# On Functions and Types

Fairouz Kamareddine

School of Mathematical and Computer Sciences, Heriot-Watt Univ., Riccarton, Edinburgh EH14 4AS, Scotland, fairouz@macs.hw.ac.uk

**Abstract.** The introduction of a general definition of function was key to Frege's formalisation of logic. Self-application of functions was at the heart of Russell's paradox. Russell introduced type theory in order to control the application of functions and hence to avoid the paradox. Since, different type systems have been introduced, each allowing different functional power. Eight of these influential systems have been unified in the so-called Barendregt cube. These systems use different binders for functions and types and do not allow types to have the same instantiation right as functions. De Bruijn did not always make these distinctions. In this paper, we discuss the modern, as well as de Bruijn's framework of functions and types and study the cube in different frameworks.

## 1 Summary

In [19, 17], a review of types and functions up to 1940 has been given. [19, 17] explain in detail the strong connection between functions and types. Since 1940, many type systems have been developed each allowing different functional power. In [3], an elegant cube which contains eight of these type systems has been given.

In the cube, both functions and types are created by abstractions: functions are created via $\lambda$ where $\lambda_{x:A}.B$ stands for the *function* from $A$ to $B$ which given $a \in A$ returns $B[x := a]$ (i.e., $B$ where $a$ is substituted for $x$); and types are created via $\Pi$ where $\Pi_{x:A}.B$ stands for the *type* of the functions from $A$ to $\cup_{a \in A} B[x := a]$ which given $a \in A$ return $fa \in B[x := a]$. However, in the cube, functions apply to arguments and $(\lambda_{x:A}.B)a$ reduces to $B[x := a]$ but types do not apply to arguments in the sense that $(\Pi_{x:A}.B)a$ is not allowed to reduce to anything. Instead, if needed, one writes $B[x := a]$ without ever implying that this could have come from $(\Pi_{x:A}.B)a$. In particular, the type of the application of a function $f$ of type $\Pi_{x:A}.B$ to an argument $a$ of type $A$ in the cube is given by $B[x := a]$ instead of $(\Pi_{x:A}.B)a$.

De Bruijn in his famous Automath [22] allowed $\Pi$-reduction where $(\Pi_{x:A}.B)a$ reduces to $B[x := a]$. He also used what we call *type instantiation* which is to apply both functions and types to arguments so that if $\lambda_{x:A}.b$ is of type $\Pi_{x:A}.B$ and $a$ is of type $A$ then $(\lambda_{x:A}.b)a$ has type $(\Pi_{x:A}.B)a$. Sometimes, de Bruijn unifed the $\lambda$ and $\Pi$ and wrote $[x : A]B$ for both $\lambda_{x:A}.B$ and $\Pi_{x:A}.B$.

Extending the cube with de Bruijn's ideas (of $\Pi$-reduction, type instantiation and unification of binders) has only partially been studied. [18] showed that type instantiation in the cube leads to the loss of subject reduction and correctness of

types. [18] did not investigate unifiying $\lambda$ and $\Pi$. Laan (private communication) rewrote the cube by replacing in the terms and rules, every $\lambda$ and $\Pi$ with a unique binder $\pi$. He did not investigate the extension with type instantiation.

In Section 2.2, we review the cube and its properties. In Section 2.3, we review the $\Pi$-cube of [18] where both $\Pi$-reduction and type instantiation are present and explain why correctness of types and subject reduction are lost. In Section 3, we present the $\pi$-cube where only $\Pi$-reduction is present and show that $\Pi$-redexes do not occur in legal terms and that this $\pi$-cube is isomorphic to the cube in the sense that they both have the same typing judgements and type the same terms. We show that the $\pi$-cube has all the properties of the cube. In Section 4, we present the $\flat$-cube where both $\lambda$ and $\Pi$ are written as $\flat$. We show that this $\flat$-cube is isomorphic to the $\pi$-cube and also satisfies all the properties.

The isomorphism between the type judgements of the $\flat$-cube, the $\pi$-cube and the cube shows that one gains nothing in by either unifying binders or allowing $\Pi$-reduction. The use of one binder can be seen as a cosmetic operation no more.

Next, we move to a cube where binders are unified and where also the instantiation behaviour of types and functions is allowed. In Section 5 we add to the $\flat$-cube type instantiation. We also add $\bullet$*parameters* which allow the treatment of higher level as well as lower level functions within one framework, $\bullet$*definitions* which enable us to abbreviate large expressions using identifiers and $\bullet$*explicit substitutions* which enable us to control the evaluation of terms. The result will be the $\flat_{i\delta\sigma p}$-cube with unified binders, instantiation on types, definitions, explicit substitutions and parameters providing a flexible and expressive framework.

## 2 The formal machinery

Frege gave in [8], a precise formalisation of logic which depended on a very general definition of function given via the *Abstraction Principle*:

> "If in an expression, [...] a simple or a compound sign has one or more occurrences and if we regard that sign as replaceable in all or some of these occurrences by something else (but everywhere by the same thing), then we call the part that remains invariant in the expression a function, and the replaceable part the argument of the function."

Frege's general notion of function meant that functions can take a variety of arguments and he was aware that types help avoid the paradoxes.

> " Now just as functions are fundamentally different from objects, so also functions whose arguments are and must be functions are fundamentally different from functions whose arguments are objects and cannot be anything else. I call the latter first-level, the former second-level."

Despite being cautious, his work faced a paradox discovered by Russell. To avoid the paradoxes, Russell used *ramified* types which have a double hierarchy: one of (simple) types (informally due Frege [9]) and one of orders. Ramified types faced many problems and hence the orders of the types were left out.

Church's combination of $\lambda$-calculus with simple types gave $\lambda\!\!\rightarrow$, the basis for most modern type systems. $\lambda\!\!\rightarrow$ however is very restrictive. The hierarchy of the

simple theory of types used by $\lambda\!\rightarrow$ leads to a duplication of work. For example, numbers, booleans, the identity function have to be defined at every level. This led to new (modern) type theories that allow more general notions of functions (e.g, *polymorphic*[1] functions) which avoid these and other problems (cf. [3]).

Unlike simple types, modern types have similar features to functions:

- We can construct a type by abstraction. E.g., the type $\Pi_{A:*}.\Pi_{y:A}.A$ of the polymorphic identity function is obtained by taking any type $A$ (we write $A : *$ for "$A$ is a type") and returning the type $\Pi_{y:A}.A$ of the identity function on $A$. The polymorphic identity function is written as $\lambda_{A:*}.\lambda_{y:A}.y$, the identity function over $A$ is $\lambda_{y:A}.y$ and the type of $\lambda_{A:*}.\lambda_{y:A}.y$ is $\Pi_{A:*}.\Pi_{y:A}.A$.
- We can *instantiate* types. E.g., if $A$ above is the set of natural numbers $\mathbb{N}$ then we are concerned with the identity function over $\mathbb{N}$ whose type is $\Pi_{y:A}.A$ where $A$ is substituted by $\mathbb{N}$ (written $(\Pi_{y:A}.A)[A := \mathbb{N}]$), i.e., $\Pi_{y:\mathbb{N}}.\mathbb{N}$.

De Bruijn [22] identifies the abstractions obtained by $\lambda$ and $\Pi$ whereas other modern type systems don't. De Bruijn also, unlike others, gives $\Pi$-terms the same instantiation power as $\lambda$-terms (i.e., Both $(\lambda_{x:A}B)C$ and $(\Pi_{x:A}B)C$ reduce to $B[x := C]$). In this paper we study de Bruijn's framework of types and functions by unifying both abstractions (via $\flat$), and allowing type instantiation within a set of eight type systems which form the Barendregt Cube.

### 2.1 Basic notions

**Definition 1** [Terms]

- We define the set of terms $\mathcal{T}$ by: $\mathcal{T} ::= *|\Box|\mathcal{V}|\pi_{\mathcal{V}:\mathcal{T}}.\mathcal{T}|\mathcal{T}\mathcal{T}$ where $\pi \in \{\lambda, \Pi\}$.
- We define the set of $\flat$-terms (or terms when no confusion occurs) $\mathcal{T}_\flat$ by: $\mathcal{T}_\flat ::= * \mid \Box \mid \mathcal{V} \mid \flat_{\mathcal{V}:\mathcal{T}_\flat}.\mathcal{T}_\flat \mid \mathcal{T}_\flat\mathcal{T}_\flat$.

**Notation 2** *We take $\mathcal{V}$ to be a set of variables over which, $x$, $y$, $z$, $x_1$, etc. range. We take capital letters $A$, $B$, $a$, $b$, etc. sometimes also indexed by Arabic numerals such as $A_1$, $A_2$, to range over terms. We use* FV($A$) *resp.* BV($A$) *to denote the free resp. bound variables of $A$, and $A[x := B]$ to denote the (implicit) substitution of all the free occurrences of $x$ in $A$ by $B$. 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. We also assume the Barendregt convention (BC) where names of bound variables are always chosen so that they differ from free ones in a term and where different abstraction operators bind different variables. Hence, for example, we write for the ordinary cube $(\pi_{y:A}.y)x$ instead of $(\pi_{x:A}.x)x$. (BC) will also be assumed for contexts and typings (for each of the calculi presented) 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.*

**Definition 3** [Reductions]

- Define $\beta$-reduction as the compatible closure of $(\lambda_{x:A}.B)C \rightarrow_\beta B[x := C]$.
- Define $\flat$-reduction as the compatible closure of $(\flat_{x:A}.B)C \rightarrow_\flat B[x := C]$.

[1] Polymorphism was already recognized by Russell as *typical ambiguity*.

- Define $\Pi$-reduction as the compatible closure of $(\Pi_{x:A}.B)C \to_\Pi B[x := C]$.
- We define the union of reduction relations as usual. For example, $\beta\Pi$-reduction is union of $\to_\beta$ and $\to_\Pi$.
- For each reduction relation $r$, $\twoheadrightarrow_r$ is the reflexive transitive closure of $\to_r$ and $=_r$ is the equivalence closure of $\to_r$. We write $\twoheadrightarrow_r^+$ to denote $r$-reduction in one or more steps. We say that $A$ is strongly normalising with respect to $\to_r$ (notation $\mathrm{SN}_{\to_r}(A)$) if there are no infinite $\to_r$-reductions starting at $A$.

**Definition 4** [declarations, contexts]

1. A *declaration* $d$ is of the form $x : A$. We define $\mathtt{var}(d) \equiv x$, $\mathtt{type}(d) \equiv A$ and $\mathrm{FV}(d) \equiv \mathrm{FV}(A)$. We let $d$, $d'$, $d_1$, etc. range over declarations.
2. A *context* $\Gamma$ is a (possibly empty) concatenation of declarations $d_1, d_2, \cdots, d_n$ such that if $i \neq j$, then $\mathtt{var}(d_i) \not\equiv \mathtt{var}(d_j)$. We define $\mathrm{DOM}(\Gamma) = \{\mathtt{var}(d) \mid d \in \Gamma\}$. The *empty context* is denoted throughout by $\langle\rangle$ or simply by $\emptyset$. We use $\Gamma$, $\Gamma_1$, $\Delta$, etc. as meta-variables for contexts.
3. We define substitutions on contexts by: $\emptyset[x := A] \equiv \emptyset$, and $(\Gamma, y : B)[x := A] \equiv \Gamma[x := A], y : B[x := A]$.

**Definition 5** [statements, judgements] Let $\Gamma$ be a context, $A, B, C$ be terms. Let $\vdash$ be one of the typing relations of Section 2

1. $A : B$ is called a *statement*. $A$ and $B$ are its *subject* and *predicate* respectively.
2. $\Gamma \vdash A : B$ is a *judgement*, and $\Gamma \vdash A : B : C$ denotes $\Gamma \vdash A : B \wedge \Gamma \vdash B : C$.
3. $\Gamma$ is *legal* if $\exists A_1, B_1$ terms such that $\Gamma \vdash A_1 : B_1$.
4. $A$ is a *$\Gamma$-term* if $\exists B_1$ term such that $[\Gamma \vdash A : B_1 \vee \Gamma \vdash B_1 : A]$.
5. $A$ is *legal* if $\exists \Gamma_1 [A$ is a $\Gamma_1$-term$]$.
6. If $d$ is a declaration then $\Gamma \vdash d$ iff $\Gamma \vdash \mathtt{var}(d) : \mathtt{type}(d)$.

### 2.2 Reviewing the Barendregt cube

In the Barendregt cube of [3], eight well-known type systems are presented in a uniform way. The weakest system is Church's simply typed $\lambda$-calculus $\lambda\!\to$ [6], and the strongest system is the Calculus of Constructions $\lambda$C [7]. The second order $\lambda$-calculus [10, 24] discovered independently by Girard and Reynolds figures on the cube between $\lambda\!\to$ and $\lambda$C. Moreover, via the Propositions-as-Types principle (see [14]), many logical systems can be described in the cube.

In the cube, we have in addition to the usual $\lambda$-abstraction, a type forming operator $\Pi$. Briefly, if $A$ is a type, and $B$ is a type possibly containing the variable $x$, then $\Pi_{x:A}.B$ is the type of functions that, given a term $a : A$, output a value of type $B[x := a]$. Here $a : A$ expresses that $a$ is of type $A$. If $x$ does not occur in $B$, then $\Pi_{x:A}.B$ is the type of functions from $A$ to $B$, written $A \to B$. To the $\Pi$-abstraction at the level of types corresponds $\lambda$-abstraction at the level of objects. Roughly speaking, if $M$ is a term of type $B$ ($M$ and $B$ possibly containing $x$), then $\lambda_{x:A}.M$ is a term of type $\Pi_{x:A}.B$. The cube has two sorts $*$ (the set of types) and $\Box$ (the set of kinds) with $* : \Box$. If $A : *$ (resp. $A : \Box$) we say $A$ is a type (resp. a kind). All systems of the cube have the same typing rules but are distinguished from one another by the set $\boldsymbol{R}$ of pairs of sorts $(s_1, s_2)$

allowed in the so-called *type-formation* or *$\Pi$-formation* rule, ($\Pi$). Each system of the cube has its own set $\boldsymbol{R}$ (which must contain $(*,*)$). A $\Pi$-type can only be formed in a specific system of the cube if rule ($\Pi$) is satisfied for some $(s_1, s_2)$ in the set $\boldsymbol{R}$ of that system. The rule ($\Pi$) is as follows:

$$(\Pi) \qquad \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash (\Pi_{x:A}.B) : s_2} \qquad (s_1, s_2) \in \boldsymbol{R}$$

As there are only two sorts, $*$ and $\Box$, and each set $\boldsymbol{R}$ must contain $(*,*)$, there are only eight possible different systems of the cube (see Figure 1). The dependencies between these systems is depicted in Figure 2. Furthermore, the systems in the cube are related to other type systems as is shown in the overview of Figure 1 (see [3]). With the rule ($\Pi$), an important aspect of the cube is that it provides a factorisation of the expressive power of the Calculus of Constructions into three features: *polymorphism, type constructors*, and *dependent types*:

- $(*,*)$ is the basic rule that forms types. All the cube systems have this rule.
- $(\Box,*)$ is the rule that takes care of polymorphism. Girard's System (also known as $\lambda 2$) is the weakest system on the cube that features this rule.
- $(\Box,\Box)$ takes care of type constructors. The system $\lambda\underline{\omega}$ is the weakest system on the cube that features this rule.
- $(*,\Box)$ takes care of term dependent types. The system $\lambda$P is the weakest system on the cube that features this rule.

**Definition 6** [The cube] The cube has $\mathcal{T}$ as the set of terms and $\beta$-reduction $\rightarrow_\beta$ for the reduction relation. Let $\boldsymbol{R} \subseteq \{(*,*),(*,\Box),(\Box,*),(\Box,\Box)\}$ such that $(*,*) \in \boldsymbol{R}$. The type system $\lambda\boldsymbol{R}$ describes how judgements $\Gamma \vdash_{\boldsymbol{R}} A : B$ (or $\Gamma \vdash A : B$, if it is clear which $\boldsymbol{R}$ is used) can be derived. $\Gamma \vdash A : B$ states that $A$ has type $B$ in context $\Gamma$. The typing rules are given in Figure 3 ($s, s_1, s_2 \in \{*, \Box\}$).

| System | Related system | Names, references | | | | |
|---|---|---|---|---|---|---|
| $\lambda\!\rightarrow$ | $\lambda^\tau$ | simply typed $\lambda$-calculus; [6, 2, 13] | $(*,*)$ | | | |
| $\lambda 2$ | F | 2nd order typed $\lambda$-calculus; [10, 24] | $(*,*)$ | $(\Box,*)$ | | |
| $\lambda$P | AUT-QE, LF | [5, 11] | $(*,*)$ | | $(*,\Box)$ | |
| $\lambda$P2 | | [20] | $(*,*)$ | $(\Box,*)$ | $(*,\Box)$ | |
| $\lambda\underline{\omega}$ | POLYREC | [23] | | | | |
| $\lambda\omega$ | F$\omega$ | [10] | $(*,*)$ | | | $(\Box,\Box)$ |
| $\lambda$P$\underline{\omega}$ | | | $(*,*)$ | | $(*,\Box)$ | $(\Box,\Box)$ |
| $\lambda$C | CC | Calculus of Constructions; [7] | $(*,*)$ | $(\Box,*)$ | $(*,\Box)$ | $(\Box,\Box)$ |

**Fig. 1.** Systems of the Barendregt cube

Below, we list the standard properties for the cube (see [3] for proofs).

**Theorem 7 (Church-Rosser Theorem for $\mathcal{T}$ and $\rightarrow_\beta$)** *Let $A, B_1, B_2 \in \mathcal{T}$. If $A \twoheadrightarrow_\beta B_1$ and $A \twoheadrightarrow_\beta B_2$ then there is a $C$ such that $B_1 \twoheadrightarrow_\beta C$ and $B_2 \twoheadrightarrow_\beta C$.*

**Lemma 8 (Free Variable Lemma for $\vdash$ and $\rightarrow_\beta$)**

1. *If $d$ and $d'$ are different elements in a legal context $\Gamma$, then* $\mathtt{var}(d) \not\equiv \mathtt{var}(d')$.

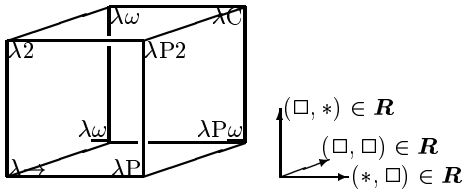


**Fig. 2.** The Barendregt cube

$$\text{(axiom)} \quad \langle\rangle \vdash * : \Box$$

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

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

$$(\Pi) \quad \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \Pi_{x:A}.B : s_2} \qquad (s_1, s_2) \in \boldsymbol{R}$$

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

$$\text{(appl)} \quad \frac{\Gamma \vdash F : \Pi_{x:A}.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x{:=}a]}$$

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

**Fig. 3.** Rules of the (Barendregt) cube

2. *If* $\Gamma \equiv \Gamma_1, d, \Gamma_2$ *and* $\Gamma \vdash B : C$ *then* $\text{FV}(d) \subseteq \text{DOM}(\Gamma_1)$ *and* $\text{FV}(B), \text{FV}(C) \subseteq \text{DOM}(\Gamma)$.

**Lemma 9 (Start and Context Lemma for $\vdash$ and $\to_\beta$)** *If* $\Gamma$ *is legal then* $\Gamma \vdash * : \Box$ *and* $\forall d \in \Gamma$, $\Gamma \vdash d$. *Moreover, if* $\Gamma \equiv \Gamma_1, d, \Gamma_2$ *then* $\Gamma_1 \vdash \texttt{type}(d) : s$ *for some sort* $s$.

**Lemma 10 (Substitution Lemma for $\vdash$ and $\to_\beta$)** *If* $\Gamma, x : A, \Delta \vdash B : C$ *and* $\Gamma \vdash D : A$ *then* $\Gamma, \Delta[x := D] \vdash B[x := D] : C[x := D]$.

**Lemma 11 (Thinning Lemma for $\vdash$ and $\to_\beta$)**
*If* $\Gamma$ *and* $\Delta$ *are legal,* $\Gamma \subseteq \Delta$, *and* $\Gamma \vdash A : B$ *then* $\Delta \vdash A : B$.

**Lemma 12 (Generation Lemma for $\vdash$ and $\to_\beta$)**

1. *If* $\Gamma \vdash s : C$ *then then* $s \equiv *$ *and* $C =_\beta \Box$, *furthermore if* $C \not\equiv \Box$ *then* $\Gamma \vdash C : s'$ *for some sort* $s'$.
2. *If* $\Gamma \vdash x : C$ *then there is a sort* $s$ *and* $B =_\beta C$ *such that* $\Gamma \vdash B : s$ *and* $x{:}B \in \Gamma$;
3. *If* $\Gamma \vdash (\lambda_{x:A}.B) : C$ *then there is sort* $s$ *and* $D$ *such that* $\Gamma \vdash (\Pi_{x:A}.D) : s$; $\Gamma, x{:}A \vdash B : D$; *and* $C =_\beta (\Pi_{x:A}.D)$;
4. *If* $\Gamma \vdash (\Pi_{x:A}.B) : C$ *then there is* $(s_1, s_2) \in \boldsymbol{R}$ *such that* $\Gamma \vdash A : s_1$, $\Gamma, x{:}A \vdash B : s_2$ *and* $C =_\beta s_2$;
5. *If* $\Gamma \vdash Fa : C$ *then there are* $A, B$ *such that* $\Gamma \vdash F : (\Pi_{x:A}.B)$, $\Gamma \vdash a : A$ *and* $C =_\beta B[x{:=}a]$.

**Lemma 13 (Correctness of types for $\vdash$ and $\to_\beta$)**
*If $\Gamma \vdash A : B$ then ($B \equiv \Box$ or $\Gamma \vdash B : s$ for some sort $s$).*

**Lemma 14 (Typability of subterms for $\vdash$ and $\to_\beta$)**
*If $A$ is legal and $B$ is a subterm of $A$, then $B$ is legal.*

**Lemma 15 (Subject Reduction for $\vdash$ and $\to_\beta$)**
*If $\Gamma \vdash A : B$ and $A \twoheadrightarrow_\beta A'$ then $\Gamma \vdash A' : B$.*

**Lemma 16 (Reduction preserves legal terms for $\vdash$ and $\to_\beta$)**
1. *If $\Gamma \vdash A : B$ and $B \twoheadrightarrow_\beta B'$ then $\Gamma \vdash A : B'$.*
2. *If $A$ is a $\Gamma$-term and $A \twoheadrightarrow_\beta A'$ then $A'$ is a $\Gamma$-term.*

**Lemma 17 (Uniqueness of Types for $\vdash$ and $\to_\beta$)**
*If $\Gamma \vdash A_1 : B_1$ and $\Gamma \vdash A_2 : B_2$ and $A_1 =_\beta A_2$, then $B_1 =_\beta B_2$.*

**Theorem 18 (Strong Normalisation for $\vdash$ and $\to_\beta$)**
*If $A$ is $\vdash$-legal then $SN_{\to_\beta}(A)$.*

### 2.3 Reviewing the $\Pi$-cube: $\Pi$-reduction and type instantiation

[18] provided the $\Pi$-cube which extends the cube with both $\Pi$-reduction and type instantiation. In this section, we review the $\Pi$-cube and its properties.

**Definition 19** [The $\Pi$-cube] The $\Pi$-cube has $\mathcal{T}$ as the set of terms and $\beta\Pi$-reduction $\to_{\beta\Pi}$ for the reduction relation. The typing rules of the $\Pi$-cube are those of Definition 6 but where $=_\beta$ in the (conv) rule is replaced by $=_{\beta\Pi}$ and (appl) is replaced by (new appl):

$$\text{(new appl)} \qquad \frac{\Gamma \vdash_\Pi F : (\Pi_{x:A}.B) \qquad \Gamma \vdash_\Pi a : A}{\Gamma \vdash_\Pi Fa : (\Pi_{x:A}.B)a}$$

We write $\vdash_\Pi$ to denote type derivation in the $\Pi$-cube.

[18] showed that Theorem 7 where one replaces every $\twoheadrightarrow_\beta$ by $\twoheadrightarrow_\beta \Pi$, and Lemmas 8..11 where one replaces every $\vdash$ by $\vdash_\Pi$ hold for the $\Pi$-cube. [18] also showed that Lemma 12 holds for the $\Pi$-cube if one replaces $\vdash$ by $\vdash_\Pi$, $=_\beta$ everywhere by $=_{\beta\Pi}$ and if in clause 5., $B[x := a]$ is replaced by $(\Pi_{x:A}.B)a$. However, [18] showed that both correctness of types Lemma 13 and subject reduction Lemma 15 fail for the $\Pi$-cube. Finally, strong normalisation Theorem 18 holds for the $\Pi$-cube where $\vdash_\Pi$ and $\to_{\beta\Pi}$ replace $\vdash$ and $\to_\beta$ respectively.

In order to understand why correctness of types and subject reduction fail in the $\Pi$-cube but not in the cube, let us reflect on the legal terms in both cubes.

**Lemma 20** *$\Gamma \not\vdash \Box : A$, $\Gamma \not\vdash AB : \Box$, $\Gamma \not\vdash \lambda_{x:A}.B : s$, and $\Gamma \not\vdash (\Pi_{x:A}.B)a : C$.*

PROOF: For the first 3 statements, see [3]. For the fourth, assume $\Gamma \vdash (\Pi_{x:A}.B)a : C$. By Lemma 12, $\exists A', B'$ such that $\Gamma \vdash \Pi_{x:A}.B : \Pi_{y:A'}.B'$. Again by Lemma 12, $\exists (s_1, s_2) \in \boldsymbol{R}$ such that $\Pi_{y:A'}.B' =_\beta s_2$ contradicting Church Rosser. $\boxtimes$

**Lemma 21** *$\Gamma \not\vdash_{\Pi} \Box : A$, $\Gamma \not\vdash_{\Pi} AB : \Box$, $\Gamma \not\vdash_{\Pi} \lambda_{x:A}.B : s$. However, terms of the form $(\Pi_{x:A}.B)a$ can be legal, but, $\Gamma \not\vdash_{\Pi} (\Pi_{x:A}.B)a : s$.*

PROOF: All the statements have the same proofs as those of Lemma 20. As for a legal $\Pi$-redex, take for example $z : * \vdash_{\Pi} (\Pi_{x:z}.z)z : **$ and hence terms of the form $(\Pi_{x:A}.B)a$ can be legal. It is these new legal terms that led to the loss of correctness of types of the $\Pi$-cube and hence of subject reduction because they can not have a sort as a type. The proof is similar to that in Lemma 20. ⊠

The fact that these new legal terms $(\Pi_{x:A}.B)a$ cannot have type $s$, that they are $\not\equiv \Box$ and they are the types of other terms, lead to the loss of correctness of types and hence of subject reduction.

**Example 22** *$z : *, x : z \vdash_{\Pi} (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$ hence loss of correctness of types. Also, $(\lambda_{y:z}.y)x \to_{\beta\Pi} x$ but $z : *, x : z \not\vdash_{\Pi} x : (\Pi_{y:z}.z)x$.*

[15] proposed the $\Pi_{\delta}$-cube which has $\Pi$-reduction and type instnatiation, but where both correctness of types and subject reduction hold. The idea was to add the so-called definitions to the $\Pi$-cube.

**Definition 23** [The $\Pi_{\delta}$-cube] The $\Pi_{\delta}$-cube has $\mathcal{T}$ as the set of terms and $\beta\Pi$-reduction $\to_{\beta\Pi}$ for the reduction relation. The contexts of the $\Pi_{\delta}$-cube are changed by allowing in addition to the usual declarations, definitions of the form $x = B : A$ which define $x$ to be $B$ and to have type $A$. The typing rules of the $\Pi_{\delta}$-cube are those of Definition 19 but where $=_{\beta\Pi}$ in the (conv) rule is replaced by $\Gamma \vdash_{\Pi_{\delta}} B \stackrel{\text{def}}{=} B'$ is the smallest equivalence relation closed under:

- If $B =_{\beta\Pi} B'$ then $\Gamma \vdash_{\Pi_{\delta}} B \stackrel{\text{def}}{=} B'$
- If $x = D : C \in \Gamma$ and $B'$ arises from $B$ by substituting one particular free occurrence of $x$ in $B$ by $D$ then $\Gamma \vdash_{\Pi_{\delta}} B \stackrel{\text{def}}{=} B'$

and three new rules are added:

$$\text{(start-def)} \quad \frac{\Gamma \vdash_{\Pi_{\delta}} A : s \qquad \Gamma \vdash_{\Pi_{\delta}} B : A}{\Gamma, x = B{:}A \vdash_{\Pi_{\delta}} x : A} \quad x \notin \text{DOM}(\Gamma)$$

$$\text{(weak-def)} \quad \frac{\Gamma \vdash_{\Pi_{\delta}} A : B \qquad \Gamma \vdash_{\Pi_{\delta}} C : s \qquad \Gamma \vdash_{\Pi_{\delta}} D : C}{\Gamma, x = D{:}C \vdash_{\Pi_{\delta}} A : B} \quad x \notin \text{DOM}(\Gamma)$$

$$\text{(def)} \quad \frac{\Gamma, x = B{:}A \vdash_{\Pi_{\delta}} C : D}{\Gamma \vdash_{\Pi_{\delta}} (\pi_{x:A}.C)B : D[x := B]}$$

Let us see now how the problem explained in Example 22 disappears:
First, the example is no longer a counterexample for correctness of types:
By (weak-def) $z : *, x : z, y = x : z \vdash_{\Pi_{\delta}} z : *$.
Hence by (def) $z : *, x : z \vdash_{\Pi_{\delta}} (\Pi_{y:z}.z)x : *[y := x] \equiv *$.
Second, the example is no longer a counterexample for subject reduction:
As $z : *, x : z \vdash_{\Pi_{\delta}} x : z$, $z : *, x : z \vdash_{\Pi_{\delta}} (\Pi_{y:z}.z)x : *$ and $z : *, x : z \vdash_{\Pi_{\delta}} z \stackrel{\text{def}}{=} (\Pi_{y:z}.z)x$, we use (conv) to get: $z : *, x : z \vdash_{\Pi_{\delta}} x : (\Pi_{y:z}.z)x$.

## 3 The $\pi$-cube: allowing $\Pi$-reduction only

We extend the cube with $\Pi$-reduction (without type instantiation). Unlike the $\Pi$-cube, we show that the $\pi$-cube has all the properties of the cube. However, we will also show that the $\pi$-cube is a trivial extension of the cube in the sense that if $\Gamma \vdash_\pi A : B$ then $\Gamma \vdash A : B$ and $\Gamma, A$ and $B$ are free of $\Pi$-redexes.

**Definition 24** [The $\pi$-cube] The $\pi$-cube has $\mathcal{T}$ as the set of terms and $\beta\Pi$-reduction $\to_{\beta\Pi}$ for the reduction relation. The typing rules of the $\pi$-cube are those of Definition 6 but where $=_\beta$ in the (conv) rule is replaced by $=_{\beta\Pi}$.

We write $\vdash_\pi$ to denote type derivation in the $\pi$-cube.

As the typing relation does not play a role in the Church Rosser Theorem, Church Rosser for the $\pi$-cube holds and has the same proof as that for the $\Pi$-cube. Lemmas 8..11 where one replaces every $\vdash$ by $\vdash_\pi$ hold for the $\pi$-cube and have the same proofs as those for the ordinary cube. The generation lemma for the $\pi$-cube (and its proof) is the same as that of Lemma 12 but where $\vdash_\pi$ and $=_{\beta\Pi}$ replace $\vdash$ and $=_\beta$. Also, Lemmas 13 and 14 where one replaces every $\vdash$ by $\vdash_\pi$ hold for the $\pi$-cube and have the same proofs as those for the ordinary cube.

Now, having Church Rosser and the generation, substitution and typability of subterms lemmas for the $\pi$-cube, we can establish the following lemma:

**Lemma 25** *$\Gamma \not\vdash_\pi \Box : A$, $\Gamma \not\vdash_\pi AB : \Box$, $\Gamma \not\vdash_\pi \lambda_{x:A}.B : s$, and $\Gamma \not\vdash_\pi (\Pi_{x:A}.B)a : C$. Moreover, if $\Gamma \vdash_\pi A : B$ then all of $\Gamma, A$ and $B$ are free of $\Pi$-redexes.*

PROOF: The proof of each statement except the last is similar to that in Lemma 20. For the last statement, use induction on $\Gamma \vdash_\pi A : B$. We only show the (appl) case. By induction $F$ and $a$ and $\Gamma$ are free of $\Pi$-redexes. By this lemma, $Fa$ is also free of $\Pi$-redexes. By generation and substitution we can show that $\Gamma \vdash_\pi B[x := a] : s$ and by Lemma 14 all subterms of $B[x := a]$ are typable. Hence, by this lemma, none of the subterms of $B[x := a]$ can be a $\Pi$-redex. $\boxtimes$

By this lemma, the proof of subject reduction is similar to that for the cube.

**Lemma 26 (Subject Reduction for $\vdash_\pi$ and $\to_{\beta\Pi}$)**
*If $\Gamma \vdash_\pi A : B$ and $A \twoheadrightarrow_{\beta\Pi} A'$ then $\Gamma \vdash_\pi A' : B$.*

PROOF: Similar to Lemma 15 as in the (appl) case, in the derivable statement $\Gamma \vdash_\pi Fa : B[x := a]$, it is not possible that $F$ be of the form $\Pi_{y:C}.D$. $\boxtimes$

Lemmas 16 and 17 hold for the $\pi$-cube and have similar proofs to those of the cube (change to $\twoheadrightarrow_{\beta\Pi}$ and $\vdash_\pi$). Next we show that $\Pi$-redexes play no role.

**Lemma 27** *1. Let $\Gamma \vdash_\pi A : B$. a) if $A \to_{\beta\Pi} A'$ then $A \to_\beta A'$. b) $A \twoheadrightarrow_{\beta\Pi} A'$ then $A \twoheadrightarrow_\beta A'$. c) if $A =_{\beta\Pi} A'$ then $A =_\beta A'$*
*2. $\Gamma \vdash A : B$ if and only if $\Gamma \vdash_\pi A : B$.*

PROOF:

1. a) By Lemma 25, $A$ is free of $\Pi$-redexes. b) By induction on $A \twoheadrightarrow_{\beta\Pi} A'$. Assume $A \twoheadrightarrow^n_{\beta\Pi} A'' \to_{\beta\Pi} A'$. By subject reduction, $\Gamma \vdash_\pi A'' : B$ and hence by IH, $A \twoheadrightarrow^n_\beta A''$ and $A'' \to_\beta A'$. Hence, $A \to_\beta A'$. c) By Church Rosser, $\exists C$ such that $A \twoheadrightarrow_{\beta\Pi} C$ and $A' \twoheadrightarrow_{\beta\Pi} C$. By subject reduction, $\Gamma \vdash_\pi A' : B$. Hence by a), $A \twoheadrightarrow_\beta C$ and $A' \twoheadrightarrow_\beta C$. Hence $A =_\beta A'$.
2. One direction is trivial because every $\vdash$-rule is also a $\vdash_\pi$-rule (for (conv), note that $=_\beta \subseteq =_{\beta\Pi}$). For the other direction, use induction on $\Gamma \vdash_\pi A : B$. We only show the (conv) case. If $\Gamma \vdash_\pi A : B'$ comes from $\Gamma \vdash_\pi A : B$, $\Gamma \vdash_\pi B : s$ and $B' =_{\beta\Pi} B$. By a) $B' =_\beta B$. Hence, by IH and (conv) $\Gamma \vdash A : B$. ⊠

**Theorem 28 (Strong Normalisation for $\vdash_\pi$ and $\to_{\beta\Pi}$)** *If $A$ is $\vdash_\pi$-legal then $SN_{\to_{\beta\Pi}}(A)$.*

PROOF: We only need to show that if $\Gamma \vdash_\pi A : B$ then $\mathrm{SN}_{\to_{\beta\Pi}}(A)$. By Lemma 27.2, $\Gamma \vdash A : B$ and by Theorem 18 $\mathrm{SN}_{\to_\beta}(A)$. If there is an infinite path $A \to_{\beta\Pi} A_1 \to_{\beta\Pi} A_2 \ldots$ then by Lemma 27.1, there is an infinite path $A \to_\beta A_1 \to_\beta A_2 \ldots$. Contradiction. ⊠

## 4 The $\flat$-cube: Identifying $\lambda$ and $\Pi$ in the cube

In Section 3, we showed that adding $\Pi$-reduction to the cube preserves all the properties, but that this addition does not have any influence on the legal terms or typing relation. That is, $\Pi$-reduction never takes place on legal terms, and one cannot type more terms than already possible. The typing relations of the cube and the $\pi$-cube are equivalent. Although, we allowed $\Pi$ and $\lambda$ to behave alike in reductions, in legal terms only $\lambda$ redexes exist and are active at reductions. $\Pi$-redexes never occur in legal terms, and hence never take place. What if we rename both $\lambda$s and $\Pi$s using one unique name, say $\flat$? Definitely legal terms will contain $\flat$-redexes, but do we keep all the desirable properties of the cube?

Our study is motivated by de Bruijn [22] who wrote $[x : A]B$ for both $\lambda_{x:A}.B$ and $\Pi_{x:A}.B$. We will replace all the $\lambda$s and $\Pi$s of Section 2.2 by $\flat_{x:A}.B$ which represents de Bruijn's $[x : A]B$. This variant of the ordinary cube will be shown to be equivalent to the $\pi$-cube and to have all the desirable properties.

**Definition 29** [The $\flat$-cube] The $\flat$-cube has $\mathcal{T}_\flat$ as the set of terms and $\flat$-reduction $\to_\flat$ for the reduction relation. The $\flat$-cube judgements are defined by changing in Definition 6, every $\lambda$ and $\Pi$ in the rules ($\Pi$), ($\lambda$) and (appl) to $\flat$. We call these new rules ($\flat_1$), ($\flat_2$) and (app$\flat$) respectively. When necessary, we write $\Gamma \vdash_\flat A : B$ instead of $\Gamma \vdash A : B$.

In order to investigate the connection between the $\flat$-cube, and the $\pi$-cube and cube. It is useful to define a translation function between their terms $\mathcal{T}$ and $\mathcal{T}_\flat$:

**Definition 30** – For $A \in \mathcal{T}$, we define $\overline{A} \in \mathcal{T}_\flat$ as follows:
$\overline{s} \equiv s \quad \overline{x} \equiv x \quad \overline{AB} \equiv \overline{A}\,\overline{B} \quad \overline{\lambda_{x:A}.B} \equiv \overline{\Pi_{x:A}.B} \equiv \flat_{x:\overline{A}}.\overline{B}.$
For contexts we define: $\overline{\langle\rangle} \equiv \langle\rangle \quad \overline{\Gamma, x : A} \equiv \overline{\Gamma}, x : \overline{A}.$

– For $A \in \mathcal{T}_\flat$, we define $[A]$ to be $\{A' \in \mathcal{T}$ such that $\overline{A'} \equiv A\}$.
For context, obviously: $[\Gamma] \equiv \{\Gamma'$ such that $\overline{\Gamma'} \equiv \Gamma\}$.

**Lemma 31**

1. *If $A, B \in \mathcal{T}$ then $\overline{A[x := B]} \equiv \overline{A}[x := \overline{B}]$.*
2. *Let $A, B \in \mathcal{T}_\flat$ and $R \in \{\to, \twoheadrightarrow\}$. If $A R_\flat B$ then $A' R_{\beta\Pi} B'$ for all $A' \in [A]$ and $B' \in [B]$.*
3. *Let $A, B \in \mathcal{T}$ and $R \in \{\to, \twoheadrightarrow, =\}$. If $A R_{\beta\Pi} B$ then $\overline{A} R_\flat \overline{B}$.*
4. *If $A \in \mathcal{T}_\flat$ then $[A] \neq \emptyset$.*
5. *Let $A \in \mathcal{T}$. If $SN_{\to_{\beta\Pi}}(A)$ then $SN_{\to_\flat}(\overline{A})$.*
6. *Let $A \in \mathcal{T}_\flat$. If $SN_{\to_\flat}(A)$ then $SN_{\to_{\beta\Pi}}(A')$ for all $A' \in [A]$.*

PROOF: 1. By induction on $A$. 2. For $\to_\flat$, by induction on $A \to_\flat B$. For $\twoheadrightarrow_\flat$, use induction on the number of reduction steps. 3. For $\to_{\beta\Pi}$, by induction on $A \to_{\beta\Pi} B$. For $\twoheadrightarrow_{\beta\Pi}$, use induction on the number of reduction steps. For $=_{\beta\Pi}$, take $A =_{\beta\Pi} B$ and use Church Rosser for the $\Pi$-cube to find $A \twoheadrightarrow_{\beta\Pi} A_1$ and $B \twoheadrightarrow_{\beta\Pi} A_1$ and then use the earlier statement for $\twoheadrightarrow_{\beta\Pi}$. 4. By induction on $A$. (An $A'$ can be found by replacing each $\flat$ by $\lambda$.) 5. Let $A$ where $\mathrm{SN}_{\to_{\beta\Pi}}(A)$. Assume an infinite path $\overline{A} \to_\flat A_1 \to_\flat A_2 \ldots$. By 4, let $A'_i \in [A_i]$. Then, by 2, $A \to_{\beta\Pi} A'_1 \to_{\beta\Pi} A'_2 \ldots$ contradiction. 6. similar to 5, using 3. ⊠

**Theorem 32 (Church-Rosser Theorem)** *Let $A, B_1, B_2 \in \mathcal{T}_\flat$. If $A \twoheadrightarrow_\flat B_1$ and $A \twoheadrightarrow_\flat B_2$ then there is a $C$ such that $B_1 \twoheadrightarrow_\flat C$ and $B_2 \twoheadrightarrow_\flat C$.*

PROOF: By Lemma 31, $[A]$, $[B]$ and $[B']$ are all non empty. Let $A' \in [A]$, $B'_1 \in [B_1]$ and $B'_2 \in [B_2]$. By Lemma 31, $A' \twoheadrightarrow_{\beta\Pi} B'_1$ and $A' \twoheadrightarrow_{\beta\Pi} B'_2$ and hence by Church Rosser for the $\Pi$-cube, there is a $C'$ such that $B'_1 \twoheadrightarrow_{\beta\Pi} C'$ and $B'_2 \twoheadrightarrow_{\beta\Pi} C'$. Now use Lemma 31 again to get that $B_1 \equiv \overline{B'_1} \twoheadrightarrow_\flat C'$ and $B_2 \equiv \overline{B'_2} \twoheadrightarrow_\flat C'$. ⊠

**Corollary 33** *Let $A, B \in \mathcal{T}_\flat$. If $A =_\flat B$ then $A' =_{\beta\Pi} B'$ for all $A' \in [A]$ and $B' \in [B]$.*

Lemmas 8..11 and 13..17 are formulated for the $\flat$-cube in a similar way (replace all $\Pi$s and $\lambda$s by $\flat$ and every $\vdash$ by $\vdash_\flat$) and have similar proofs to the cube. For the generation lemma, as now both $(\flat_1)$ and $(\flat_1)$ type terms of the form $\flat_{x:A}.B$, we need to combine clauses 3. and 4. of Lemma 12 depending on whether $(\flat_1)$ or $(\flat_2)$ are used. The generation lemma changes as below, but its proof is similar to that of Lemma 12. Note that only one of the subclauses applies.

**Lemma 34 (Generation lemma for the $\flat$-cube)** *The generation lemma for the $\flat$-cube has clauses 1., 2., and 5., of Lemma 12, where $\vdash$, $=_\beta$ and $\Pi_{x:A}.B$ change to $\vdash_\flat$, $=_\flat$ and $\flat_{x:A}.B$ respectively and clauses 3. and 4. change to:*

*3+4. If $\Gamma \vdash_\flat (\flat_{x:A}.B) : C$ then* **only one** *of the following holds:*

– *Either there is sort $s$ and $D$ such that $\Gamma \vdash_\flat (\flat_{x:A}.D) : s$; $\Gamma, x{:}A \vdash_\flat B : D$; and $C =_\flat (\flat_{x:A}.D)$;*

– *Or there is* $(s_1, s_2) \in \boldsymbol{R}$ *such that* $\Gamma \vdash_\flat A : s_1$, $\Gamma, x{:}A \vdash_\flat B : s_2$ *and* $C =_\flat s_2$;

The next theorem connects the typing judgements in the cube with the $\pi$-cube. A less general version of this theorem was stated (for the cube and without a proof) in a short note by Twan Laan (private communications) in which he also stated Definition 29, item 1 of Definition 30 and the generation lemma.

**Theorem 35** *1. If* $\Gamma \vdash_\pi A : B$ *then* $\overline{\Gamma} \vdash_\flat \overline{A} : \overline{B}$.

*2. If* $\Gamma \vdash_\flat A : B$ *then there exists* $\Gamma' \in [\Gamma]$ *such that* $\Gamma'$ *is the only* $\vdash_\pi$*-legal context of* $[\Gamma]$ *and there are unique* $A' \in [A]$ *and* $B' \in [B]$ *such that* $\Gamma' \vdash_\pi A' : B'$.

PROOF: 1. By induction on $\Gamma \vdash_\pi A : B$. 2. By induction on $\Gamma \vdash_\flat A : B$. (axiom) and (start) are easy. **(weak)**: If $\Gamma, x : C \vdash_\flat A : B$ comes from $\Gamma \vdash_\flat A : B$ and $\Gamma \vdash_\flat C : s$, then by IH, let $\Gamma'$ be the unique legal context in $[\Gamma]$. Let $A', B'$ and $C'$ be the unique elements such that $\Gamma' \vdash_\pi A' : B'$ and $\Gamma' \vdash_\pi C' : s$ (by unicity of the legal context, we write $\Gamma'$ in both judgements). Hence, by (weak) $\Gamma', x : C' \vdash_\pi A' : B'$. As for unicity, if $\Gamma'', x : C'' \vdash_\pi A'' : B''$ then by context lemma $\Gamma'' \vdash_\pi C'' : s'$ where $\Gamma'' \in [\Gamma]$ and $C'' \in [C]$. Hence $\Gamma'' \equiv \Gamma'$ by unicity of legal $\Gamma' \in [\Gamma]$ and $C'' \equiv C'$ by IH. As $FV(A'', B'') = FV(A', B') \subseteq \text{DOM}(\Gamma')$, hence $\Gamma' \vdash_\pi A'' : B''$ and hence by IH, $A'' \equiv A'$ and $B'' \equiv B'$.
**($\flat_2$)**: Assume $\Gamma \vdash_\flat \flat_{x:A}.b : \flat_{x:A}.B$ comes from $\Gamma, x : A \vdash_\flat b : B$ and $\Gamma \vdash_\flat \flat_{x:A}.B : s$. By IH, $\Gamma', x : A' \vdash_\pi b' : B'$ and $\Gamma'' \vdash_\pi \Pi_{x:A''}.B'' : s$. (Note the use of $\Pi$ instead of $\lambda$ because it is easy to show that $\Gamma \not\vdash_\pi \lambda_{x:D}.E : s$.) It is easy to show that $B' \not\equiv \Box$ and hence by correctness of types, $\Gamma', x : A' \vdash_\pi B' : s$. Now, by generation lemma, $\Gamma'', x : A'' \vdash_\pi B'' : s$ and hence by IH, $\Gamma'' \equiv \Gamma'$, $A' \equiv A''$ and $B' \equiv B''$. Hence, by $(\Pi)$ $\Gamma' \vdash_\pi \lambda_{x:A'}.B' : \Pi_{x:A'}.B'$. (Note the use of $\lambda$ and $\Pi$. It is easy to show that using $\vdash_\pi$ it is not possible to derive $\Pi_{x:D}.e : \Pi_{x:D}.E$, $\Pi_{x:D}.e : \lambda_{x:D}.E$ or $\lambda_{x:D}.e : \lambda_{x:D}.E$.) As for unicity, Assume $\Gamma'' \vdash_\pi \lambda_{x:A''}.B'' : \Pi_{x:A''}.B''$ where all elements belong to the right class. Obviously by uniqueness of legal contexts in the same class, $\Gamma'' \equiv \Gamma'$. By correctness of types, $\Gamma' \vdash_\pi \Pi_{x:A''}.B'' : s$. By generation, $\Gamma', x : A'' \vdash_\pi B'' : s$ and hence by IH, $A'' \equiv A'$. As $\Gamma', x : A' \vdash_\pi B' : s'$, by IH again, $B'' \equiv B'$.
**($\flat_1$)**: Assume $\Gamma \vdash_\flat \flat_{x:A}.B : s_2$ comes from $\Gamma, x : A \vdash_\flat B : s_2$ and $\Gamma \vdash_\flat A : s_1$ for $(s_1, s_2)$ rule. By IH, there is are unique legal contexts $\Gamma' \in [\Gamma]$ and $\Gamma'', x : A'' \in [\Gamma, x : A]$ and there are unique terms $A' \in [A], B' \in [B]$ such that $\Gamma' \vdash_\pi A' : s_1$ and $\Gamma'', x : A'' \vdash_\pi B' : s_2$. By context lemma, $\Gamma'' \vdash_\pi A'' : s'$ with $\Gamma'' \in [\Gamma]$ and $A'' \in [A]$. Hence, by IH, $\Gamma'' \equiv \Gamma'$ and $A'' \equiv A'$. Hence, by $(\Pi)$ we have $\Gamma' \vdash_\pi \Pi_{x:A'}.B' : s_2$ with $\Pi_{x:A'}.B' \in [\flat_{x:A}.B]$. As for unicity, if there are other class elements such that $\Gamma'' \vdash_\pi \Pi_{x:A''}.B'' : C$ then by unicty of legal $\Gamma' \in [\Gamma]$, $\Gamma'' \equiv \Gamma'$. As $C \in [s_2]$ then $C \equiv s_2$. By generation lemma, $\Gamma', x : A'' \vdash_\pi B'' : s_2$ and $\Gamma' \vdash_\pi A'' : s$. Hence by IH, $A'' \equiv A'$ and again by IH, $B'' \equiv B'$.
**(conv)**: Assume $\Gamma \vdash_\flat A : C$ comes from $\Gamma \vdash_\flat A : B$, $\Gamma \vdash_\flat C : s$ and $B =_\flat C$. By IH, there is a unique legal context $\Gamma' \in [\Gamma]$ and there are unique $A' \in [A], B' \in [B]$ and $C' \in [C]$ such that $\Gamma' \vdash_\pi A' : B'$ and $\Gamma' \vdash_\pi C' : s$. By Corollary 33, $B' =_{\beta\Pi} C'$. Hence, by (conv), $\Gamma' \vdash_\pi A' : C'$. For unicity, assume $\Gamma' \vdash_\pi A'' : C''$ where $A'' \in [A]$ and $C'' \in [C]$ (recall $\Gamma'$ is the only legal context in $[\Gamma]$). By

correctness of types lemma, either $C'' \equiv \Box$ or $\Gamma' \vdash_\pi C'' : s'$. But $C'' \not\equiv \Box$ else $C' \equiv \Box$ and $\Gamma' \vdash_\pi \Box : s'$ absurd. As $\Gamma' \vdash_\pi C'' : s'$, $\Gamma' \vdash_\pi C' : s$, and $C'' \in [C]$, we get by IH, $C'' \equiv C'$. Note that $B' \not\equiv \Box$, else, $C' \twoheadrightarrow_{\beta\Pi} \Box$, and as $\Gamma' \vdash_\pi C' : s$, we get by subject reduction that $\Gamma' \vdash_\pi \Box : s$, absurd. Hence, as $\Gamma' \vdash_\pi A' : B'$, we get by correctness of types that $\Gamma' \vdash_\pi B' : s'$. Now, as $\Gamma' \vdash_\pi A'' : C'$, $\Gamma' \vdash_\pi B' : s'$ and $B' =_{\beta\Pi} C'$, by (conv) $\Gamma' \vdash_\pi A'' : B'$. Hence, by IH, $A' \equiv A''$.
**(app$\flat$)**: Assume $\Gamma \vdash_\flat Fa : B[x := a]$ comes from $\Gamma \vdash_\flat F : \flat_{x:A}.B$ and $\Gamma \vdash_\flat a : A$. By IH, there is a unique legal $\Gamma' \in [\Gamma]$ and there are unique $a' \in [a], F' \in [F], A', A'' \in [A], B' \in [B]$ such that $\Gamma' \vdash_\pi F' : \Pi_{x:A'}.B'$ and $\Gamma' \vdash_\pi a' : A''$. (Note that we took $\Gamma' \vdash_\pi F' : \Pi_{x:A'}.B'$ instead of $\Gamma' \vdash_\pi F' : \lambda_{x:A'}.B'$ because otherwise, we get $\Gamma' \vdash_\pi \lambda_{x:A'}.B' : s$ contradicting Lemma 25.) By generation lemma on $\Gamma' \vdash_\pi F' : \Pi_{x:A'}.B'$, $\Gamma' \vdash_\pi A' : s_1$ and $\Gamma', x : A' \vdash_\pi B' : s_2$ for $(s_1, s_2)$ rule. Hence, $A'' \not\equiv \Box$ (else $A \equiv A' \equiv \Box$ and hence $\Gamma' \vdash_\pi \Box : s$ contradicting Lemma 25). Hence, by correctness of types on $\Gamma' \vdash_\pi a' : A''$ we get $\Gamma' \vdash_\pi A'' : s$. By IH, $A'' \equiv A'$. Hence by (app$\flat$) $\Gamma' \vdash_\pi \mathcal{F}'a' : B'[x := a']$. For unicity, assume that $\Gamma' \vdash_\pi \mathcal{F}''a'' : C$. Then by generation, $\Gamma' \vdash_\pi F'' : \Pi_{y:D}.E$, $\Gamma' \vdash_\pi a'' : D$ and $C =_{\beta\Pi} E[y := a'']$. By IH, $F'' \equiv F'$, $a'' \equiv a'$, $y \equiv x$, $D \equiv A'$ and $E \equiv B'$. Hence, $C =_{\beta\Pi} B'[x := a']$. But $C \in [B'[x := a']]$. Therefore, $C \equiv B'[x := a']$. $\boxtimes$

**Theorem 36 (Strong Normalisation)** *If $A$ is $\vdash_\flat$-legal then $SN_{\to_\flat}(A)$.*

Proof: As $A$ is legal, then either $\Gamma \vdash_\flat A : B$ or $\Gamma \vdash_\flat B : A$. If $\Gamma \vdash_\flat B : A$ then by Lemma 13 for the $\flat$-cube, $A \equiv \Box$ (hence $\mathrm{SN}_{\to_\flat}(A)$) or $\Gamma \vdash_\flat A : s$. Hence, we only prove the theorem for $\Gamma \vdash_\flat A : B$. By Theorem 35, there are unique $\Gamma', A'$ and $B'$ such that $\Gamma' \vdash_\pi A' : B'$ and $\overline{\Gamma'} \equiv \Gamma$, $\overline{A'} \equiv A$ and $\overline{B'} \equiv B$. By Theorem 28, $\mathrm{SN}_{\to_{\beta\Pi}}(A')$ and hence by Lemma 31 $\mathrm{SN}_{\to_\flat}(\overline{A'})$, i.e., $\mathrm{SN}_{\to_\flat}(A)$. $\boxtimes$

## 5 The $\flat_{i\delta\sigma p}$-cube

So far, we showed that the $\flat$-cube can be seen as a cosmetic version of the cube and that in the $\flat$-cube, the cube, and the $\pi$-cube, the same terms can be typed. It is obvious to ask next, whether extending the $\flat$-cube with type instantiations will face the same fate as extending the ordinary cube with type instantiations. We have not yet investigated this question, but we suspect that the answer is yes. Note that we showed the isomorphism of type judgements in the $\pi$-cube and $\flat$-cube by heavily relying on correctness of types and subject reduction in the $\pi$-cube. Subject reduction and correctness of types do not hold in the $\Pi$-cube.

We will devote this section to a few extensions (including type instantiation) to the $\flat$-cube in the spirit of Automath. These extensions are as follows:

1. *Type instantiation* We will follow [18] and replace (app$\flat$) by (new app$\flat$):

$$\text{(new app}\flat\text{)} \qquad \frac{\Gamma \vdash F : (\flat_{x:A}.B) \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : (\flat_{x:A}.B)a}$$

   The main reason for this new rule is *compatibility*. The ($\flat_2$) rule in Definition 29 may be regarded as the compatibility property for typing with

respect to abstraction. That is, $b : B$ implies $\flat_{x:A}.b : \flat_{x:A}.B$. Compatibility for typing with respect to application is lost however. From the (app$\flat$) rule, $F : \flat_{x:A}.B$ implies $Fa : B[x := a]$ instead of $Fa : (\flat_{x:A}.B)a$.

2. *Definitions* (also known as *abbreviations* or *let expressions*) were heavily used in Automath and have since been exploited in functional languages like Haskell and ML and theorem provers like Coq. The idea is that if $k$ occurs in a text $f$, it may be practical to introduce an abbreviation for $k$: (cf. [16]):
   - The representation of $k$ may be long. This makes manipulations with $f$ a time- and memory-consuming task, in particular when $k$ occurs several times in $f$. Abbreviating $k$ can make manipulations with $f$ easier.
   - The object $k$ may represent a structure that is particularly interesting. Abbreviating $k$ opens the possibility to introduce a significant name for $k$. This makes the expression easier to understand for human beings.

   Definition 23 explained how definitions were added to the $\Pi$-cube. We will follow the same process when adding definitions in $\flat$-cube (see Definition 41).[2]
3. *Explicit substitution* The substitution required by $\beta$-reduction in any implementation of the $\lambda$-calculus must be implemented via smaller operations. Thus, there is a conceptual gap between the theory of the $\lambda$-calculus and its implementation. By representing substitutions in the structure of terms and by providing step-wise reductions to propagate the substitutions, explicit substitution provides a number of benefits such as more flexibility in ordering work and postponing unneeded work indefinitely.
   To add explicit substitutions to the $\flat$-cube, we extend the set of terms $\mathcal{T}_\flat$ with substitution terms of the form $A[x \leftarrow B]$. We add substitution rules to say how substitutions propagate through terms, and we replace $B[x := A]$ in the $\beta$-rule by $B[x \leftarrow A]$ which allows us to control the substitutions (see Definition 40). Finally, we need to add a new typing rule (subst) which explains how terms of the form $B[x \leftarrow A]$ can be typed (see Definition 41).
4. *Parameters* were heavily used in Automath and later studied in [19]. The $\lambda$-calculus accommodates the higher level approach of functions where functions are first class citizens. However, in many practical systems, the lower level approach where functions always occur with their arguments (parameters) and do not stand alone is necessary. The programming language ML for instance, was not be based on Girard's system $F$ (as it was not known then whether type checking in $F$ is decidable). Hence, terms like the polymorphic identity $\mathtt{Id} = \lambda_\alpha.\lambda_x.x : \Pi_\alpha.\alpha \to \alpha$ cannot be typed in ML. However, terms like $\mathtt{Id}(\alpha) = \lambda_x.x : \alpha \to \alpha$ can be typed in ML and hence, `Id` was introduced with parameters, and never alone. [3]
   [19] gives extensive reasons why parameters are needed in many areas of logic. Laan shows parameters have advantages on a large spectrum ranging

[2] It is interesting to investigate whether the (def) rule can be eliminated. The (def) rule was used in [15, 16] because without it, Subject reduction and Correctness of types fail for the ordinary cube with $\Pi$-reduction and (new appl). We have not investigated if Subject reduction and Correctness of types also fail for the $\flat$-cube with (new app$\flat$). Nonetheless, we add (def) because it provides smaller type derivations (see [16]).

[3] Note that ML is implicitly typed, i.e., one writes $\lambda_x.B$ instead of $\lambda_{x:A}.B$

from expressivity, to decidability, and to being able to relate different systems together. The reader is referred to [19] for more details.

We add parameters à la Laan to our $\flat$-cube. We will add parametric terms of the form $c(b_1, \ldots, b_n)$ where the parameters are $b_1, \ldots, b_n$. We need to add new typing rules to type these new terms. These can be found in Definition 41. Just as we allow several kinds of $\Pi$-constructs (via the set $\boldsymbol{R}$) in the cube, we follow [19, 16] and allow several kinds of parametric constructs via a set $\boldsymbol{P}$, consisting of tuples $(s_1, s_2)$ where $s_1, s_2 \in \{*, \Box\}$. $(s_1, s_2) \in \boldsymbol{P}$ means that we allow parametric constructs $c(b_1, \ldots, b_n) : A$ where $b_1, \ldots, b_n$ have types $B_1, \ldots, B_n$ of sort $s_1$, and $A$ is of type $s_2$. If both $(*, s_2) \in \boldsymbol{P}$ and $(\Box, s_2) \in \boldsymbol{P}$ then combinations of parameters are possible.

### 5.1 The extension

Extensions of the ordinary cube with parameters alone, with definitions and type instantiation alone, with explicit substitutions alone, and with explicit substitutions with definitions have already taken place in the literature. [16] gave an extension of the ordinary cube with three of these concepts: definitions, type instantiation and explicit substitutions, and another separate extension with parameters alone. Here we give an extension with all these concepts in one cube. In addition, our extension deals with a unified binder $\flat$ instead of the usual separate $\lambda$ and $\Pi$ present. It is the hope that providing as many of the useful extensions in one system, will result in a practical framework that combines all the advantages. Of course it remains to be investigated how the $\flat$-cube fares with each of these extensions separately. This is left for future work.

**Definition 37** The set $\mathcal{T}_a$ of terms of the $\flat_{i\delta\sigma p}$-cube is defined together with the set $\mathcal{L}_T$ of *lists of terms* as follows by:
$\mathcal{T}_a ::= * \mid \Box \mid \mathcal{V} \mid \mathcal{C}(\mathcal{L}_T) \mid \flat_{\mathcal{V}:\mathcal{T}_a}.\mathcal{T}_a \mid \mathcal{T}_a\mathcal{T}_a \mid \mathcal{T}_a[\mathcal{V} \leftarrow \mathcal{T}_a]$, and $\mathcal{L}_T ::= \varnothing \mid \mathcal{L}_T, \mathcal{T}_a$.
$\mathcal{C}$ (over which $c, c', \ldots$ range) is a set of constants disjoint from $\mathcal{V}$.

In Notation 2, the notions of $\textsc{fv}(A)$, $\textsc{bv}(A)$, implicit substitution $A[x := B]$ and compatibility are extended to take into account the new terms of the form $A[x \leftarrow B]$ and $c(b_1, \ldots, b_n)$. In particular,
if $b_i \to_\flat b_i'$ then $c(b_1, \ldots, b_i, \ldots, b_n) \to_\flat c(b_1, \ldots, b_i', \ldots, b_n)$ for $1 \leq i \leq n$.
$\textsc{fv}(A[x \leftarrow B]) = \textsc{fv}(A) \setminus \{x\}) \cup \textsc{fv}(B)$
$\textsc{fv}(c(a_1, \ldots, a_n)) = \bigcup_{i=1}^n \textsc{fv}(a_i)$
$(A[x \leftarrow B])[y := C] \equiv (A[y := C])[x \leftarrow B[y := C]]$,
$c(b_1, \ldots, b_n)[x{:=}A] \equiv c(b_1[x{:=}A], \ldots, b_n[x{:=}A])$.
In addition, Barendregt's Convention BC is extended to the new terms. E.g., a term $(\flat_{y:A}.B)[y \leftarrow C]$ is renamed to $(\flat_{x:A}.B[y := x])[y \leftarrow C]$ where $x$ is fresh.

**Definition 38** [Constants of terms] Define $\textsc{cons}(A)$, the *constants* of $A$ by:
$\textsc{cons}(s) = \textsc{cons}(x) = \emptyset$; $\textsc{cons}(c(a_1, \ldots, a_n)) = \{c\} \cup \bigcup_{i=1}^n \textsc{cons}(a_i)$;
$\textsc{cons}(AB) = \textsc{cons}(\flat_{x:A}.B) = \textsc{cons}(A[x \leftarrow B]) = \textsc{cons}(A) \cup \textsc{cons}(B)$;

**Definition 39** [declarations, definitions, contexts, $\subseteq'$]

1. A *variable declaration* $d$ is of the form $x : A$. We define $\mathtt{var}(d) \equiv x$, $\mathtt{type}(d) \equiv A$, $\text{FV}(d) \equiv \text{FV}(A)$ and $\text{CONS}(d) \equiv \text{CONS}(A)$.
2. A *constant declaration* $d$ is of the form $c(x_1{:}B_1, \ldots, x_n{:}B_n){:}A$ where $c \in \mathcal{C}$. We define $\mathtt{type}(d) \equiv A$ and dec-cons$(d) \equiv c$. $c$ is called a *primitive constant*. $x_1, \ldots, x_n$ are the *parameters* of $d$. We define $\text{FV}(d)$ to be $\text{FV}(A) \cup \text{FV}(B_1) \cdots \cup \text{FV}(B_n)$ and $\text{CONS}(d)$ to be $\text{CONS}(A) \cup \text{CONS}(B_1) \cdots \cup \text{CONS}(B_n)$.
3. A *definition* $d$ is of the form $x = B : A$ and defines $x$ of type $A$ to be $B$. We define $\mathtt{var}(d)$, $\mathtt{type}(d)$ and $\mathtt{ab}(d)$ to be $x$, $A$, and $B$ respectively. We define $\text{FV}(d) \equiv \text{FV}(A) \cup \text{FV}(B)$ and $\text{CONS}(d)$ to be $\text{CONS}(A) \cup \text{CONS}(B)$.
4. $d, d', d_1, \ldots$ range over declarations (variables/constants) and definitions.
5. A *context* $\Gamma$ is a (possibly empty) concatenation of declarations and definitions $d_1, d_2, \cdots, d_n$ such that if $i \neq j$, then $\mathtt{var}(d_i) \not\equiv \mathtt{var}(d_j)$ if $d_i$ and $d_j$ are either variable declarations or definitions, and dec-cons$(d_i) \not\equiv$ dec-cons$(d_j)$ if $d_i$ and $d_j$ are constant declarations. We define $\text{DOM}(\Gamma) = \{\mathtt{var}(d) \mid d \text{ is a variable declaration or a definition in } \Gamma\}$.
   Define $\text{CONS}(\Gamma)$ to be the set $\{\text{dec-cons}(d) \mid d \text{ is a constant declaration in } \Gamma\}$.
   $\Gamma\text{-}\mathtt{decl} = \{d \in \Gamma \mid d \text{ is a declaration }\}$ and $\Gamma\text{-}\mathtt{abb} = \{d \in \Gamma \mid d \text{ is a definition }\}$.
   We use $\Gamma, \Delta, \Gamma', \Gamma_1, \Gamma_2, \ldots$ to range over contexts.
6. We define substitutions on contexts by: $\emptyset[x := A] \equiv \emptyset$; $(\Gamma, y : B)[x := A] \equiv \Gamma[x := A], y : B[x := A]$; $(\Gamma, c(x_1 : A_1, \ldots, x_n : A_n) : C)[x := A] \equiv \Gamma[x := A], c(x_1 : A_1[x := A], \ldots, x_n : A_n[x := A]) : C[x := A]$; and $(\Gamma, y = B : C)[x := A] \equiv \Gamma[x := A], y = B[x := A] : C[x := A]$.
7. Define $\subseteq'$ between contexts as the least reflexive transitive relation satisfying:
   - $\Gamma, \Delta \subseteq' \Gamma, d, \Delta$ for $d$ a declaration (variable/constant) or a definition.
   - $\Gamma, x : A, \Delta \subseteq' \Gamma, x = B : A, \Delta$

**Definition 40** [$a$-Reduction] $a$-Reduction $\to_a$ is defined as the union of $\to_{\flat'}$ and $\to_\sigma$ which are defined as the compatible closures of, respectively:

$$
\begin{array}{ll}
(\flat_{x:A}.B)C & \to_{\flat'} B[x \leftarrow C] \\
(\flat_{y:A}.B)[x \leftarrow C] & \to_\sigma \flat_{y:A[x\leftarrow C]}.B[x \leftarrow C] \\
(c(b_1, \ldots, b_n))[x \leftarrow C] & \to_\sigma c(b_1[x \leftarrow C] \ldots, b_n[x \leftarrow C]) \\
(AB)[x \leftarrow C] & \to_\sigma A[x \leftarrow C].B[x \leftarrow C] \\
x[x \leftarrow C] & \to_\sigma C \\
A[x \leftarrow C] & \to_\sigma A \quad \text{if } x \notin \text{FV}(A)
\end{array}
$$

**Definition 41** Let $\boldsymbol{R}$ be as in Definition 6 and let $(*, *) \in \boldsymbol{P}$ and $\boldsymbol{P}$ be a subset of $\{(*,*), (*,\Box), (\Box,*), (\Box,\Box)\}$. The judgements that are derivable in $\lambda\boldsymbol{RP}$ are determined by the typing rules for $\lambda\boldsymbol{R}$ of Definition 29 where the (conv) and (app$\flat$) rules are replaced by (new conv) and (new app$\flat$) and where six new rules (start-def), (weak-def), (subst), (def), ($\vec{\text{C}}$-weak) and ($\vec{\text{C}}$-app) are added (in the last two rules, $\Delta \equiv x_1{:}B_1, \ldots, x_n{:}B_n$ and $\Delta_i \equiv x_1{:}B_1, \ldots, x_{i-1}{:}B_{i-1}$). The new and changed rules are given in Figure 4. In (new-conv), $\Gamma \vdash_a B \stackrel{\text{def}}{=} B'$ is defined on $\mathcal{T}_a$ as the smallest equivalence relation closed under:

- If $B =_a B'$ then $\Gamma \vdash_a B \stackrel{\text{def}}{=} B'$
- If $x = D : C \in \Gamma$ and $B'$ arises from $B$ by substituting one particular free occurrence of $x$ in $B$ by $D$ then $\Gamma \vdash_a B \stackrel{\text{def}}{=} B'$.

$$\text{(start-def)}\quad \frac{\Gamma \vdash_a A : s \qquad \Gamma \vdash_a B : A}{\Gamma, x = B{:}A \vdash_a x : A} \qquad x \notin \text{DOM}(\Gamma)$$

$$\text{(weak-def)}\quad \frac{\Gamma \vdash_a A : B \qquad \Gamma \vdash_a C : s \qquad \Gamma \vdash_a D : C}{\Gamma, x = D{:}C \vdash_a A : B} \qquad x \notin \text{DOM}(\Gamma)$$

$$\text{(subst)}\quad \frac{\Gamma, x = B{:}A \vdash_a C : D}{\Gamma \vdash_a C[x \leftarrow B] : D[x := B]}$$

$$\text{(def)}\quad \frac{\Gamma, x = B{:}A \vdash_a C : D}{\Gamma \vdash_a (\flat_{x:A}.C)B : D[x := B]}$$

$$\text{(new conv)}\quad \frac{\Gamma \vdash_a A : B \qquad \Gamma \vdash_a B' : s \qquad \Gamma \vdash_a B \stackrel{\text{def}}{=} B'}{\Gamma \vdash_a A : B'}$$

$$\text{(new app}\flat\text{)}\quad \frac{\Gamma \vdash_a F : (\flat_{x:A}.B) \qquad \Gamma \vdash_a a : A}{\Gamma \vdash_a Fa : (\flat_{x:A}.B)a}$$

$$\vec{\textbf{(C-weak)}}\ \frac{\Gamma \vdash_a b : B \quad \Gamma, \Delta_i \vdash_a B_i : s_i \quad \Gamma, \Delta \vdash_a A : s}{\Gamma, c(\Delta) : A \vdash_a b : B} \quad (s_i, s) \in \boldsymbol{P},\ c \notin \text{CONS}(\Gamma)$$

$$\vec{\textbf{(C-app)}}\ \frac{\begin{array}{ll}\Gamma_1, c(\Delta){:}A, \Gamma_2 \vdash_a b_i{:}B_i[x_j{:=}b_j]_{j=1}^{i-1} & (i = 1, \ldots, n)\\ \Gamma_1, c(\Delta){:}A, \Gamma_2 \vdash_a A : s & (\text{if } n = 0)\end{array}}{\Gamma_1, c(\Delta){:}A, \Gamma_2 \vdash_a c(b_1, \ldots, b_n) : A[x_j{:=}b_j]_{j=1}^{n}}$$

**Fig. 4.** New/changed rules of the cube

**Definition 42** [Statements, judgements, legal terms and contexts] Definition 4 is extended to $\mathcal{T}_a$ and $\vdash_a$ by changing everywhere in definition 4, $\mathcal{T}$ by $\mathcal{T}_a$, $\vdash$ by $\vdash_a$ and by changing item 6 to the following:
If $d$ is a variable declaration then $\Gamma \vdash_a d$ iff $\Gamma \vdash_a \mathtt{var}(d) : \mathtt{type}(d)$.
Otherwise, if $d$ is a definition then $\Gamma \vdash_a d$ iff $\Gamma \vdash_a \mathtt{var}(d) : \mathtt{type}(d) \wedge \Gamma \vdash_a \mathtt{ab}(d) : \mathtt{type}(d) \wedge \Gamma \vdash_a \mathtt{var}(d) \stackrel{\text{def}}{=} \mathtt{ab}(d)$. Otherwise,
if $d \equiv c(x_1 : B_1, \ldots, x_n : B_n){:}A$ and $n = 0$ then $\Gamma \vdash_a d$ is defined as $\Gamma \vdash_a c : A$.
Else, if $d \equiv c(x_1 : B_1, \ldots, x_n : B_n){:}A$ and $n \neq 0$ then $\Gamma \vdash_a d$ is defined as $\Gamma \vdash_a c(b_1, \ldots, b_n){:}A[x_j{:=}b_j]_{j=1}^{n}$ whenever $\Gamma \vdash_a b_i : B_i[x_j{:=}b_j]_{j=1}^{i-1}$ for $1 \leq i \leq n$.

### 5.2 Properties of the extension

The $\flat_{i\delta\sigma p}$-cube can be seen as a union of the unified binder versions of two cubes:

- The $e$-cube which is the ordinary cube extended with definitions, $\Pi$-reduction, type instantiation and explicit substitution $\Pi\sigma$DEF-cubeof [16].
- The $f$-cube which is the ordinary cube with parameters of [19, 16].

Hence, its properties are formulated by a union of those of [16], and are proved by using the same methods (but in the single binder framework) or showing isomorphic correspondences between ordinary versions and unified binder versions as we did in Section 4.

**Lemma 43 (Free variable Lemma for $\vdash_a$ and $\rightarrow_a$)**

1. *If $d$ and $d'$ are different declarations or definitions (none of which is a constant declaration) in a legal context $\Gamma$, then* $\mathtt{var}(d) \not\equiv \mathtt{var}(d')$.

2. *If $d$ and $d'$ are different constant declarations in a legal context $\Gamma$, then dec-cons $(d) \not\equiv$ dec-cons $(d')$.*
3. *If $\Gamma \equiv \Gamma_1, d, \Gamma_2$ and $\Gamma \vdash_a B : C$ then*
   - $\text{CONS}(d) \subseteq \text{CONS}(\Gamma_1)$,
   - $\text{FV}(d) \subseteq \begin{cases} \text{DOM}(\Gamma_1) & \text{if } d \text{ is a variable declaration or a definition} \\ \text{DOM}(\Gamma_1, x_1{:}B_1, \ldots, x_n{:}B_n) & \text{if } d \equiv c(x_1{:}B_1, \ldots, x_n{:}B_n){:}A \end{cases}$
   - $\text{FV}(B), \text{FV}(C) \subseteq \text{DOM}(\Gamma)$ *and* $\text{CONS}(B), \text{CONS}(C) \subseteq \text{CONS}(\Gamma)$.

**Lemma 44 (Substitution Lemma for $\vdash_a$ and $\to_a$)** *Let $d$ be $x = D : C$, $\Delta_d$ be $\Delta[x := D]$, $A_d$ be $A[x := D]$ and $B_d$ be $B[x := D]$. The following holds:*

1. *If $\Gamma, d, \Delta \vdash_a A \stackrel{\text{def}}{=} B$, $A$ and $B$ are $\Gamma, d, \Delta$-legal, then $\Gamma, \Delta_d \vdash_a A_d \stackrel{\text{def}}{=} B_d$.*
2. *If $B$ is a $\Gamma, d$-legal term, then $\Gamma, d \vdash_a B \stackrel{\text{def}}{=} B_d$.*
3. *If $\Gamma, d, \Delta \vdash_a A : B$ or $(\Gamma, x : C, \Delta \vdash_a A : B$ and $\Gamma \vdash_a D : C)$ then $\Gamma, \Delta_d \vdash_a A_d : B_d$.*

**Corollary 45** *Assume $\mathtt{var}(d) \notin \text{FV}(A) \cup \text{FV}(B) \cup \text{FV}(\Delta)$. Then:*

- *If $\Gamma, d, \Delta \vdash_a A : B$ then $\Gamma, \Delta \vdash_a A : B$.*
- *If $\Gamma, d, \Delta \vdash_a A \stackrel{\text{def}}{=} B$ then $\Gamma, \Delta \vdash_a A \stackrel{\text{def}}{=} B$.*

**Lemma 46 (Start Lemma for $\vdash_a$ and $\to_a$)** *Let $\Gamma$ be a $\vdash_a$-legal context. Then $\Gamma \vdash_a * : \Box$ and $\forall d \in \Gamma[\Gamma \vdash_a d]$.*

**Lemma 47 (Context Lemma for $\vdash_a$)** *Let $\Gamma_1, d, \Gamma_2$ be a $\vdash_a$-legal context.*

- *If $d$ is a variable declaration then $\Gamma_1 \vdash_a \mathtt{type}(d) : s$ for some sort $s$, $\Gamma_1, d \vdash_a \mathtt{var}(d) : \mathtt{type}(d)$.*
- *If $d$ is a definition then $\Gamma_1 \vdash_a \mathtt{type}(d) : s$ for some sort $s$, $\Gamma_1, d \vdash_a \mathtt{var}(d) : \mathtt{type}(d)$ and $\Gamma_1 \vdash_a \mathtt{ab}(d) : \mathtt{type}(d)$.*
- *If $d \equiv c(x_1 : B_1, \ldots, x_n : B_n){:}A$ then for some sort $s$, $\Gamma_1, x_1 : B_1, \ldots, x_n : B_n \vdash_a A : s$ and for some sorts $s_i$, for $1 \leq i \leq n$ where $(s_i, s) \in \boldsymbol{P}$, we have $\Gamma_1, x_1 : B_1, \ldots, x_{i-1} : B_{i-1} \vdash_a B_i : s_i$.*

**Lemma 48 (Thinning Lemma for $\vdash_a$ and $\to_a$)** *Let $d$ be either a declaration or a definition and let $\Gamma_1, d, \Gamma_2$ be a legal context.*

1. *If $\Gamma_1, \Gamma_2 \vdash_a A \stackrel{\text{def}}{=} B$, then $\Gamma_1, d, \Gamma_2 \vdash_a A \stackrel{\text{def}}{=} B$.*
2. *If $\Gamma_1, \Gamma_2 \vdash_a A : B$, then $\Gamma_1, d, \Gamma_2 \vdash_a A : B$.*
3. *If $d$ is $x = D : C$ and $\Gamma_1, x : C, \Gamma_2 \vdash_a A : B$, then $\Gamma_1, d, \Gamma_2 \vdash_a A : B$.*

**Lemma 49 (Generation Lemma for $\vdash_a$ and $\to_a$)**

1. *If $\Gamma \vdash_a s : C$ then $s \equiv *$ and $\Gamma \vdash_a C \stackrel{\text{def}}{=} \Box$, furthermore if $C \not\equiv \Box$ then $\Gamma \vdash_a C : s'$ for some sort $s'$.*
2. *If $\Gamma \vdash_a x : A$ then for some $d \in \Gamma$, $x \equiv \mathtt{var}(d)$, $\Gamma \vdash_a A \stackrel{\text{def}}{=} \mathtt{type}(d)$ and $\Gamma \vdash_a A : s$ for some sort $s$.*
3. *If $\Gamma \vdash_a \flat_{x:A}.B : C$ then*
   - *Either there is $D$ and sort $s$ where $\Gamma, x : A \vdash_a B : D$, $\Gamma \vdash_a \flat_{x:A}.D : s$, $\Gamma \vdash_a \flat_{x:A}.D \stackrel{\text{def}}{=} C$ and if $\flat_{x:A}.D \not\equiv C$ then $\Gamma \vdash_a C : s'$ for some sort $s'$.*

– *Or for some sorts* $s_1, s_2$*:* $\Gamma \vdash_a A : s_1$*,* $\Gamma, x : A, \vdash_a B : s_2$*,* $(s_1, s_2)$ *is a rule,* $\Gamma \vdash_a C \stackrel{\text{def}}{=} s_2$ *and if* $s_2 \not\equiv C$ *then* $\Gamma \vdash_a C : s$ *for some sort* $s$*.*

4. *If* $\Gamma \vdash_a Fa : C$*,* $F \not\equiv \flat_{x:A}.B$*, then for some* $D, E$*:* $\Gamma \vdash_a a : D$*,* $\Gamma \vdash_a F : \flat_{x:D}.E$*,* $\Gamma \vdash_a (\flat_{x:D}.E)a \stackrel{\text{def}}{=} C$ *and if* $(\flat_{x:D}.E)a \not\equiv C$ *then* $\Gamma \vdash_a C : s$ *for some* $s$*.*
5. *If* $\Gamma \vdash_a (\flat_{x:A}.D)B : C$*, then* $\Gamma, x = B : A \vdash_a D : C$*.*
6. *If* $\Gamma \vdash_a A[x \leftarrow B] : C$*, then for some term* $D$ *we have* $\Gamma, x = B : D \vdash_a A : C$*.*
7. *If* $\Gamma \vdash_a c(b_1, \ldots, b_n) : D$ *then there exist* $s$*,* $\Delta \equiv x_1 : B_1, \ldots, x_n : B_n$ *and* $A$ *such that* $D =_\beta A[x_j{:=}b_j]_{j=1}^n$*, and* $\Gamma \vdash_a b_i{:}B_i[x_j{:=}b_j]_{j=1}^{i-1}$*. Moreover,* $\Gamma \equiv \Gamma_1, c(\Delta) : A, \Gamma_2$ *and* $\Gamma_1, \Delta \vdash_a A : s$*. Finally, there are* $s_i \in S$ *such that* $\Gamma_1, \Delta_i \vdash_a B_i : s_i$ *and* $(s_i, s) \in \boldsymbol{P}$*.*

**Lemma 50 (Correctness of types for $\vdash_a$ and $\rightarrow_a$)**
*If* $\Gamma \vdash_a A : B$ *then* $(B \equiv \Box$ *or* $\Gamma \vdash_a B : s$ *for some sort* $s)$*.*

**Lemma 51 (Subject Reduction for $\vdash_a$ and $\rightarrow_a$)**
*If* $\Gamma \vdash_a A : B$ *and* $A \twoheadrightarrow_\beta A'$ *then* $\Gamma \vdash_a A' : B$*.*

Strong Normalisation can be established by translations into corresponding cubes which are shown to be strongly normalising.

**Theorem 52 (Strong Normalisation)** *If* $A$ *is* $\vdash_a$*-legal then* $SN_{\rightarrow_a}(A)$*.*

## 6 Conclusion

In this paper, we used a unique binder à la de Bruijn instead of the usual two binders $\lambda$ and $\Pi$. We studied the Barendregt cube written in this notation and established an isomorphism between the two versions of the cube. We then gave intantiation power to types similar to that of terms. Armed by the loss of important properties of the ordinary cube with type instantiation, and considering the solutions to these problems, we decided not to study the cube with unified binders and type instantiation, but instead to move to a larger extensions where other features like definitions, parameters and explicit substitutions are added. The interesting next step is to assess the implications of the collapse of both $\lambda$ and $\Pi$ into a unique binder. For example, does the unified binder give a well behaved cube with type instantiation alone or with explicit substitutions alone? Type instantiation in the ordinary cube faces the problem of loss of Subject reduction (cf. [18]). Explicit substitutions in the ordinary cube also faces the problem of loss of Subject reduction (cf. [4]).

## References


1. S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors. *Handbook of Logic in Computer Science, Volume 2.* Oxford University Press, 1992.
2. H.P. Barendregt. *The Lambda Calculus: its Syntax and Semantics.* Studies in Logic and the Foundations of Mathematics 103. North-Holland. 1984.

3. H.P. Barendregt. Lambda calculi with types. In [1], pages 117–309. Oxford University Press, 1992.
4. R. Bloo. *Preservation of Termination for Explicit Substitutions.* PhD thesis, Eindhoven University of Technology, 1997.
5. N.G. de Bruijn. The mathematical language AUTOMATH, its usage and some of its extensions. In M. Laudet, D. Lacombe, and M. Schuetzenberger, editors, *Symposium on Automatic Demonstration*, pages 29–61, IRIA, Versailles, 1968. Springer Verlag, Berlin, 1970. Lecture Notes in Mathematics **125**; also in [22], pages 73–100.
6. A. Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5:56–68, 1940.
7. T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.
8. G. Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens.* Nebert, Halle, 1879. Also in [12], pages 1–82.
9. G. Frege. *Funktion und Begriff, Vortrag gehalten in der Sitzung vom 9. Januar der Jenaischen Gesellschaft für Medicin und Naturwissenschaft.* Hermann Pohle, Jena, 1891. English translation in [21], pages 137–156.
10. J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur.* PhD thesis, Université Paris VII, 1972.
11. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *Proceedings Second Symposium on Logic in Computer Science*, pages 194–204, 1987.
12. J. van Heijenoort, editor. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931.* Harvard University Press, Cambridge, Massachusetts, 1967.
13. J.R. Hindley and J.P. Seldin. *Introduction to Combinators and $\lambda$-calculus*, volume 1 of *London Mathematical Society Student Texts.* Cambridge University Press, 1986.
14. W.A. Howard. The formulas-as-types notion of construction. In [25], pages 479–490, 1980.
15. F. Kamareddine, R. Bloo, and R. Nederpelt. On $\pi$-conversion in the $\lambda$-cube and the combination with abbreviations. *Annals of Pure and Applied Logic*, 97:27–45, 1999.
16. F. Kamareddine, T. Laan, and R. Nederpelt. Revisiting the notion of function. *Logic and Algebraic Programming*, to appear.
17. F. Kamareddine, T. Laan, and R. Nederpelt. Types in logic and mathematics before 1940. *Bulletin of Symbolic Logic*, 8(2):185–245, 2002.
18. F. Kamareddine and R.P. Nederpelt. Canonical typing and $\Pi$-conversion in the Barendregt Cube. *Journal of Functional Programming*, 6(2):245–267, 1996.
19. T. Laan. *The Evolution of Type Theory in Logic and Mathematics.* PhD thesis, Eindhoven University of Technology, 1997.
20. G. Longo and E. Moggi. Constructive natural deduction and its modest interpretation. Technical Report CMU-CS-88-131, Carnegie Mellono University, 1988.
21. B. McGuinness, editor. *Gottlob Frege: Collected Papers on Mathematics, Logic, and Philosophy.* Basil Blackwell, Oxford, 1984.
22. R.P. Nederpelt, J.H. Geuvers, and R.C. de Vrijer, editors. *Selected Papers on Automath.* Studies in Logic and the Foundations of Mathematics **133**. North-Holland, Amsterdam, 1994.
23. G.R. Renardel de Lavalette. Strictness analysis via abstract interpretation for recursively defined types. *Information and Computation*, 99:154–177, 1991.
24. J.C. Reynolds. *Towards a theory of type structure*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer, 1974.
25. J.P. Seldin and J.R. Hindley, editors. *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press, New York, 1980.