular, appears that adding rules \((\text{AI})\) and \((A\lambda)\) to a system satisfying strong normalization would preserve strong normalization.

Another possibility would be to interpret \( A \) as the type \( \omega \) that is the type of every term (or pseudoterm in a Church-style syntax, but with \( \omega \) 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 \((\text{AI})\), we would add the rule
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 \((\omega I)\), 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 9. They differ in that:

- For the first, the syntax of the \(\lambda\)-calculus (terms in \(T_c\)) was not extended but for this approach to work, the term \(\text{Label} \equiv \lambda u x . z x\) had to be typed and for this, we had to impose L-completeness conditions on the specifications by adding extra axioms in \(A\) and rules in \(R\).

- In the second approach the syntax was extended with terms of the form \(l A (\lambda x . b)\) and this did not require extra 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 this case, we did not need any new axioms/rules to type \(l A (\lambda x . B)\) whereas for the first approach, we needed to impose L-completeness in order to type \(\text{Label}\) when in fact, we only need \(\text{Label}\) 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 attempted to study the middle grounds between the first and second. Here, we extended the syntax with terms of the form \(l' A\) and in order to type \(l' A (\lambda x . B)\), we needed to also type \(l' A\). To do this, new axioms and rules needed to be added but less than those needed for the first approach (compare L-completeness with \(l'\)-completeness).

The following remarks are in order:

Remark 65 [Subtyping] If subtyping is present, \(\text{Label} A (\lambda x . M)\), \(l A (\lambda x . M)\) and \(l' A (\lambda x . M)\) are in a sense a restriction of \(\lambda x . M\) to the type \(A\) as domain, and \(\lambda x . M\) is a kind of universal function. Since \(\text{Label} A (\lambda x . M)\), \(l A (\lambda x . M)\) 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 \(\lambda\)-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 66** [$\eta$-reduction] If we could find a way to add $\eta$-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 $x \notin \text{FV}(M)$,

$$(\lambda x:B.M((\lambda u:B.u)x))$$

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

$$(\lambda x:B.M((\lambda u:B.u)x)) \to (\lambda x:B.Mx),$$

so $(\lambda x:B.Mx)$ also represents the restriction of $M$ to domain type $B$. But if we then apply a contraction by ($\eta_{\text{Ch}}$), 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 $\beta$-reduction.

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

**Remark 67** [multivariate $\lambda$-calculus] In his paper [17], Garrel Pottinger introduces a variety of the $\lambda$-calculus, which he calls the multivariate $\lambda$-calculus, in which a term of the form $\lambda x_1 x_2 \ldots x_n.M$ 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, $(\lambda x_1 x_2 \ldots x_nyz.M)NP$ is not a redex, but $(\lambda x_1 x_2 \ldots x_n(yz))MN$ is. If the multivariate $\lambda$-calculus is used, then Label would be a term which requires three arguments to make a redex.

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

$$(\lambda x_1 x_2 \ldots x_n.M)$$

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

$$\lambda u_1 \cdot \lambda u_2 \cdot \ldots \cdot \lambda u_n \cdot (\lambda x_1 x_2 \ldots x_n.M)u_1 u_2 \ldots u_n \to \beta \lambda u_1 \cdot \lambda u_2 \cdot \ldots \cdot \lambda u_n [(u_1/x_1, u_2/x_2, \ldots, u_n/x_n)M] \to \alpha \lambda x_1 \cdot \lambda x_2 \cdot \ldots \cdot \lambda x_n.M$$
and
\[ \lambda u_1 \lambda u_2 \ldots \lambda u_n. (\lambda x_1 x_2 \ldots x_n. M) u_1 u_2 \ldots u_n \rightarrow_\eta \lambda u_1 \lambda u_2 \ldots \lambda u_{n-1}. (\lambda x_1 x_2 \ldots x_n. M) u_1 u_2 \ldots u_{n-1} \]
\[ \vdots \]
\[ \rightarrow_\eta (\lambda x_1 x_2 \ldots x_n. M). \]

This tells us that if \( \eta \)-reduction is introduced into the multivariate \( \lambda \)-calculus in connection with the use with \textbf{Label} in the previous paragraph, then the distinction between functions and their restrictions would be lost, just as it is with \( \eta \) in the Church-style syntax as shown in Remark 65.

\section*{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 \( \eta \)-contractions and also the contraction scheme
\[ \lambda x:B. M \rightarrow \lambda x:A. M. \] \hfill (1)
This is roughly equivalent to adding Curry-style abstractions to the Church-style syntax and then adding the contraction scheme
\[ \lambda x:B. M \rightarrow \lambda x. M. \]
In conjunction with this extended reduction, the typing rule \((A\lambda)\) would preserve the Subject-Reduction Theorem. 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 \( \omega \) 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 \( \omega \) 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?

\section*{Acknowledgements}

We would like to thank Martin Bunder, Roger Hindley and Garrel Pottinger 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).
<table>
<thead>
<tr>
<th>Terms</th>
<th>$\tau$-reduction</th>
<th>$\Gamma^\tau$ Rules</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>$(\lambda x . A . B) C \rightarrow \beta B[x := C]$</td>
<td>$\vdash_{\beta}$ of Figures 2 and 3</td>
<td>PTSs [2]</td>
</tr>
<tr>
<td>$T_c$</td>
<td>$(\lambda x . C) \rightarrow \beta B[x := C]$</td>
<td>$\vdash_{\beta}$ of Figures 2 and 4</td>
<td>DFPTSs [3]</td>
</tr>
<tr>
<td>$\bar{T} \subset T_c$</td>
<td>$A =<em>{\beta} B \xrightarrow{LEM} \bar{A} =</em>{\beta} \bar{B}$</td>
<td>$\Gamma \vdash_{\beta} A : B$</td>
<td>DFPTS is $\lambda$-complete</td>
</tr>
<tr>
<td>$\bar{T} + \text{Label}$</td>
<td>$\bar{B} : (\lambda x . B) \rightarrow \beta B[x := C]$</td>
<td>$\vdash_{\beta}$ of Figures 2 and 4</td>
<td>$\lambda$-complete</td>
</tr>
<tr>
<td>$T_l$</td>
<td>$\beta = l \cup \beta'$ where $l A(\lambda x . b) \rightarrow_{l} \lambda x . b$</td>
<td>$\vdash_{\beta}$ of Figures 2, 4 and 5</td>
<td>LPTSs</td>
</tr>
<tr>
<td>$\bar{T} \subset T_l$</td>
<td>$A =<em>{\beta} B \xrightarrow{LEM} \bar{A} =</em>{\beta} \bar{B}$</td>
<td>$\Gamma \vdash_{\beta} A : B$</td>
<td>LPTSs</td>
</tr>
<tr>
<td>$T'$</td>
<td>$\beta' =  \Gamma' \cup \beta'$ where $\Gamma' A(\lambda x . b) \rightarrow_{\Gamma'} \lambda x . b$</td>
<td>$\vdash_{\beta'}$ of Figures 2, 4 and 6</td>
<td>$\Gamma'$-complete</td>
</tr>
<tr>
<td>$\bar{T} \subset T_l'$</td>
<td>$A =<em>{\beta} B \xrightarrow{LEM} \bar{A} =</em>{\beta'} \bar{B}$</td>
<td>$\Gamma \vdash_{\beta'} A : B$</td>
<td>$\Gamma'$-complete</td>
</tr>
<tr>
<td>$\text{Ch}$</td>
<td>$\Gamma \vdash M : B \xrightarrow{LEM} \Gamma \vdash_{S^{\text{Ch}}} M^{\text{Ch}} : B^{\text{Ch}}$</td>
<td>$\vdash_{\beta A}$ of Figures 2, 3 and $(A\lambda)$</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 9. Systems studied in this paper
References