On Π-conversion in the λ-cube and the combination with abbreviations

Fairouz Kamareddine

University of Glasgow
Department of Computing Science
17 Lilybank Gardens
Glasgow G12 8QQ
Scotland
FAX: +44 141 330 4913
fairouz@dcs.gla.ac.uk

Roel Bloo and Rob Nederpelt

Mathematics and Computing Science
Eindhoven University of Technology
P.O.Box 513
5600 MB Eindhoven
The Netherlands
FAX: +31 40 2463992
{bloo, wsinrpn}@win.tue.nl

Abstract

Typed λ-calculus uses two abstraction symbols (λ and Π) which are usually treated in different ways: $\lambda_{x:*}.x$ has as type the abstraction $\Pi_{x:*}.*$, yet $\Pi_{x:*}.*$ has type $\Box$ rather than an abstraction; moreover, $(\lambda_{x:A}.B)C$ is allowed and β-reduction evaluates it, but $(\Pi_{x:A}.B)C$ is rarely allowed. Furthermore, there is a general consensus that λ and Π are different abstraction operators. While we agree with this general consensus, we find it nonetheless important to allow Π to *act* as an abstraction operator. Moreover, experience with AUTOMATH and the recent revivals of Π-reduction as in [KN 95b, PM 97], illustrate the elegance of giving Π-redexes a status similar to λ-redexes. However, Π-reduction in the λ-cube faces serious problems as shown in [KN 95b, PM 97]: it is not safe as regards subject reduction, it does not satisfy type correctness, it loses the property that the type of an expression is well-formed and it fails to make any expression that contains a Π-redex well-formed.

In this paper, we propose a solution to all those problems. The solution is to use a concept that is heavily present in most implementations of programming languages and theorem provers: abbreviations (viz. by means of a definition) or `let`-expressions. We will show that the λ-cube extended with Π-conversion and abbreviations satisfies all the desirable properties of the cube and does not face any of the serious problems of Π-reduction. We believe that this extension of the λ-cube is very useful: it gives a full formal study of two concepts (Π-reduction and abbreviations) that are useful for theorem proving and programming languages.

Keywords: Π-conversion, Abbreviations, Barendregt's cube, Subject Reduction, λ versus Π.

1 Introduction

Both Π-reduction and the use of names to abbreviate large expressions, are useful for automating mathematics, for theorem proving and for programming languages. Evidence of this is their presence in the various implementations of mathematics, theorem proving and programming languages. In what follows, we explicit the advantages and/or problems of these two concepts and we explain why combining them is even more useful.

1.1 On Π-reduction

Type theory has almost always been studied without Π-conversion (which is the analogue of β-conversion on product type level). That is, $\to_\beta$: $(\lambda_{x:A}.b)C \to_\beta b[x := C]$ is always assumed but not $\to_\Pi$: $(\Pi_{x:A}.B)C \to_\Pi B[x := C]$. The exceptions to this are: some AUTOMATH-languages (see [NGV 94]), the λ-cube extended with Π-reduction in [KN 95b] and the intermediate language in compilers for source languages as in [PM 97]. We claim that $\to_\Pi$ is desirable for the following reasons:

A. Π is, in a sense, a kind of λ. In higher order type theory, arrow-types of the form $A \to B$ are replaced by dependent products $\Pi_{x:A}.B$, where x may be free in B, and thus B *depends on* x. This means that abstraction can be over types: $\Pi_{x:A}.B$ as well as over terms: $\lambda_{x:A}.b$. But, once we allow abstraction over types, it would be nice to discuss the reduction rules which govern these types. In fact, Π is indeed a kind of λ as regards the abstraction over a variable and hence is eligible for an application.

B. Compatibility. Here are two important rules in the λ-cube:

$$\text{(abstraction rule)} \qquad \frac{\Gamma.\langle x : A\rangle \vdash b : B \qquad \Gamma \vdash \Pi_{x:A}.B : S}{\Gamma \vdash \lambda_{x:A}.b : \Pi_{x:A}.B}$$

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

The (abstraction rule) may be regarded as the compatibility property for typing with respect to abstraction. That is: $b : B$ implies $\lambda_{x:A}.b : \Pi_{x:A}.B$.

The compatibility property for the typing with respect to application is lost however. In fact, from the (application rule), one does not have: $F : \Pi_{x:A}.B$ implies $Fa : (\Pi_{x:A}.B)a$, but instead $F : \Pi_{x:A}.B$ implies $Fa : B[x := a]$.

To get compatibility for typing with respect to application, one needs to add $\to_\Pi$ and to change the (application rule) to:

$$\text{(new application rule)} \qquad \frac{\Gamma \vdash F : \Pi_{x:A}.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : (\Pi_{x:A}.B)a}$$

C. The AUTOMATH experience. One might argue that *implicit* Π-reduction (as is the case of the ordinary λ-cube with the (application rule) above) is closer to intuition in the most usual applications. However, experiences with the AUTOMATH-languages ([NGV 94]), containing *explicit* Π-reduction, demonstrated that there exists no formal or informal objection against the use of this explicit Π-reduction in natural applications of type systems.

D. Preference types, higher degrees, conversion. In [KN 95b], Π-reduction was shown to have various advantages which include the possibility of calculating the preference type $\tau(\Gamma, A)$ of a term A in a context Γ, the ability of incorporating different degrees of abstraction (rather than just two, λ and Π, as in the λ-cube) and the fact that the following rule of the λ-cube is superfluous:

$$\text{(conversion rule)} \qquad \frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : S \qquad B = B'}{\Gamma \vdash A : B'}$$

E. Programming languages. In programming language studies, a thriving area is that of the use of richly-typed intermediate languages in sophisticated compilers for higher-order, typed source languages ([PJ 96, SA 95, TMCSHL 96]). The recently developed language

[PM 97] aims at reducing the number of data types and the volume of code required in the compiler, by avoiding duplications. To do this, [PM 97] uses the whole λ-cube extended with Π-reduction and gives the following motivation:

- For reduction there is now only one set of rules $=_\beta$.
- With the old application rule, matters get very complicated when one adds further expressions (such as let and case).
- In a compiler, Π-reduction allows to separate the type finder from the evaluator since $\vdash$ no longer mentions substitution. One first extracts the type and only then evaluates it.

$A \ldots E$ above (especially B and D), can be viewed as *syntactic* motivations for Π-conversion. There are also *semantic* motivations for Π-conversion. Of these, we mention:

A. Infinite levels of abstractions One may abandon the two levels (λ and Π) as used in the current type systems and reformulate all type theory using different levels of λ's, where say, λ^0 is what we call λ, λ^1 is the Π, etc. Then, one will be able to give every term $\lambda^n_{x:A}.B$ the type $\lambda^{n+1}_{x:A}.C$ where C is a type of B, and the current type theory will be reproduced at each level. This would be rather interesting to investigate.

B. Finding and evaluating types become two separate events Π-reduction allows to separate the type finder from the evaluator since $\vdash$ no longer mentions substitution. This approach was presented in detail in [KN 95b] where the question $\Gamma \vdash A : B$ was split into $\Gamma \vdash A$ (A is typable in Γ) and $\tau(\Gamma, A) =_{\beta\Pi} B$ (B is convertible to the preference type of A).

All the above are reasons why it is interesting to study Π-conversion in the λ-cube. However, extending the λ-cube with Π-conversion is not a straightforward adding of $(\Pi_{x:A}.B)C \to_\Pi B[x := C]$ and of the (new application rule) (see [KN 95b]). Changing the (application rule) in the λ-cube to the (new application rule) as presented above results in the following problems:

1. **Correctness of types no longer holds.** With Π-reduction, one can have $\Gamma \vdash A : B$ without $B \equiv \Box$ or $\exists S\,.\,\Gamma \vdash B : S$. For example, $\langle z : *\rangle.\langle x : z\rangle \vdash (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$ yet $(\Pi_{y:z}.z)x \not\equiv \Box$ and $\forall S\,.\,\langle z : *\rangle.\langle x : z\rangle \not\vdash (\Pi_{y:z}.z)x : S$. The problem arises because of the new terms that contain Π-redexes (which did not exist in the λ-cube) and because [KN 95b] showed that in the λ-cube extended with Π-reduction:

   ($\uparrow$) If $\Gamma \vdash A : B$ then neither Γ nor A contain Π-redexes and if B contains a Π-redex, then B is itself that Π-redex.

2. **The system is no longer safe**. More precisely, subject reduction (SR) fails. That is, with Π-reduction and the (new application rule), $\Gamma \vdash A : B$ and $A \twoheadrightarrow A'$ may not imply $\Gamma \vdash A' : B$. For example, $\langle z : *\rangle.\langle x : z\rangle \vdash (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$ and $(\lambda_{y:z}.y)x \to x$, but one can't show $\langle z : *\rangle.\langle x : z\rangle \vdash x : (\Pi_{y:z}.z)x$. To show this last formula, one needs to use the above (conversion rule) and for this, one needs that $\exists S.\langle z : *\rangle.\langle x : z\rangle \vdash (\Pi_{y:z}.z)x : S$. But ($\uparrow$) in 1 above makes this impossible.

3. **The type of an expression may not be well-formed.** This is related to type correctness above. We say that A is well-formed if $A \equiv \Box$ or $\exists \Gamma, B\,.\,\Gamma \vdash A : B$. Now consider $\langle z : *\rangle.\langle x : z\rangle \vdash (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$. The type $(\Pi_{y:z}.z)x$ of $(\lambda_{y:z}.y)x$ is not well-formed by ($\uparrow$).

4. **Π-redexes are not well-formed.**

   From (↑) above follows that no expression that contains a Π-redex is well-formed.

Despite these shortcomings of Π-reduction, [PM 97] claims that its advantages are persuasive. In this paper we will repair the shortcomings of Π-reduction. It is amazing that the way to repair the problem is itself a very useful way in type theory: the use of abbreviations.

1.2 On abbreviations

In many type theories and lambda calculi, there is no formal possibility to use abbreviations, i.e., to introduce names for large expressions which can be used several times in a program or a proof. This possibility is essential for practical use, and indeed implementations of Pure Type Systems such as Coq [Dow 91], Lego [LP 92] and Nuprl [Con 86] do provide this possibility. Moreover, most implementations of programming languages (Haskell, ML, CAML, etc.) use names for large expressions via a well-known programming language concept: *let expressions*.

Example 1.1 $\texttt{Let}\ id : A \to A\ \texttt{be}\ (\lambda_{x:A}.x)\ \texttt{in}\ (\lambda_{y:A\to A}.id)id$ abbreviates the complex expression $(\lambda_{x:A}.x)$ as id in a more complex expression in which id occurs two times.

The intended meaning of "$\texttt{let}\ x : A\ \texttt{be}\ a\ \texttt{in}\ b$" is that a can be substituted for x in the expression b. In a sense, the expression $\texttt{let}\ x : A\ \texttt{be}\ a\ \texttt{in}\ b$ is similar to $(\lambda_{x:A}.b)a$ which β-reduces to $b[x := a]$, i.e., b with all free occurrences of x replaced by a. In the `let`-expression, however, it is not intended to necessarily replace all the occurrences of x in b by a. Nor is it intended that such a `let`-expression is a part of our term. Rather, the `let`-expression will live in the environment (or context) in which we evaluate or reason about the expression.

One of the advantages of the expression $\texttt{let}\ x : A\ \texttt{be}\ a\ \texttt{in}\ b$ over the redex $(\lambda_{x:A}.b)a$ is that it is convenient to have the freedom of substituting only some of the occurrences of an expression in a given formula. Another advantage is efficiency; one evaluates a in $\texttt{let}\ x : A\ \texttt{be}\ a\ \texttt{in}\ b$ only once, even in lazy languages.[1] A further advantage is that using x to be a in b can be used to type b efficiently, since the type A of a has to be calculated only once.[2] Furthermore, practical experiences with type systems show that `let`-expressions are absolutely indispensable for any realistic application. Without `let`-expressions, terms soon become forbiddingly complicated. By using `let`-expressions one can avoid such an explosion in complexity. This is, by the way, a very natural thing to do: the apparatus of mathematics, for instance, is unimaginable without a form of `let`-expressions (viz. definitions).

[1] Note that *smart* lazy languages will use explicit substitution or sharing techniques to evaluate a in $b[x := a]$ only once. Nevertheless, our extension with abbreviations as is considered here, is a straightforward extension of the λ-calculus and hence has less machinery than is involved with sharing or explicit substitution. Moreover, it may be that this simple concept of abbreviations as we introduce it can be used to formalise the notion of sharing. In fact, the work already done in [AFMOW95, BLR96] to show that some formal systems of explicit substitution or generalised reduction can formalise sharing, can be adapted to show that a formal system of abbreviations can also be a successful model for sharing.

[2] Here, in $b[x := a]$, the type A of a is calculated many times. Of course with the presence of Subject Reduction (SR), we we do not need to calculate the type of $b[x := a]$. Instead, we calculate the type of $(\lambda x : A.b)a$ (where the type of a is calculated only once) and we use subject reduction to derive the type of $b[x := a]$. However, many programming languages (PLs) do not have a clear notion of SR. Similarly, although in many PLs, sharing is used in order to calculate the type A of a only once in $b[x := a]$, there can be no escape from showing that this sharing technique of PLs is correct. Our system of definitions can be used to formalize this sharing technique of PLs.

There exist already two formal studies of `let`-expressions in the λ-cube [BKN 96, SP 93] where those `let`-expressions are called *definitions*. In this paper we differ from both accounts and use the simplest way of renaming large expressions and describe such renaming as *abbreviations*. We differ from [SP 93] in that we do not introduce new terms (`let`-terms) into our syntax and do not extend β-reduction to deal with those new terms. We differ from [BKN 96] in that we do not use nested definitions, which are needed for generalised reduction in [BKN 96] but not for Π-reduction.

We write $\langle x : A \rangle a$ to describe that x of type A, abbreviates a. We include abbreviations in contexts such that if an abbreviation occurs in a context then it can be used anywhere in the term we are reasoning about in that context.

In this paper, we will use abbreviations to repair all the problems of Π-reduction mentioned in Subsection 1.1. In particular, we extend the λ-cube with both Π-reduction and abbreviations and we show that this extension satisfies all the desirable properties (including Subject Reduction).

1.3 On combining abbreviations and Π-reductions

We shall show in this paper that the λ-cube extended with both abbreviations and Π-reduction ($\rightarrow_\Pi$ and the new application rule), preserves all its original properties (including safety, correctness of types and well-formedness of the type of an expression) and allows a non-limited occurrence of Π-redexes in the well-formed terms. This means that abbreviations (being important on their own) have repaired the problems of Π-reduction in the λ-cube. Let us here explain why the shortcomings of Π-reduction disappear with abbreviations:

Looking at the four problems of Π-reduction in Subsection 1.1, one sees that one needs to be able to type Π-redexes. This is not possible if the λ-cube is simply extended with the (new application rule) and $\rightarrow_\Pi$ as [KN 95b] showed. There are several routes to follow:

A. Infinite levels of abstractions as discussed under the semantic motivations for Π-conversion. This would be very interesting to investigate, but we feel it is a drastic change from current type theory and we are not sure what complications or contradictions will arise from different levels of λ's. We leave it as a point for future research.

B. Abbreviations. One may introduce (λ- and Π-) redexes as a separate (compound) term, which can be typed using abbreviations. In the type, the abbreviation is unfolded. The idea is simple: extend contexts with abbreviations and add the following rule:

$$\text{(abb rule)} \qquad \frac{\Gamma.\langle x : A \rangle B \vdash C : D}{\Gamma \vdash (\pi_{x:A}.C)B : D[x := B]} \text{ where } \pi \in \{\lambda, \Pi\}$$

This rule says that if $C : D$ can be typed using the abbreviation that x of type A is B, then $(\pi_{x:A}.C)B : D[x := B]$ can be typed without this abbreviation. This simple extension solves all the problems of Π-reduction mentioned in Subsection 1.1. Here is how:

1. **Correctness of types holds.** We demonstrate this with the example of problem 1 of Subsection 1.1. Recall that we have $\langle z : * \rangle.\langle x : z \rangle \vdash (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$ and want that for some S, $\langle z : * \rangle.\langle x : z \rangle \vdash (\Pi_{y:z}.z)x : S$. Here is how the latter formula now holds:

$$\begin{array}{ll} \langle z : * \rangle.\langle x : z \rangle \vdash z : * & \text{(start and weakening)} \\ \langle z : * \rangle.\langle x : z \rangle.\langle y : z \rangle x \vdash z : * & \text{(weakening)} \\ \langle z : * \rangle.\langle x : z \rangle \vdash (\Pi_{y:z}.z)x : *[y := x] \equiv * & \text{(abb rule)} \end{array}$$

2. **The system is now safe.** We demonstrate this with the example of problem 2 of Subsection 1.1. Recall that we have $\langle z:*\rangle.\langle x:z\rangle \vdash (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$ and $(\lambda_{y:z}.y)x \to_\beta x$ and we need to show that $\langle z:*\rangle.\langle x:z\rangle \vdash x : (\Pi_{y:z}.z)x$. Here is how the latter formula now holds:

$$\begin{array}{lll} a. & \langle z:*\rangle.\langle x:z\rangle \vdash x:z & \text{(start and weakening)} \\ b. & \langle z:*\rangle.\langle x:z\rangle \vdash (\Pi_{y:z}.z)x:* & \text{(from 1 above)} \\ & \langle z:*\rangle.\langle x:z\rangle \vdash x:(\Pi_{y:z}.z)x & \text{(conversion, } a, b, \text{ and } z =_\beta (\Pi_{y:z}.z)x) \end{array}$$

3. **The type of an expression is well-formed.** We demonstrate this with the example of problem 3 of Subsection 1.1. Recall that we have $\langle z:*\rangle.\langle x:z\rangle \vdash (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$ and we want to show that $(\Pi_{y:z}.z)x$ is typable (note that $(\Pi_{y:z}.z)x \not\equiv \Box$). By 1 above, we have that $\langle z:*\rangle.\langle x:z\rangle \vdash (\Pi_{y:z}.z)x : *$.

4. **Π-redexes are well-formed.** Π-redexes can now occur in contexts, terms, and types and all the obtained Π-redexes are indeed well-formed.

Remark 4.2, Lemma 4.12 and Theorem 4.13 will show that indeed all the problems of Π-reduction are solved. Intuitively, the reason is that abbreviations keep information in the context about the defined values of variables.

We divide the paper as follows:

1. In Section 2, we set up the machinery for both abbreviations and Π-reduction.

2. In Section 3, we introduce the original relation of the λ-cube $\vdash_\beta$ and the extended relation $\vdash_{\beta\Pi}$ as in [KN 95b]. We list the properties of both $\vdash_\beta$ and $\vdash_{\beta\Pi}$.

3. In Section 4, we introduce $\vdash_{ra}$ which is $\vdash_r$ (for $r = \beta$ or $\beta\Pi$) extended with abbreviations. We show that all the properties of the λ-cube remain valid for $\vdash_{ra}$. This establishes that extending the λ-cube with abbreviations or with both abbreviations and Π-reduction results in a well-behaved system. Due to the uniformity of our presentation, we prove almost all the results for $\vdash_{ra}$ rather than separately prove them for $\vdash_{\beta a}$ and $\vdash_{\beta\Pi a}$.

2 The Formal Machinery

The systems of the λ-cube (see [Ba 92]), are based on a set of *pseudo-expressions* or *terms* $\mathcal{T}$ defined by the following abstract syntax:

$$\mathcal{T} = * \mid \Box \mid V \mid \mathcal{T}\mathcal{T} \mid \pi_{V:\mathcal{T}}.\mathcal{T}$$

where $\pi \in \{\lambda, \Pi\}$, V is an infinite collection of variables ($\alpha, \beta, x, y, z, \ldots$ range over V), $*$ and $\Box$ are sorts ($S, S_1, S_2, \ldots$ range over $\{*, \Box\}$). We let $A, B, a, b \ldots$ range over $\mathcal{T}$.

Bound and free variables and substitution are defined as usual where the binding power of Π is similar to that of λ. We write $BV(A)$ and $FV(A)$ to represent the bound and free variables of A respectively. We write $A[x := B]$ to denote the term where all the free occurrences of x in A have been replaced by B. Furthermore, we take terms to be equivalent up to variable renaming and let $\equiv$ denote syntactic equality. For example, we take $\lambda_{x:A}.x \equiv \lambda_{y:A}.y$. We assume moreover, the Barendregt variable convention which is formally stated as follows:

Convention 2.1 *(BC: Barendregt's Convention)*
Names of bound variables will always be chosen such that they differ from the free ones in a term. Moreover, different abstraction operators have different variables as subscript. Hence, we will not have $(\pi_{x:A}.x)x$*, but* $(\pi_{y:A}.y)x$ *instead.*

We say that a rewrite relation r between terms is *compatible*, if for all terms A, B, C, it holds that $A\ r\ B$ implies $AC\ r\ BC$, $CA\ r\ CB$, $\pi_{x:A}.C\ r\ \pi_{x:B}.C$ and $\pi_{x:C}.A\ r\ \pi_{x:C}.B$. For a given set of rewrite rules r on terms, we define *the reduction relation* $\to_r$ as the compatible closure of r. Furthermore, for a reduction relation $\to_r$, we define $\twoheadrightarrow_r$ and $=_r$ to be its reflexive transitive closure and its equivalence closure respectively. We take $r \in \{\beta, \beta\Pi\}$ throughout and use the relations: $\to_\beta$ and $\to_{\beta\Pi}$ generated by: $(\lambda_{x:A}.B)C \to_\beta B[x := C]$, and $(\Pi_{x:A}.B)C \to_{\beta\Pi} B[x := C]$ respectively.

In the following definition, declarations are familiar from the λ-cube. The relation $\subseteq'$ enables 'information-preserving' extensions of the pseudocontext. We let $d, d_1, d_2, \ldots$ range over declarations and abbreviations and $\Gamma, \Delta, \Gamma', \Gamma_1, \Gamma_2, \ldots$ over pseudocontexts.

Definition 2.2 *(declarations, abbreviations, pseudocontexts,* $\subseteq'$*)*

1. *A* declaration *d is of the form $\langle x : A\rangle$. We define* $\mathtt{var}(d) \equiv x$ *and* $\mathtt{type}(d) \equiv A$.
2. *An* abbreviation *d is of the form $\langle x : A\rangle B$ and introduces x of type A to abbreviate B. We define* $\mathtt{var}(d)$, $\mathtt{type}(d)$ *and* $\mathtt{ab}(d)$ *to be x, A, and B respectively.*
3. *A* pseudocontext *Γ is a (possibly empty) concatenation of declarations and abbreviations $d_1.d_2.\cdots.d_n$ such that if $i \neq j$, then $\mathtt{var}(d_i) \not\equiv \mathtt{var}(d_j)$.*
4. *Define $dom(\Gamma) = \{\mathtt{var}(d) \mid d \in \Gamma\}$, $\Gamma\text{-}\mathtt{decl} = \{d \in \Gamma \mid d$ is a declaration $\}$ and $\Gamma\text{-}\mathtt{abb} = \{d \in \Gamma \mid d$ is an abbreviation $\}$. Note that $dom(\Gamma) = \{\mathtt{var}(d) \mid d \in \Gamma\text{-}\mathtt{decl} \cup \Gamma\text{-}\mathtt{abb}\}$.*
5. *Define $\subseteq'$ between pseudocontexts as the least reflexive transitive relation satisfying:*
   - $\Gamma.\Delta \subseteq' \Gamma.d.\Delta$ *for d a declaration or an abbreviation.*
   - $\Gamma.\langle x : A\rangle.\Delta \subseteq' \Gamma.\langle x : A\rangle B.\Delta$

In the rest of this section, we let $\vdash_r$ be a notion of derivability. The following is familiar (cf. [Ba 92]):

Definition 2.3 *Let Γ be a pseudocontext.*

1. *$A : B$ is called a* statement. *A and B are its* subject *and* predicate *respectively.*
2. *$\Gamma \vdash_r A : B$ is called a* judgement, *and $\Gamma \vdash_r A : B : C$ denotes $\Gamma \vdash_r A : B \wedge \Gamma \vdash_r B : C$.*
3. *Γ is called* legal *if $\exists A, B \in \mathcal{T}$ such that $\Gamma \vdash_r A : B$.*
4. *$A \in \mathcal{T}$ is called a Γ-*term *if $\exists B \in \mathcal{T}[\Gamma \vdash_r A : B \vee \Gamma \vdash_r B : A]$.*
5. *$A \in \mathcal{T}$ is called* legal *if $\exists \Gamma[A$ is a Γ-term$]$.*

The following is needed in the conversion rule where we replace $A =_r B$ by $\Gamma \vdash_r A =_{\mathtt{ab}} B$.

Definition 2.4 *(Abbreviational r-equality) For all pseudocontexts Γ we define the binary relation $\Gamma \vdash_r \cdot =_{\mathtt{ab}} \cdot$ to be the equivalence relation generated by*

- *if* $A =_r B$ *then* $\Gamma \vdash_r A =_{\mathtt{ab}} B$
- *if* $d \in \Gamma$-$\mathtt{abb}$ *and* $A, B \in \mathcal{T}$ *such that* B *arises from* A *by substituting one particular free occurrence of* $\mathtt{var}(d)$ *in* A *by* $\mathtt{ab}(d)$*, then* $\Gamma \vdash_r A =_{\mathtt{ab}} B$.

Remark 2.5 If no abbreviations are present in Γ then $\Gamma \vdash_r A =_{\mathtt{ab}} B$ is the same as $A =_r B$.

The following definition groups some preconditions of some typing rules. For example, instead of postulating for the start rule (in the case of a declaration) that $\Gamma \vdash \mathtt{type}(d) : S$ and $\mathtt{var}(d) \notin \Gamma$, we say $\Gamma \prec d$. This becomes particularly useful in the case of abbreviations.

Definition 2.6 *For* d *an abbreviation or declaration, we say* Γ *admits* d*, notation* $\Gamma \prec d$*, iff*

- $\Gamma.d$ *is a pseudocontext*
- $\Gamma \vdash_r \mathtt{type}(d) : S$ *for some sort* S.
- *if* d *is an abbreviation then* $\Gamma \vdash_r \mathtt{ab}(d) : \mathtt{type}(d)$

Finally, we extend the λ-cube notion $\Gamma \vdash d$ to deal with the case of d being either a declaration or an abbreviation.

Definition 2.7 *Let* Γ *be a pseudocontext. Let* $d, d_1, \ldots, d_n$ *be declarations and abbreviations. We define* $\Gamma \vdash_r d$ *and* $\Gamma \vdash_r d_1 \cdots d_n$ *as follows:*

- *If* d *is a declaration then* $\Gamma \vdash_r d$ *iff* $\Gamma \vdash_r \mathtt{var}(d) : \mathtt{type}(d)$*. Otherwise, if* d *is an abbreviation then* $\Gamma \vdash_r d$ *iff* $\Gamma \vdash_r \mathtt{var}(d) : \mathtt{type}(d) \wedge \Gamma \vdash_r \mathtt{ab}(d) : \mathtt{type}(d) \wedge \Gamma \vdash_r \mathtt{var}(d) =_{\mathtt{ab}} \mathtt{ab}(d)$.
- $\Gamma \vdash_r d_1 \cdots d_n$ *iff* $\Gamma \vdash_r d_i$ *for all* $1 \leq i \leq n$.

3 Extending the λ-cube with Π-reduction

First we introduce the λ-cube as presented in [Ba 92]. There the only declarations allowed are of the form $\langle x : A \rangle$, hence there are no abbreviations in the contexts. Thus, $\Gamma \prec d$ is of the form $\Gamma \prec \langle x : A \rangle$ and means that $\Gamma \vdash A : S$ for some S and that x is fresh in Γ, A. Moreover, recall that $\mathtt{var}(\langle x : A \rangle) \equiv x$ and $\mathtt{type}(\langle x : A \rangle) \equiv A$ and Π-reduction is not allowed.

Definition 3.1 *($\vdash_\beta$) Axioms and rules of the λ-cube; d is a declaration, $=_{\mathtt{ab}}$ is $=_\beta$:*

(axiom) $\quad \langle\rangle \vdash_\beta * : \Box$

(start rule) $\quad \dfrac{\Gamma \prec d}{\Gamma.d \vdash_\beta \mathtt{var}(d) : \mathtt{type}(d)}$

(weakening rule) $\quad \dfrac{\Gamma \prec d \qquad \Gamma \vdash_\beta D : E}{\Gamma.d \vdash_\beta D : E}$

(formation rule) $\quad \dfrac{\Gamma \vdash_\beta A : S_1 \qquad \Gamma.\langle x : A\rangle \vdash_\beta B : S_2}{\Gamma \vdash_\beta \Pi_{x:A}.B : S_2}$ *if* (S_1, S_2) *is a rule*

(abstraction rule) $\quad \dfrac{\Gamma.\langle x : A\rangle \vdash_\beta b : B \qquad \Gamma \vdash_\beta \Pi_{x:A}.B : S}{\Gamma \vdash_\beta \lambda_{x:A}.b : \Pi_{x:A}.B}$

(application rule) $\quad \dfrac{\Gamma \vdash_\beta F : \Pi_{x:A}.B \qquad \Gamma \vdash_\beta a : A}{\Gamma \vdash_\beta Fa : B[x := a]}$

(conversion rule) $\quad \dfrac{\Gamma \vdash_\beta A : B \qquad \Gamma \vdash_\beta B' : S \qquad \Gamma \vdash_\beta B =_{\mathtt{ab}} B'}{\Gamma \vdash_\beta A : B'}$

Each of the eight systems of the λ-cube is obtained by taking the (S_1, S_2) rules allowed from a subset of $\{(*,*), (*,\Box), (\Box,*), (\Box,\Box)\}$. These systems are given in the following table:

System	*Set of specific rules*			
$\lambda_\to$	$(*,*)$			
$\lambda 2$	$(*,*)$	$(\Box,*)$		
λP	$(*,*)$		$(*,\Box)$	
$\lambda P2$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	
$\lambda\underline{\omega}$	$(*,*)$			$(\Box,\Box)$
$\lambda\omega$	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$
$\lambda P\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$
$\lambda P\omega = \lambda C$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$

[KN 95b] extended this λ-cube by changing $\twoheadrightarrow_\beta$ to $\twoheadrightarrow_{\beta\Pi}$ and by changing $\vdash_\beta$ to $\vdash_{\beta\Pi}$ (note that $\Gamma \vdash_{\beta\Pi} B =_{\mathtt{ab}} B'$ is the same as $B =_{\beta\Pi} B'$, as there are no abbreviations):

Definition 3.2 *($\vdash_{\beta\Pi}$)*
$\vdash_{\beta\Pi}$ is $\vdash_\beta$ where β is replaced by $\beta\Pi$ throughout, and the application rule changes to:

(new application rule) $\quad \dfrac{\Gamma \vdash_{\beta\Pi} F : \Pi_{x:A}.B \qquad \Gamma \vdash_{\beta\Pi} a : A}{\Gamma \vdash_{\beta\Pi} Fa : (\Pi_{x:A}.B)a}$

Now we list some properties of $\vdash_\beta$ and $\vdash_{\beta\Pi}$ without proofs (see [KN 95b]). These properties (except of course the loss of type correctness, of SR and the non well-formedness of some types and of Π-redexes) will be established for the λ-cube extended with either abbreviations alone, or with both abbreviations and Π-reduction in Section 4.

Theorem 3.3 *(The Church Rosser Theorem CR, for $\twoheadrightarrow_r$, $r = \beta$ or $\beta\Pi$)*
If $A \twoheadrightarrow_r B$ and $A \twoheadrightarrow_r C$ then there exists D such that $B \twoheadrightarrow_r D$ and $C \twoheadrightarrow_r D$ $\Box$

Lemma 3.4 *(Start Lemma for $\vdash_r$ for $r = \beta$ or $\beta\Pi$)*
Let Γ be a $\vdash_r$-legal context. Then $\Gamma \vdash_r * : \Box$ and $\forall d \in \Gamma[\Gamma \vdash_r d]$. □

Lemma 3.5 *(Correctness of types for $\vdash_\beta$,* not *for $\vdash_{\beta\Pi}$)*
For $r = \beta$, but not *for $r = \beta\Pi$ we have:*
If $\Gamma \vdash_r A : B$ then ($B \equiv \Box$ or $\Gamma \vdash_r B : S$ for some sort S).
(cf. the counterexample of Subsection 1.1, Problem 1.) □

Lemma 3.6 *(Subject Reduction SR, for $\vdash_\beta$,* not *for $\vdash_{\beta\Pi}$)*
For $r = \beta$, but not *for $r = \beta\Pi$ we have:*
If $\Gamma \vdash_r A : B$ and $A \twoheadrightarrow_\beta A'$ then $\Gamma \vdash_r A' : B$
(cf. Subsection 1.1, Problem 2.) □

However, a weak form of SR holds for $\vdash_{\beta\Pi}$. First we need the following definition which removes the outermost Π-redex of a $\vdash_{\beta\Pi}$-legal term:

Definition 3.7 *For A $\vdash_{\beta\Pi}$-legal, let $\hat{A}$ be $C[x := D]$ if $A \equiv (\Pi_{x:B}.C)D$ and A otherwise.*

Lemma 3.8 *(Weak Subject Reduction for $\vdash_{\beta\Pi}$ and $\twoheadrightarrow_{\beta\Pi}$)*
If $\Gamma \vdash_{\beta\Pi} A : B$ and $A \twoheadrightarrow_{\beta\Pi} A'$, then $\Gamma \vdash_{\beta\Pi} A' : \hat{B}$ □

Lemma 3.9 *(Well-formedness of types for $\vdash_\beta$,* not *for $\vdash_{\beta\Pi}$)*
For $r = \beta$, but not *for $r = \beta\Pi$ we have:*
If $\Gamma \vdash_r A : B$ then $B \equiv \Box$ or $\exists C\,.\,\Gamma \vdash_r B : C$.
(cf. Subsection 1.1, Problem 3.) □

Lemma 3.10 *(Non-well-formedness of Π-redexes)*
For no Γ, A, B and C there is D such that $\Gamma \vdash_{\beta\Pi} (\Pi_{x:A}.B)C : D$.
(cf. Subsection 1.1, Problem 4.) □

Lemma 3.11 *(Uniqueness of Types for $\vdash_r$ and $\twoheadrightarrow_r$ for $r = \beta$ or $\beta\Pi$)*
If $\Gamma \vdash_r A : B_1$ and $\Gamma \vdash_r A : B_2$, then $B_1 =_r B_2$ □

Theorem 3.12 *(Strong Normalisation with respect to $\vdash_r$ and $\to_r$ for $r = \beta$ or $\beta\Pi$)*
If A is $\vdash_r$-legal then $SN_{\to_r}(A)$; i.e. A is strongly normalising with respect to $\to_r$. □

In the rest of the paper, we use the Π-cube to denote the λ-cube extended *with* Π-*reduction* and with the *new application rule*. We write π-cube for either the λ- or the Π-cube. Recall that $r \in \{\beta, \beta\Pi\}$ and $\pi \in \{\lambda, \Pi\}$.

4 Extending the π-cube with abbreviations

We shall extend the derivation rules of $\vdash_r$ so that we can use abbreviations in the context. The rules remain unchanged except for the following points:

- One rule, the (*abb rule*), is added.
- Not only declarations but also abbreviations are allowed in contexts.

- The use of $\Gamma \vdash B =_{\mathtt{ab}} B'$ in the conversion rule really has an effect, since $=_{\mathtt{ab}}$ is now a real extension of $=_r$ and Γ may contain abbreviations necessary to establish $B =_{\mathtt{ab}} B'$.

Note that the intended scope of $\langle x : A\rangle$ in $\Gamma.\langle x : A\rangle B.\Delta \vdash_r C : D$ is Δ, C and D. This is what should be expected since the scope of $\langle x : A\rangle$ in $\Gamma.\langle x : A\rangle.\Delta \vdash_r C : D$ is the same.

Definition 4.1 *(Axioms and rules of the λ-cube extended with abbreviations; d ranges over declarations and abbreviations. Recall that r is either β or $\beta\Pi$; in the system where $r = \beta$, we have $\pi = \lambda$; if $r = \beta\Pi$, π can be λ or Π.)*

We extend the relation $\vdash_r$ to $\vdash_{ra}$ by adding the following abbreviation rule:

$$\textit{(abb rule)} \qquad \frac{\Gamma.\langle x : A\rangle B \vdash_{ra} C : D}{\Gamma \vdash_{ra} (\pi_{x:A}.C)B : D[x := B]}$$

The (*abb rule*) says that if $C : D$ can be deduced using an abbreviation $d \equiv \langle x : A\rangle B$, then $(\pi_{x:A}.C)B$ will be of type D where d has been unfolded in D.

Remark 4.2 *(Well-Formedness of Π-redexes for $\vdash_{\beta\Pi a}$)*

From the (abb rule) and the (new application rule), if $\Gamma \vdash_{\beta\Pi a} A : B$ then both A and B may contain Π-redexes.

Remark 4.3 When considering an abbreviation in a term to be equivalent to a redex, the (abb rule) is quite natural: for instance, deriving a type for $(\lambda_{x:\alpha}.x)y$ via abbreviating y to be x gives the same type as the derivation via abstraction followed by ordinary application (let $\Gamma \equiv \langle \alpha : *\rangle.\langle y : \alpha\rangle$):

$$\text{(abb rule)} \quad \frac{\Gamma.\langle x : \alpha\rangle y \vdash_{ra} x : \alpha}{\Gamma \vdash_{ra} (\lambda_{x:\alpha}.x)y : \alpha[x := y]}$$

$$\text{(appl)} \quad \frac{\text{(abstr)} \quad \dfrac{\Gamma.\langle x : \alpha\rangle \vdash_{ra} x : \alpha \qquad \Gamma \vdash_{ra} (\Pi_{x:\alpha}.\alpha) : *}{\Gamma \vdash_{ra} \lambda_{x:\alpha}.x : \Pi_{x:\alpha}.\alpha} \quad \Gamma \vdash_{ra} y : \alpha}{\Gamma \vdash_{ra} (\lambda_{x:\alpha}.x)y : \alpha[x := y]}$$

Let us now give an example which shows why abbreviations are useful:

Example 4.4 $\langle \beta : *\rangle.\langle y : \beta\rangle \not\vdash_r (\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)y)\beta : \beta$. We need $y : \alpha$ to be able to type $(\lambda_{x:\alpha}.x)y$. Looking carefully however, we find that $(\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)y)\beta$ is abbreviating β by α. So here is how the above derivation can be obtained using abbreviations (we present a short-cut and do not mention all the steps, nor the names of the rules):

$$\begin{array}{l}
\langle \beta : *\rangle.\langle y : \beta\rangle.\langle \alpha : *\rangle\beta \vdash_{ra} \lambda_{x:\alpha}.x : \Pi_{x:\alpha}.\alpha \\
\langle \beta : *\rangle.\langle y : \beta\rangle.\langle \alpha : *\rangle\beta \vdash_{ra} y : \beta \\
\langle \beta : *\rangle.\langle y : \beta\rangle.\langle \alpha : *\rangle\beta \vdash_{ra} \alpha =_{\mathtt{ab}} \beta \\
\langle \beta : *\rangle.\langle y : \beta\rangle.\langle \alpha : *\rangle\beta \vdash_{ra} y : \alpha \\
\langle \beta : *\rangle.\langle y : \beta\rangle.\langle \alpha : *\rangle\beta \vdash_{ra} (\lambda_{x:\alpha}.x)y : \alpha \\
\langle \beta : *\rangle.\langle y : \beta\rangle \vdash_{ra} (\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)y)\beta : \alpha[\alpha := \beta] \\
\langle \beta : *\rangle.\langle y : \beta\rangle \vdash_{ra} (\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)y)\beta : \beta
\end{array}$$

Remark 4.5 In [BKN 96], we introduced a notion of generalised definitions which, like abbreviations, bind a name to a complex expression. In [BKN 96], a generalisation of β-reduction was inspired by a special notation (see [KN 95a]). With that generalisation of reduction (which may contract some redex r before other redexes upon which this r depends have been

contracted), definitions had to be nested to mirror this generalised reduction. Such nesting is unnecessary for the reductions we are using in the present paper.

We now study ordinary (non-nested) definitions combined with Π-reduction. We note that any abbreviation in the sense of the present paper is also a definition in the sense of [BKN 96] when the notation is changed. Furthermore any type derivation with abbreviations in this paper (not involving Π-reduction) is also a type derivation with definitions in [BKN 96]. That is, if $\Gamma \vdash_{\beta a} A : B$ then $\mathcal{I}(\Gamma) \vdash_e \mathcal{I}(A) : \mathcal{I}(B)$ where $\vdash_e$ is the type derivation of [BKN 96] and $\mathcal{I}$ translates terms to the notation of [KN 95a].

Now, we go through the usual properties of the λ-cube showing that they hold for $\vdash_{ra}$.

Lemma 4.6 *(Free variable Lemma for $\vdash_{ra}$)*
Let Γ be a legal context such that $\Gamma \vdash_{ra} B : C$. Then the following holds:

1. *If d and d' are two different elements of Γ-$\mathtt{decl} \cup \Gamma$-$\mathtt{abb}$, then $\mathtt{var}(d) \not\equiv \mathtt{var}(d')$.*
2. *$FV(B), FV(C) \subseteq dom(\Gamma)$.*
3. *If $\Gamma \equiv \Gamma_1.d.\Gamma_2$ then $FV(d) \subseteq dom(\Gamma_1)$.*

Proof: *All by induction on the derivation of $\Gamma \vdash_{ra} B : C$.* □

Lemma 4.7 *(Start Lemma for $\vdash_{ra}$)*
If Γ is legal, then $\Gamma \vdash_{ra} * : \Box$ and $\forall d \in \Gamma[\Gamma \vdash_{ra} d]$.

Proof: *If Γ is legal then for some terms B, C: $\Gamma \vdash_{ra} B : C$; now use induction on the derivation of $\Gamma \vdash_{ra} B : C$.* □

Lemma 4.8 *(Transitivity Lemma for $\vdash_{ra}$)*
Let Γ and Δ be legal contexts such that $\Gamma \vdash \Delta$.

1. *If $\Delta \vdash_{ra} A =_{\mathtt{ab}} B$ then $\Gamma \vdash_{ra} A =_{\mathtt{ab}} B$.*
2. *If $\Delta \vdash_{ra} A : B$ then $\Gamma \vdash_{ra} A : B$.*

Proof:

1. *For all $d \in \Delta$, $\Gamma \vdash_{ra} \mathtt{var}(d) =_{\mathtt{ab}} \mathtt{ab}(d)$. If A' results from A by replacing one free occurrence of $\mathtt{var}(d)$ in A by $\mathtt{ab}(d)$, then $\Gamma \vdash_{ra} A =_{\mathtt{ab}} A'$ by repeating the process of proving $\Gamma \vdash_{ra} \mathtt{var}(d) =_{\mathtt{ab}} \mathtt{ab}(d)$ on the particular occurrence of $\mathtt{var}(d)$ in A.*
2. *Induction on the derivation $\Delta \vdash_{ra} A : B$, using 1 in the case of the conversion rule.* □

Note in the following lemmas how definitions behave well in thinning and substitution.

Lemma 4.9 *(Thinning Lemma for $\vdash_{ra}$)*

1. *If $\Gamma_1.\Gamma_2 \vdash_{ra} A =_{\mathtt{ab}} B$, $\Gamma_1.\Delta.\Gamma_2$ is a legal context, then $\Gamma_1.\Delta.\Gamma_2 \vdash_{ra} A =_{\mathtt{ab}} B$.*
2. *If Γ and Δ are legal contexts such that $\Gamma \subseteq' \Delta$ and if $\Gamma \vdash_{ra} A : B$, then $\Delta \vdash_{ra} A : B$.*

Proof: *1. is by induction on the derivation $\Gamma_1.\Gamma_2 \vdash_{ra} A =_{\mathtt{ab}} B$. 2. is as follows:*

- *If $\Gamma.\Delta \vdash_{ra} A : B$, $\Gamma \vdash_{ra} C : S$, x is fresh, then also $\Gamma.\langle x : C\rangle.\Delta \vdash_{ra} A : B$. We show this by induction on the derivation $\Gamma.\Delta \vdash_{ra} A : B$ using 1. for conversion.*

- *If* $\Gamma.\Delta \vdash_{ra} A : B$, $\Gamma \vdash_{ra} C : D : S$, *$x$ is fresh, then also* $\Gamma.\langle x : D\rangle C.\Delta \vdash_{ra} A : B$. *We show this by induction on the derivation* $\Gamma.\Delta \vdash_{ra} A : B$.

- *If* $\Gamma.\langle x : A\rangle.\Delta \vdash_{ra} B : C$, $\Gamma \vdash_{ra} D : A$, *then* $\Gamma.\langle x : A\rangle D.\Delta \vdash_{ra} B : C$ *is shown by induction on the derivation* $\Gamma.\langle x : A\rangle.\Delta \vdash_{ra} B : C$ *(for conversion, use 1.; note that* $\Gamma.\langle x : A\rangle.\Delta \vdash_{ra} B_1 =_{\mathtt{ab}} B_2$ *is equivalent to* $\Gamma.\Delta \vdash_{ra} B_1 =_{\mathtt{ab}} B_2$*).* □

Lemma 4.10 *(Substitution Lemma for* $\vdash_{ra}$*)*
Let $d = \langle x : C\rangle D$, $\Delta_d = \Delta[x := D]$, $A_d = A[x := D]$ *and* $B_d = B[x := D]$.

1. *If* $\Gamma.d.\Delta \vdash_{ra} A =_{\mathtt{ab}} B$, *$A$ and B are* $\Gamma.d.\Delta$*-legal, then* $\Gamma.\Delta_d \vdash_{ra} A_d =_{\mathtt{ab}} B_d$.

2. *If B is a* $\Gamma.d$*-legal term, then* $\Gamma.d \vdash_{ra} B =_{\mathtt{ab}} B_d$.

3. *If* $\Gamma.d.\Delta \vdash_{ra} A : B$, *then* $\Gamma.\Delta_d \vdash_{ra} A_d : B_d$.

4. *If* $\Gamma.\langle x : C\rangle.\Delta \vdash_{ra} A : B$ *and* $\Gamma \vdash_{ra} D : C$, *then* $\Gamma.\Delta_d \vdash_{ra} A_d : B_d$.

Proof: *1. Induction on the generation of* $=_{\mathtt{ab}}$. *2. Induction on the structure of B. 3. Induction on the derivation rules, using 1., 2. and thinning. 4. Idem.* □

Lemma 4.11 *(Generation Lemma for* $\vdash_{ra}$*)*

1. *If* $\Gamma \vdash_{ra} S : C$ *then* $S \equiv *$ *and* $\Gamma \vdash_{ra} C =_{\mathtt{ab}} \Box$, *furthermore if* $C \not\equiv \Box$ *then* $\Gamma \vdash_{ra} C : S'$ *for some sort* S'.

2. *If* $\Gamma \vdash_{ra} x : A$ *then for some* $d \in \Gamma$, $x \equiv \mathtt{var}(d)$, $\Gamma \vdash_{ra} A =_{\mathtt{ab}} \mathtt{type}(d)$ *and* $\Gamma \vdash_{ra} A : S$ *for some sort S.*

3. *If* $\Gamma \vdash_{ra} \lambda_{x:A}.B : C$ *then for some D and sort S:* $\Gamma.\langle x : A\rangle \vdash_{ra} B : D$, $\Gamma \vdash_{ra} \Pi_{x:A}.D : S$, $\Gamma \vdash_{ra} \Pi_{x:A}.D =_{\mathtt{ab}} C$ *and if* $\Pi_{x:A}.D \not\equiv C$ *then* $\Gamma \vdash_{ra} C : S'$ *for some sort* S'.

4. *If* $\Gamma \vdash_{ra} \Pi_{x:A}.B : C$ *then for some sorts* S_1, S_2*:* $\Gamma \vdash_{ra} A : S_1$, $\Gamma.\langle x : A\rangle \vdash_{ra} B : S_2$, (S_1, S_2) *is a rule,* $\Gamma \vdash_{ra} C =_{\mathtt{ab}} S_2$ *and if* $S_2 \not\equiv C$ *then* $\Gamma \vdash_{ra} C : S$ *for some sort S.*

5. *If* $\Gamma \vdash_{ra} Fa : C$, $F \not\equiv \pi_{x:A}.B$, *then for some* D, E*:* $\Gamma \vdash_{ra} a : D$, $\Gamma \vdash_{ra} F : \Pi_{x:D}.E$, $\Gamma \vdash_{ra} T =_{\mathtt{ab}} C$ *and if* $T \not\equiv C$ *then* $\Gamma \vdash_{ra} C : S$ *for some S, where* $T \equiv (\Pi_{x:D}.E)a$ *if* $r = \beta\Pi$ *and* $T \equiv E[x := a]$ *if* $r = \beta$.

6. *If* $\Gamma \vdash_{ra} (\pi_{x:A}.D)B : C$, *then* $\Gamma.\langle x : A\rangle B \vdash_{ra} D : C$

Proof: *1., 2., 3., 4. and 5. follow by induction on the derivations (use Thinning). As to 6., an easy induction on the derivation rules shows that one of the following holds:*

- $\Gamma.\langle x : A\rangle B \vdash_{ra} D : E, \Gamma \vdash_{ra} E[x := B] =_{\mathtt{ab}} C$ *and* $E[x := B] \not\equiv C \Rightarrow \exists S.\Gamma \vdash_{ra} C : S$.

- $\Gamma \vdash_{ra} B : F$, $\Gamma \vdash_{ra} \lambda_{x:A}.D : \Pi_{y:F}.G$, $\Gamma \vdash_{ra} C =_{\mathtt{ab}} T$ *and if* $T \not\equiv C$ *where* $T \equiv (\Pi_{y:F}.G)B$ *if* $r = \beta\Pi$ *and* $T \equiv G[y := B]$ *if* $r = \beta$, *then* $\Gamma \vdash_{ra} C : S$ *for some sort S.*

In both cases use thinning and conversion; in the second case use also 3. □

Now, recall that correctness of types fails for $\vdash_{\beta\Pi}$ but holds for $\vdash_{\beta}$. Here we show it for $\vdash_{ra}$.

Lemma 4.12 *(Correctness (and hence well-formedness) of Types for $\vdash_{ra}$)*
If $\Gamma \vdash_{ra} A : B$ then $B \equiv \Box$ or $\Gamma \vdash_{ra} B : S$ for some sort S.
Proof: *By induction on the derivation rules. The interesting cases are:*

- *Abbreviation: If $\Gamma \vdash_{ra} (\pi_{x:A}.D)B : C[x := B]$ where $\Gamma.\langle x : A\rangle B \vdash_{ra} D : C$, then by IH, $C \equiv \Box$ or $\exists S, \Gamma.\langle x : A\rangle B \vdash_{ra} C : S$. If $C \equiv \Box$ then $C[x := B] \equiv \Box$; else, by Substitution Lemma $\Gamma \vdash_{ra} C[x := B] : S[x := B] \equiv S$.*

- *Application: If $\Gamma \vdash_{\beta\Pi a} Fa : (\Pi_{x:A}.B)a$ where $\Gamma \vdash_{\beta\Pi a} F : \Pi_{x:A}.B$ and $\Gamma \vdash_{\beta\Pi a} a : A$, then by IH, $\exists S, \Gamma \vdash_{\beta\Pi a} \Pi_{x:A}.B : S$. By Generation $\Gamma.\langle x : A\rangle \vdash_{\beta\Pi a} B : S$. By Thinning $\Gamma.\langle x : A\rangle a \vdash_{\beta\Pi a} B : S$ and by the (abb rule) $\Gamma \vdash_{\beta\Pi a} (\Pi_{x:A}.B)a : S[x := a] \equiv S$.* □

From correctness of types for $\vdash_{ra}$, we can establish its subject reduction.

Theorem 4.13 *(Subject Reduction for $\vdash_{ra}$ and $\twoheadrightarrow_r$)*
If $\Gamma \vdash_{ra} A : B$ and $A \twoheadrightarrow_r A'$ then $\Gamma \vdash_{ra} A' : B$.
Proof: *We prove by simultaneous induction on the derivation rules:*

1. *If $\Gamma \vdash_{ra} A : B$ and Γ' results from contracting one of the terms in the declarations and abbreviations of Γ by a one step r-reduction, then $\Gamma' \vdash_{ra} A : B$*

2. *If $\Gamma \vdash_{ra} A : B$ and $A \to_r A'$ then $\Gamma \vdash_{ra} A' : B$*

We will only treat the case $r = \beta\Pi$. If the derivation rule is (axiom): easy. If it is (start rule): we consider the case $d \equiv \langle x : A\rangle B$, $A \to_{\beta\Pi} A'$. The other cases are similar or easy. We have: $\Gamma.\langle x : A\rangle B \vdash_{\beta\Pi a} x : A$ where $\Gamma \prec \langle x : A\rangle B$, i.e. $\Gamma \vdash_{\beta\Pi a} B : A : S$. By IH, $\Gamma \vdash_{\beta\Pi a} A' : S$. By conversion $\Gamma \vdash_{\beta\Pi a} B : A'$. Hence $\Gamma.\langle x : A'\rangle B \vdash_{\beta\Pi a} x : A'$ and again by conversion $\Gamma.\langle x : A'\rangle B \vdash_{\beta\Pi a} x : A$.
If the derivation rule is (weak), (formation), (conversion) or (abstraction): use IH (and conversion for abstraction). Now we treat the rest:

- *(abbreviation): $\Gamma \vdash_{\beta\Pi a} (\pi_{x:A}.D)B : C[x := B]$ where $\Gamma.\langle x : A\rangle B \vdash_{\beta\Pi a} D : C$. Now $\Gamma' \vdash_{\beta\Pi a} (\pi_{x:A}.D)B : C[x := B]$, $\Gamma \vdash_{\beta\Pi a} (\pi_{x:A}.D')B : C[x := B]$ and $\Gamma \vdash_{\beta\Pi a} (\pi_{x:A'}.D)B : C[x := B]$ by IH. Furthermore, if $B \to_{\beta\Pi} B'$ then $\Gamma \vdash_{\beta\Pi a} C[x := B] =_{\mathtt{ab}} C[x := B']$ and by IH and the (abb rule) we get $\Gamma \vdash_{\beta\Pi a} (\pi_{x:A}.D)B' : C[x := B']$. Now by Lemma 4.12, applied to the judgement $\Gamma.\langle x : A\rangle B \vdash_{\beta\Pi a} D : C$, we get: $C \equiv \Box$ or $\exists S, \Gamma.\langle x : A\rangle B \vdash_{\beta\Pi a} C : S$. If $C \equiv \Box$ then $C[x := B] \equiv C \equiv C[x := B']$. Else, by the Substitution Lemma $\Gamma \vdash_{\beta\Pi a} C[x := B] : S[x := B] \equiv S$, so by conversion $\Gamma \vdash_{\beta\Pi a} (\pi_{x:A}.D)B' : C[x := B]$.*

  For the last possibility, $(\pi_{x:A}.D)B \to_{\beta\Pi} D[x := B]$, we remark that by the Substitution Lemma, $\Gamma.\langle x : A\rangle B \vdash_{\beta\Pi a} D : C$ leads to $\Gamma \vdash_{\beta\Pi a} D[x := B] : C[x := B]$.

- *(application): $\Gamma \vdash_{\beta\Pi a} Fa : (\Pi_{x:A}.B)a$ where $\Gamma \vdash_{\beta\Pi a} F : \Pi_{x:A}.B$ and $\Gamma \vdash_{\beta\Pi a} a : A$. Then $\Gamma' \vdash_{\beta\Pi a} Fa : (\Pi_{x:A}.B)a$ and $\Gamma \vdash_{\beta\Pi a} F'a : (\Pi_{x:A}.B)a$ by IH, and $\Gamma \vdash_{\beta\Pi a} Fa' : (\Pi_{x:A}.B)a$ because by IH $\Gamma \vdash_{\beta\Pi a} Fa' : (\Pi_{x:A}.B)a'$, by Lemma 4.12 $\exists S, \Gamma \vdash_{\beta\Pi a} (\Pi_{x:A}.B)a : S$, so by conversion $\Gamma \vdash_{\beta\Pi a} Fa' : (\Pi_{x:A}.B)a$.*

  Now the crucial case: $F \equiv (\pi_{y:C}.D)$, $Fa \to_{\beta\Pi} D[y := a]$. Then $\Gamma \vdash_{\beta\Pi a} (\pi_{y:C}.D)a : (\Pi_{x:A}.B)a$ so by Generation $\Gamma.\langle y : C\rangle a \vdash_{\beta\Pi a} D : (\Pi_{x:A}.B)a$, now by Substitution $\Gamma \vdash_{\beta\Pi a} D[y := a] : ((\Pi_{x:A}.B)a)[y := a]$, but by BC $((\Pi_{x:A}.B)a)[y := a] \equiv (\Pi_{x:A}.B)a$. □

The proof of strong normalisation (SN) is based on SN of the λ-cube extended with abbreviations as in [BKN 96]. $\mathrm{SN}_{\to_r}(A)$ denotes that A is strongly normalising with respect to $\to_r$.

Theorem 4.14 *(Strong Normalisation for the λ-cube with respect to $\vdash_{\beta a}$ and $\to_\beta$)*
If A is a $\vdash_{\beta a}$-legal term then $SN_{\to_\beta}(A)$.

Proof: *By Remark 4.5, $\vdash_{\beta a}$ is a subset of $\vdash_e$ of [BKN 96] in that if $\Gamma \vdash_{\beta a} A : B$ then $\Gamma \vdash_e A : B$ (abstracting from the different notation). Now, SN for $\vdash_{\beta a}$ follows from that of $\vdash_e$ (see [BKN 96] for the lengthy but standard proof (similar to that of [Geu 95]) of SN for $\vdash_e$ which can be adapted to $\vdash_{\beta a}$).* □

SN of $\vdash_{\beta\Pi a}$ is a consequence of that of $\vdash_{\beta a}$. First we change Π-redexes into λ-redexes.[3]

Definition 4.15

- *For all pseudo-expressions A we define $\widetilde{A}$ to be the term A where in all Π-redexes the Π-symbol has been changed into a λ-symbol, creating a λ-redex instead.*
- *For a context $\Gamma \equiv d_1.\cdots.d_n$ we define $\widetilde{\Gamma}$ to be $\widetilde{d_1}.\cdots.\widetilde{d_n}$, where $\langle \widetilde{x : A}\rangle \equiv \langle x : \widetilde{A}\rangle$ and $\langle \widetilde{x : A}\rangle B \equiv \langle x : \widetilde{A}\rangle \widetilde{B}$.*

Lemma 4.16 *If $\Gamma \vdash_{\beta\Pi a} A : B$ then $\widetilde{\Gamma} \vdash_{\beta a} \widetilde{A} : \widetilde{B}$.*

Proof: *Induction on the derivation rules of $\vdash_{\beta\Pi a}$. All rules except the* (new application rule) *are trivial since they are also rules in $\vdash_{\beta a}$.*

If $\Gamma \vdash_{\beta\Pi a} Fa : (\Pi_{x:A}.B)a$ results from $\Gamma \vdash_{\beta\Pi a} F : \Pi_{x:A}.B$ and $\Gamma \vdash_{\beta\Pi a} a : A$. Then by IH $\widetilde{\Gamma} \vdash_{\beta a} \widetilde{F} : \Pi_{x:\widetilde{A}}.\widetilde{B}$ and $\widetilde{\Gamma} \vdash_{\beta a} \widetilde{a} : \widetilde{A}$, so by application of $\vdash_{\beta a}$, $\widetilde{\Gamma} \vdash_{\beta a} \widetilde{F}\widetilde{a} : \widetilde{B}[x := \widetilde{a}]$.

As $\widetilde{\Gamma} \vdash_{\beta a} \widetilde{F} : \Pi_{x:\widetilde{A}}.\widetilde{B}$, we also get $\widetilde{\Gamma}.\langle x : \widetilde{A}\rangle \vdash_{\beta a} \widetilde{B} : S$ and hence by thinning and the (abb rule) for $\vdash_{\beta a}$, $\widetilde{\Gamma} \vdash_{\beta a} (\lambda_{x:\widetilde{A}}.\widetilde{B})\widetilde{a} : S$, so by conversion $\widetilde{\Gamma} \vdash_{\beta a} \widetilde{F}\widetilde{a} : (\lambda_{x:\widetilde{A}}.\widetilde{B})\widetilde{a}$.

It remains to be shown that $\widetilde{F}\widetilde{a} \equiv \widetilde{Fa}$, i.e., F is not a Π-term. Suppose towards a contradiction that F is a Π-term, then by generation on $\Gamma \vdash_{\beta\Pi a} F : \Pi_{x:A}.B$, $\Gamma \vdash_{\beta\Pi a} \Pi_{x:A}.B =_{\mathtt{ab}} S$ for some sort S. This is clearly a contradiction. □

Theorem 4.17 *(Strong Normalisation for the λ-cube with respect to $\vdash_{\beta\Pi a}$ and $\to_{\beta\Pi}$)*
If A is a $\vdash_{\beta\Pi a}$-legal term then $SN_{\to_{\beta\Pi}}(A)$.

Proof: *If A is $\vdash_{\beta\Pi a}$-legal then $\widetilde{A}$ is $\vdash_{\beta a}$-legal by Lemma 4.16 and hence $SN_{\to_\beta}(\widetilde{A})$ (Theorem 4.14). Due to Subject Reduction of $\vdash_{\beta a}$, no Π-redexes can be created in the course of $\to_\beta$-reduction of $\widetilde{A}$, therefore $SN_{\to_\beta}(\widetilde{A})$ implies $SN_{\to_{\beta\Pi}}(A)$.* □

See [KN 95b, BKN 96] for other properties of the λ-cube with Π-reduction or abbreviations.

5 Conclusion

In type theory, abstraction is done via both λ and Π and one writes either $\lambda_{x:A}.B$ or $\Pi_{x:A}.B$. Reduction, however, is usually restricted to λ-redexes. Hence, one evaluates $(\lambda_{x:A}.B)C$ to $B[x := C]$, but usually one does not allow $(\Pi_{x:A}.B)C$ as a term which can be evaluated to

[3] Note that in general this requires the level of the type system in the cube (i.e. abstractions that are allowed) to be raised.

$B[x := C]$. An exception to this is the AUTOMATH notation where the distinction between λ and Π is absent and one writes $[x : A]B$ to express either $\lambda_{x:A}.B$ or $\Pi_{x:A}.B$. In all type systems however (including AUTOMATH and the system of this paper), there *is* a distinction between λ and Π. The various accounts differ in how large such a distinction is.

We believe that λ and Π can be treated similarly to a great extent, by the incorporation of Π-reduction. As we have seen, applications of type systems use Π-reduction. AUTOMATH did introduce Π-reduction, but its formal properties were only established for the first time in [KN 95b] and later the problems were reconsidered in [PM 97].

In this paper, we made Π behave more like an abstraction operator and gave a Π-abstraction the right to be applied to another term. We did not however allow a Π-abstraction to be typed by another abstraction. Otherwise, one will need more abstraction operators than λ and Π and this naturally leads to an infinite level of abstraction operators as discussed in Subsection 1.3. Our choice has been to express the type of a Π-term as a simple sort given by the formation rule and to type a Π-redex using abbreviations. To do this, we typed $(\Pi_{x:A}.C)B$ in context Γ by typing C in context $\Gamma.\langle x : A\rangle B$. Our addition of abbreviations (which differ in this paper from the existing notions of definitions in the literature), is simple and worth studying. Furthermore, this addition enabled us to solve the problems of the Π-cube that were left open in [KN 95b].

There are many arguments why Π-reduction and abbreviations must be considered and why a system combining both, without losing any of the nice properties of the λ-cube, is certainly worth considering. Moreover, we find it intriguing that so far in the literature, abbreviations have been added for reasons of efficiency of implementation and not because they solve theoretical problems. In this paper, we have shown that abbreviations solve the problems of the λ-cube extended with Π-reduction. In [BKN 96], we have shown that definitions solve the problem of subject reduction in the λ-cube extended with a notion of *generalised reduction*. The reason why abbreviations solve these problems is that they keep information in the context about the defined values of some variables, thus preventing that this information gets lost in a reduction process.

Hence, our paper contributes to other work on definitions not only in that it offers a simple and attractive account of definitions or abbreviations which keeps all the original properties of the λ-cube, but also because it shows that abbreviations are theoretically important and should hence be introduced in the λ-cube. Figure 1 on page 17 summarizes our results in this paper where we use the following notational conventions: CR, SN, SR and TC stand for Church Rosser, strong normalisation, subject reduction and type correctness respectively; WF stands for well-formedness of the type of an expression, π^{abb}-cube is the π-cube extended with abbreviations for $\pi = \lambda$ or Π.

A question may now occur to the reader: "If our paper is concerned with abstractions via Π and λ, then why have another kind of abstraction in the contexts? Why write $\langle x : A\rangle$ and $\langle x : A\rangle B$ to describe the binding rather than use λ and Π?". We believe strongly that the binding operators λ and Π should be used in the contexts as well and indeed in many of our work we do so [KN 95b, BKN 96]. In this paper however, we refrain from this option because we do not want to give the false impression that a binding in the context using either λ or Π has anything to do with solving the problems of Π-reduction that we are tackling. If the reader however is interested in using λ's and Π's in the context and hence in writing $\lambda_{x:A}$ or $\Pi_{x:A}$, for declarations and $(\lambda_{x:A}.-)B$ or $(\Pi_{x:A}.-)B$ for abbreviations, then he/she may like

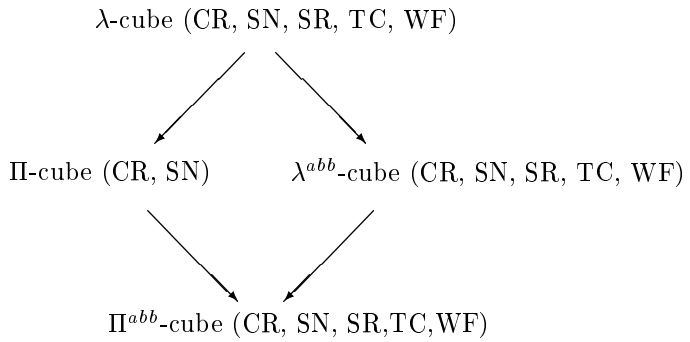

Figure 1: Properties of the λ-cube with various extensions

to know that with such an account and the following new version of the (abb rule):

$$(\pi\text{-abb rule}) \qquad \frac{\Gamma.(\pi_{x:A}.-)B \vdash C : D}{\Gamma \vdash (\pi_{x:A}.C)B : D[x := B]} \text{ where } \pi \in \{\lambda, \Pi\}$$

one can show the following lemma, the proof of which is by a simple induction on the derivation rules:

Lemma 5.1 *($\lambda\Pi$-exchanging) The following holds:*

1. $\Gamma.\lambda_{x:A}.\Delta \vdash_{\beta\Pi a} C : D \iff \Gamma.\Pi_{x:A}.\Delta \vdash_{\beta\Pi a} C : D$
2. $\Gamma.(\lambda_{x:A}.-)B.\Delta \vdash_{\beta\Pi a} C : D \iff \Gamma.(\Pi_{x:A}.-)B.\Delta \vdash_{\beta\Pi a} C : D$

Note that, although λ and Π can be freely interchanged in contexts, it remains desirable to keep a distinction between them as two abstractors in our terms. This is what we do in this article. In fact, we don't distinguish them in the context (by writing $\langle x : A\rangle$ instead of either $\lambda_{x:A}$ or $\Pi_{x:A}$). We do however distinguish them on the right hand side of $\vdash$. For example, from $\langle x : *\rangle \vdash x : *$ using the formation rule $(\Box, \Box)$, we get: $\vdash \lambda_{x:*}.x : \Pi_{x:*}.*$ and $\vdash \Pi_{x:*}.* : \Box$. Note also that $\vdash \Pi_{x:*}.x : *$ which shows that a λ-abstraction has a different type than a Π-abstraction.

6 Acknowledgements

The authors are grateful to an anonymous referee for the useful comments and suggestions. Kamareddine is grateful to Assaf Kfoury and Joe Wells for their hospitality while preparing this article. Moreover, she is grateful to the Department of Mathematics and Computing Science, Eindhoven University of Technology, for their financial support and hospitality from October 1991 to September 1992, and during various regular visits since then. She is, furthermore, grateful to the Dutch organisation of research (NWO), to the British Council and to the Action for Basic Research ESPRIT Project "Types for Proofs and Programs" for their financial support. Bloo has been supported by the Netherlands Computer Science Research Foundation (SION) with financial support from the Netherlands Organisation for Scientific Research (NWO) and is grateful to the Department of Computing Science, Glasgow University, for their financial support and hospitality during work on this subject.

References

[AFMOW95] Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler, A call-by-need lambda calculus, In *Conf. Rec. 22nd Ann. ACM Symp. Principles Programming Languages*, 1995.

[Ba 84] H. Barendregt, *Lambda Calculus: its Syntax and Semantics,* North-Holland, 1984.

[Ba 92] H. Barendregt, Lambda calculi with types, *Handbook of Logic in Computer Science, II*, eds. S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, Oxford University Press, 118-414, 1992.

[BLR96] Z. Benaissa, P. Lescanne, and K. Rose, Modeling Sharing and Recursion for Weak Reduction Strategies Using Explicit Substitution, PLILP96, *Lecture Notes in Computer Science 1140*, Springer Verlag, 1996.

[BKN 96] R. Bloo, F. Kamareddine, and R.P. Nederpelt, The Barendregt Cube with Definitions and Generalised Reduction, *Information and Computation 126(2)*, 123-143, 1996.

[Con 86] R. Constable et al., *Implementing Mathematics with the NUPRL Development System*, Prentice-Hall, 1986.

[Dow 91] G. Dowek, et al. The Coq proof assistant version 5.6, users guide, rapport de recherche 134, INRIA, 1991.

[Geu 95] H. Geuvers, A short and flexible proof of strong normalization for the Calculus of Constructions, in *Types for Proofs and Programs*, eds. P. Dybjer, B. Nordström, and J. Smith, International Workshop TYPES '94, LNCS 996, 14-38, Springer, 1995.

[LP 92] Z. Luo, and R. Pollack,, LEGO proof development system: User's manual, Technical report ECS-LFCS-92-211, LFCS, University of Edinburgh, 1992.

[KN 95a] F. Kamareddine, and R.P. Nederpelt, Refining reduction in the λ-calculus, *Functional Programming 5 (4)*, 637-651, 1995.

[KN 95b] F. Kamareddine, and R.P. Nederpelt, Canonical Typing and Π-Conversion in the Barendregt Cube, *Functional Programming 6 (2)*, 245-267, 1996.

[NGV 94] R.P. Nederpelt, J.H. Geuvers, and R.C. de Vrijer, eds., *Selected Papers on AUTOMATH* , North-Holland, 1994.

[PJ 96] S. Peyton Jones, Compilation by transformation: a report from the trenches, in *European Symposium on programming (ESOP'96)*, Springer Verlag LNCS 1058, 1996.

[PM 97] S. Peyton Jones and E. Meijer, *Henk*: a typed intermediate language, *Types In Compilations Workshop*, 1997.

[SA 95] Z. Shao and A.W. Appel, A type-based compiler for standard ML, in *SIGPLAN Symposium on Programming Language Design and Implementation (PLDI'95)*, La Jolla, ACM, 1995.

[SP 93] P. Severi, and E. Poll, Pure Type Systems with Definitions, Computing Science Note 93/24, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1993.

[TMCSHL 96] D. Tarditi, G. Morrisett, P. Cheng, C. Stone, R. Harper and P. Lee, TIL: A Type-Directed Optimizing Compiler for ML, in *SIGPLAN Symposium on Programming Language Design and Implementation (PLDI'96)*, Philadelphia, ACM, 1996.