Definitions and Π-conversion in Type Theory[*][†]

Fairouz Kamareddine [‡] Roel Bloo[§] and Rob Nederpelt [¶]

April 15, 1997

Abstract

In [KN 95b], the Barendregt Cube was extended with Π-conversion. The resulting system had only a Weak form of Subject Reduction. In this paper, the Cube is extended with explicit definitions. We show that the Cube extended with either explicit definitions alone or with both explicit definitions and Π-conversion satisfies all its original properties including Subject Reduction.

1 Introduction

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 exception for this are some Automath languages in [NGV 95] and the current work of [KN 94] and [KN 95b]. We claim that $\to_\Pi$ is desirable for the following reasons:

1. Π is a kind of λ. In various higher order type theories, arrow-types of the form $A \to B$ are replaced by dependent products $\Pi_{x:A}.B$, where B may contain x as a free variable, and thus may *depend on* x. This means that abstraction can be over types, similarly to the abstraction 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 and hence is eligible for an application.

2. Compatibility. Here are two important rules in the Cube:

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

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

[*]Kamareddine 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 short visits since 1993. She is, furthermore, grateful to the Dutch organisation of research (NWO) and to the Action for Basic Research ESPRIT Project "Types for Proofs and Programs" for their financial support. Last but not least, Kamreddine's work is carried under the EPSRC grant number GR/K25014.

[†]Bloo is 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, Glagow University, for their financial support and hospitality during work on this subject.

[‡]Department of Computing Science, 17 Lilybank Gardens, University of Glasgow, Glasgow G12 8QQ, Scotland, *email*: fairouz@dcs.glasgow.ac.uk

[§]Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O.Box 513, 5600 MB Eindhoven, the Netherlands, *email*: bloo@win.tue.nl

[¶]Same address as Bloo. *email*: wsinrpn@win.tue.nl

The abstraction rule may be regarded as the compatibility property for the typing of abstraction. That is,

$$b : B \Rightarrow \lambda_{x:A}.b : \Pi_{x:A}.B$$

The compatibility property for the typing of application is lost however. In fact, from the application rule, we do not have

$$F : \Pi_{x:A}.B \Rightarrow Fa : (\Pi_{x:A}.B)a$$

but instead

$$F : \Pi_{x:A}.B \Rightarrow Fa : B[x := a]$$

To get compatibility for the typing of application, we need to add $\rightarrow_\Pi$ and to change the application rule to:

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

3. The Automath experience. One might argue that *implicit* Π-reduction (as is the case of the ordinary Cube with the old application rule above) is closer to intuition in the most usual applications. However, experiences with the Automath-languages ([NGV 95]), 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.

4. Preference types, higher order, conversion. In [KN 95b], Π-reduction was shown to have various advantages which include calculating the preference type of a term, the ability of incorporating different degrees (rather than just the two, λ and Π, as in the cube), the splitting of $\Gamma \vdash A : B$ into $\Gamma \vdash A$ (A is typable in Γ) and $\tau(\Gamma, A) = B$ (B is convertible to the preference type of A), and the getting rid of the following rule of the cube:

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

All the above are reasons why it is interesting to study Π-conversion in the cube. Alas however, Π-conversion added to the cube is not a straightforward adding of $(\Pi_{x:A}.B)C \rightarrow_\Pi B[x := C]$ and of the new application rule (see [KN 95b]). First, Subject Reduction (SR) fails. That is, with Π-reduction and the new application, $\Gamma \vdash A : B$ and $A \twoheadrightarrow A'$ may not imply $\Gamma \vdash A' : B$. Church-Rosser (CR) and Strong Normalisation (SN) do hold however. Furthermore, if $\Gamma \vdash A : B$ then neither Γ nor A contain Π-redexes and if B contained a Π-redex, then B is itself that Π-redex. This means that the presence of Π-redexes is restricted.

The problem really lies in a fundamental shortcoming of the usual formulations of type theory. That is, most type theories avoid *explicit definitions* even though actual implementations of these type theories do use such explicit definitions.

In many type theories and lambda calculi, there is no possibility to introduce definitions which are abbreviations for large expressions and 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 HOL ([GM 93]) do provide this possibility. But what are definitions and why are they attractive? Definitions are name abbreviating expressions and occur in contexts where we reason about terms.

Example 1.1 `Let` $id : A \rightarrow A$ `be` $(\lambda_{x:A}.x)$ `in` $(\lambda_{y:A\rightarrow A}.id)id$ defines id to be $(\lambda_{x:A}.x)$ in a more complex expression in which id occurs two times.

The intended meaning of a definition `Let` $x : A$ `be` a `in` b is that the definiendum x can be substituted by the definiens a in the expression b. In a sense, the expression `let` $x : A$ `be` a `in` b is similar to $(\lambda_{x:A}.b)a$. It is not intended however to substitute all the occurrences of x in b by a. Nor is it intended that such a definition is a part of our term. Rather, the definition will live in the environment (or context) in which we evaluate or reason about the expression.

One of the advantages of the definition `let` $x : A$ `be` a `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 `let` $x : A$ `be` a `in` b only once, even in lazy languages. A further advantage is that defining 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. A disadvantage is that the definition may hide information, as is shown in the following example.

Example 1.2 Without definitions, it is not possible to type $\lambda_{y:x}.\lambda_{f:a\rightarrow a}.fy$ even when we somehow know that x is an abbreviation for a. This is because f expects an argument of type a, and y is of type x. Once we make use of the fact that x is defined to be a in our context, then y will have type a and the term will be typable (see Example 1.3).

Practical experiences with type systems show that definitions are absolutely indispensable for any realistic application. Without definitions, terms soon become forbiddingly complicated. By using definitions 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 definitions.

Introducing definitions in Pure Type Systems is an interesting subject of research at the moment (see [SP 93] and [BKN 9y]). Furthermore, [BKN 9y] has shown that the generated type derivations for terms in the Cube with definitions become much shorter than those in the absence of definitions. Our approach in this paper is to introduce definitions as redexes where the body is not written. For example, $(\lambda_{x:A}.-)a$ defines x to be a. Furthermore, we include definitions in contexts with the condition that if a definition occurs in a context then it can be used anywhere in the term we are reasoning about in that context. This explains why we did not write the body of the definition. In other words, it is redundant to write the body. Now, if we look at Example 1.2, then we can type the term now that we allow definitions to occur in contexts and we extend $\vdash$ slightly so that it can see what is in its context.

Example 1.3 We use as context $(\lambda_{x:A}.-)a$, establishing that x of type A is defined as a. Now,

$$(\lambda_{x:A}.-)a.\lambda_{y:x}.\lambda_{f:a\rightarrow a} \vdash f : a \rightarrow a$$
$$(\lambda_{x:A}.-)a.\lambda_{y:x}.\lambda_{f:a\rightarrow a} \vdash y : x = a$$
$$(\lambda_{x:A}.-)a.\lambda_{y:x}.\lambda_{f:a\rightarrow a} \vdash fy : a$$
$$(\lambda_{x:A}.-)a.\lambda_{y:x} \vdash \lambda_{f:a\rightarrow a}.fy : (a \rightarrow a) \rightarrow a$$
$$(\lambda_{x:A}.-)a \vdash \lambda_{y:x}.\lambda_{f:a\rightarrow a}.fy : x \rightarrow (a \rightarrow a) \rightarrow a \ = \ a \rightarrow (a \rightarrow a) \rightarrow a$$

In this paper, we shall show that the cube extended with explicit definitions satisfies all its original properties and is a conservative extension of the cube without definitions. We shall show furthermore, that the cube extended with both explicit definitions and Π-reduction ($\rightarrow_\Pi$ and the new application rule), also preserves all its original properties (including SR). This means that explicit definitions (being important on their own) have repaired the problem of SR in the cube with Π-reduction as was left in [KN 95b].

It may be puzzling as to why would definitions solve the problem of Subject reduction. Let us explain why.

Subject Reduction in the Cube extended with Π-reduction was lost because type correctness itself was lost. That is, one can have $\Gamma \vdash A : B$ without having $B \equiv \Box$ or $\Gamma \vdash B : S$ for some sort $S \in \{*, \Box\}$. More precisely, the new terms that one can get of the form $(\Pi_{x:A}.B)C$ are neither equivalent to $\Box$ nor of type S form some S. I.e.

$$(\dagger) \qquad \text{For every } S,\ \Gamma \not\vdash (\Pi_{x:A}.B)C : S$$

This implies that, for example, even though $\lambda_{z:*}.\lambda_{x:z} \vdash (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$ and $(\lambda_{y:z}.y)x \to x$, we can't show $\lambda_{z:*}.\lambda_{x:z} \vdash x : (\Pi_{y:z}.z)x$.

In fact, to show this last formula, the only option is to use the conversion rule (above) and for this we need that $\lambda_{z:*}.\lambda_{x:z} \vdash (\Pi_{y:z}.z)x : S$ for some S. But this is not possible according to $(\dagger)$ above.

Well, some reflection leads us to the following: look at $(\Pi_{y:z}.z)x$. It can be seen as expressing that the variable y must be read as x. This is not too far away from saying that y is *defined* to be x. So if we include definitions in our contexts we solve the problem. First let us give the extra typing rule which deals with definitions (here π ranges over λ and Π):

$$\text{(def rule)} \qquad \frac{\Gamma.(\pi_{x:A}-)B \vdash C : D}{\Gamma \vdash (\pi_{x:A}.C)B : D[x := B]}$$

The intuition behind this definition is obvious. It says that if $C : D$ can be typed using the definition that x of type A is B, then $(\pi_{x:A}.C)B : D[x := B]$ can be typed without this definition (π ranges over both λ and Π). Furthermore, with explicit definitions, there is no restriction on Π-redexes in terms, types or contexts.

Now, using this def rule, we can solve the problem of correctness of types (and hence retrieve Subject Reduction). This can be seen in our above example as follows:

$\lambda_{z:*}.\lambda_{x:z} \vdash z : *$ and hence $\lambda_{z:*}.\lambda_{x:z}.(\Pi_{y:z}.-)x \vdash z : *$. Now, we use the def rule to get: $\lambda_{z:*}.\lambda_{x:z} \vdash (\Pi_{y:z}.z)x : *[y := x] = *$ which is what we wanted.

Now, use conversion on $\Gamma \vdash x : z$, $\Gamma \vdash (\Pi_{y:z}.z)x : *$ and $(\Pi_{y:z}.z)x = x$ for $\Gamma \equiv \lambda_{z:*}.\lambda_{x:z}$, to get $\Gamma \vdash x : (\Pi_{y:z}.z)x$.

(Note that in the def rule, we took $(\pi_{x:A}.C)B : D[x := B]$ instead of $(\pi_{x:A}.C)B : (\pi_{x:A}.D)B$. This is in order to avoid terms like $(\pi_{x:A}.\Box)B$ which are not acceptable in the Cube.)

Some readers might complain now that adding definitions in order to repair SR in the presence of Π-reduction is too strong. They might prefer the following rule:

$$\text{(appfor rule)} \qquad \frac{\Gamma \vdash \pi_{x:A}.C : S \qquad \Gamma \vdash B : A}{\Gamma \vdash (\pi_{x:A}.C)B : S}$$

However, with this rule, we lose the compatibility of the typing of application. Moreover, a simple yet powerful system of definitions (as we propose in this paper) is worth studying.

Following the above observations, we divide the paper as follows:

1. In section 2, we set up the machinery for both explicit definitions and Π-reduction.

2. In section 3, we introduce the original relation of the 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 $\vdash_{re}$ which is $\vdash_r$ (for $r = \beta$ or $\beta\Pi$) extended with definitions. We show that all the properties of the cube remain valid for $\vdash_{re}$.

4. In section 5, we compare our system of definitions in this paper to other typing systems.

2 The Formal Machinery

The systems of the Cube (see [Barendregt 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 π ranges over the abstraction operators Π and λ, V is an infinite collection of variables over which $\alpha, \beta, x, y, z, \ldots$ range. $*$ and $\Box$ are called sorts over which $S, S_1, S_2, \ldots$ are used to range. We take $A, B, a, b \ldots$ to 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. For example, we take $\lambda_{x:A}.x \equiv \lambda_{y:A}.y$ where $\equiv$ is used to denote syntactical equality of terms. 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.

Terms can be related via a reduction relation $\to_r$. We assume the usual definition of the compatibility of a reduction relation, and define $\twoheadrightarrow_r$ to be its reflexive transitive closure and $=_r$ to be its equivalence closure. We use in this paper two reduction relations: $\to_\beta$ generated by the axiom $(\lambda_{x:A}.B)C \to_\beta B[x := C]$ and $\to_{\beta\Pi}$ generated by the axiom $(\pi_{x:A}.B)C \to_{\beta\Pi} B[x := C]$. Throughout, we let r range over $\{\beta, \beta\Pi\}$.

In the following definition, declarations are familiar from the Cube. We write them however as $\pi_{x:A}$ instead of $x : A$ because we think it clearer and because in the case of a definition, we need a kind of redex, hence the π must be present. $\subseteq'$ moreover is familiar in the case of declarations. In the case of a definition however, it says that changing a declaration into a definition which preserves that declaration, does not affect the information in the context.

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

1. *A declaration d is of the form $\pi_{x:A}$. We define* `subj`(d) *and* `pred`(d) *to be x and A respectively.*

2. *A definition d is of the form $(\pi_{x:A}.-)B$ and defines x of type A to be B. We define* `subj`(d)*,* `pred`(d) *and* `def`(d) *to be x, A, and B respectively.*

3. *We use $d, d_1, d_2, \ldots$ to range over declarations and definitions.*

4. *A pseudocontext Γ is a (possibly empty) concatenation of declarations and definitions $d_1.d_2.\cdots.d_n$ such that if $i \neq j$, then $\mathtt{subj}(d_i) \not\equiv \mathtt{subj}(d_j)$. We use $\Gamma, \Delta, \Gamma', \Gamma_1, \Gamma_2, \ldots$ to range over pseudocontexts.*

5. *We define $dom(\Gamma) = \{\mathtt{subj}(d) \mid d \in \Gamma\}$, $\Gamma\text{-}\mathtt{decl} = \{d \in \Gamma \mid d \text{ is a declaration }\}$ and $\Gamma\text{-}\mathtt{def} = \{d \in \Gamma \mid d \text{ is a definition }\}$ for any pseudocontext Γ. Note that $dom(\Gamma) = \{\mathtt{subj}(d) \mid d \in \Gamma\text{-}\mathtt{decl} \cup \Gamma\text{-}\mathtt{def}\}$.*

6. *Define $\subseteq'$ between pseudocontexts as the least reflexive transitive relation satisfying:*
   - *$\Gamma.\Delta \subseteq' \Gamma.d.\Delta$ for d a declaration or a definition.*
   - *$\Gamma.\pi_{x:A}.\Delta \subseteq' \Gamma.(\pi_{x:A}.-)B.\Delta$*

Again the following definition is familiar from the Cube. The only mysterious concept might be $\prec$. It is here however in order to group some preconditions of some typing rules. For example, instead of postulating for the start rule (in the case of a declaration) that $\Gamma \vdash \mathtt{pred}(d) : S$ and $\mathtt{subj}(d) \notin \Gamma$, we say $\Gamma \prec d$. This becomes particularly useful in the case of definitions.

In the rest of this section, we assume $\vdash$ to be a notion of derivability.

Definition 2.3 *Let Γ be a pseudocontext and r a reduction relation.*

1. *A statement is of the form $A : B$, A and B are called the subject and the predicate of the statement respectively.*

2. *When $A : B$ is a statement, we call $\Gamma \vdash_r A : B$ a judgement, and write $\Gamma \vdash_r A : B : C$ to mean $\Gamma \vdash_r A : B \wedge \Gamma \vdash_r B : C$.*

3. *For $d \in \Gamma\text{-}\mathtt{def} \cup \Gamma\text{-}\mathtt{decl}$, we say Γ invites d, notation $\Gamma \prec d$, iff*
   - *$\Gamma.d$ is a pseudocontext*
   - *$\Gamma \vdash_r \mathtt{pred}(d) : S$ for some sort S.*
   - *if d is a definition then $\Gamma \vdash_r \mathtt{def}(d) : \mathtt{pred}(d)$*

4. *Γ is called legal if $\exists P, Q \in \mathcal{T}$ such that $\Gamma \vdash_r P : Q$.*

5. *$A \in \mathcal{T}$ is called a Γ-term if $\exists B \in \mathcal{T}[\Gamma \vdash_r A : B \vee \Gamma \vdash_r B : A]$.*
   We take $\Gamma\text{-terms} = \{A \in \mathcal{T} \mid \exists B \in \mathcal{T}[\Gamma \vdash_r A : B \vee \Gamma \vdash_r B : A]\}$.
   $A \in \mathcal{T}$ is called legal if $\exists \Gamma[A \in \Gamma\text{-terms}]$.

Remark 2.4 *The definition of $\Gamma \prec d$ and the definition of $\Gamma \vdash_r A : B$ depend mutually recursively on each other. This isn't a problem however since in the definition of $\Gamma \vdash_r A : B$ only $\Gamma' \prec d$ are needed for Γ' smaller than Γ.*

Let r be either β or $\beta\Pi$. The following definition is needed in the conversion rule (in the presence of explicit definitions) where we replace $A =_r B$ by $\Gamma \vdash_r A =_{\mathtt{def}} B$.

Definition 2.5 *(Definitional r-equality) For all pseudocontexts Γ we define the binary relation $\Gamma \vdash_r \cdot =_{\mathtt{def}} \cdot$ to be the equivalence relation generated by*

- *if $A =_r B$ then $\Gamma \vdash_r A =_{\texttt{def}} B$*
- *if $d \in \Gamma\text{-}\texttt{def}$ and $A, B \in \mathcal{T}$ such that B arises from A by substituting one particular occurrence of $\texttt{subj}(d)$ in A by $\texttt{def}(d)$, then $\Gamma \vdash_r A =_{\texttt{def}} B$.*

Remark 2.6 If no definitions are present in Γ then $\Gamma \vdash_r A =_{\texttt{def}} B$ is the same as $A =_r B$.

Finally, the following definition is again familiar from the cube, but we extend it to deal with definitions. That is, $\Gamma \vdash_r (\pi_{x:A}.-)B$ iff $\Gamma \vdash_r x : A$, $\Gamma \vdash_r B : A$ and $\Gamma \vdash_r x =_{\texttt{def}} B$.

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

- *If d is a declaration: $\Gamma \vdash_r d$ iff $\Gamma \vdash_r \texttt{subj}(d) : \texttt{pred}(d)$.*
- *If d is a definition: $\Gamma \vdash_r d$ iff $\Gamma \vdash_r \texttt{subj}(d) : \texttt{pred}(d) \wedge \Gamma \vdash_r \texttt{def}(d) : \texttt{pred}(d) \wedge \Gamma \vdash_r \texttt{subj}(d) =_{\texttt{def}} \texttt{def}(d)$.*
- *$\Gamma \vdash_r d_1 \cdots d_n$ iff $\Gamma.d_1.\cdots.d_{i-1} \vdash_r d_i$ for all $1 \leq i \leq n$.*

3 Extending the Cube with Π-reduction

In the Cube as presented in [Barendregt 92], the only declarations allowed are of the form $\lambda_{x:A}$. Hence there are no definitions in the contexts, nor declarations of the form $\Pi_{x:A}$. Therefore, $\Gamma \prec d$ is of the form $\Gamma \prec \lambda_{x:A}$ and means that $\Gamma \vdash A : S$ for some S and that x is fresh in Γ, A. Moreover, for any $d \equiv \lambda_{x:A}$, remember that $\texttt{subj}(d) \equiv x$ and $\texttt{pred}(d) \equiv A$. Moreover, Π-reduction is not allowed. **Hence, in the following definition, d is a meta-variable for declarations only and $=_{\texttt{def}}$ is the same as $=_\beta$ (which is independent of $\vdash_\beta$).**

Definition 3.1 *(Axioms and rules of the Cube: d is a declaration, $=_{\texttt{def}}$ is $=_\beta$)*

(axiom) $\quad <> \vdash_\beta * : \Box$

(start rule)
$$\frac{\Gamma \prec d}{\Gamma.d \vdash_\beta \texttt{subj}(d) : \texttt{pred}(d)}$$

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

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

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

(conversion rule)
$$\frac{\Gamma \vdash_\beta A : B \qquad \Gamma \vdash_\beta B' : S \qquad \Gamma \vdash_\beta B =_{\texttt{def}} B'}{\Gamma \vdash_\beta A : B'}$$

(formation rule)
$$\frac{\Gamma \vdash_\beta A : S_1 \qquad \Gamma.\lambda_{x:A} \vdash_\beta B : S_2}{\Gamma \vdash_\beta \Pi_{x:A}.B : S_2} \quad \text{if } (S_1, S_2) \text{ is a rule}$$

Each of the eight systems of the Cube is obtained by taking the (S_1, S_2) rules allowed from a subset of $\{(*,*), (*,\Box), (\Box,*), (\Box,\Box)\}$. The basic system is the one where $(S_1, S_2) = (*,*)$ is the only possible choice. All other systems have this version of the formation rules, plus one or more other combinations of $(*,\Box)$, $(\Box,*)$ and $(\Box,\Box)$ for (S_1, S_2). Here is the table which presents the eight systems of the Cube:

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 the above Cube by changing $\twoheadrightarrow_\beta$ to $\twoheadrightarrow_{\beta\Pi}$ and by changing $\vdash_\beta$ to $\vdash_{\beta\Pi}$:

Definition 3.2 *($\vdash_{\beta\Pi}$) We define $\vdash_{\beta\Pi}$ as $\vdash_\beta$ with the difference that the application and conversion rules change as follows:*

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

(new conversion rule) $$\frac{\Gamma \vdash_{\beta\Pi} A : B \qquad \Gamma \vdash_{\beta\Pi} B' : S \qquad \Gamma \vdash_{\beta\Pi} B =_{\mathtt{def}} B'}{\Gamma \vdash_{\beta\Pi} A : B'}$$

Note that $\Gamma \vdash_{\beta\Pi} B =_{\mathtt{def}} B'$ is the same as $B =_{\beta\Pi} B'$, as no definitions are allowed in the context.

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 and SR) will be established for the cube extended with either definitions alone, or with both definitions 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 $C \twoheadrightarrow_r D$ □

Lemma 3.4 *(Start Lemma for $\vdash_r$)*
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 $\vdash_\beta$)*
If $\Gamma \vdash_\beta A : B$ then ($B \equiv \Box$ or $\Gamma \vdash_\beta B : S$ for some sort S). □

Remark 3.6 *(Correctness of types does not hold for $\vdash_{\beta\Pi}$)*
The new legal terms of the form $(\Pi_{x:B}.C)A$ imply the failure of type correctness for $\vdash_{\beta\Pi}$. That is, even in $\lambda_\to$, $\Gamma \vdash_{\beta\Pi} A : B \not\Rightarrow$ ($B \equiv \Box$ or $\Gamma \vdash_{\beta\Pi} B : S$ for some sort S). For example, if $\Gamma \equiv \lambda_{z:*}.\lambda_{x:z}$ then $\Gamma \vdash_{\beta\Pi} (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$, but $\Gamma \not\vdash_{\beta\Pi} (\Pi_{y:z}.z)x : S$ from Lemma 3.8.

Failure of correctness of types for $\vdash_{\beta\Pi}$ implies failure of Subject Reduction even in $\lambda_\to$:

Example 3.7 In $\lambda_\to$, $\lambda_{z:*}.\lambda_{x:z} \not\vdash_{\beta\Pi} x : (\Pi_{y:z}.z)x$. Otherwise, by generation: $\lambda_{z:*}.\lambda_{x:z} \vdash_{\beta\Pi} (\Pi_{y:z}.z)x : S$, which is absurd by Lemma 3.8. Yet in $\lambda_\to$, $\lambda_{z:*}.\lambda_{x:z} \vdash_{\beta\Pi} (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$.

Lemma 3.8 *For any A, B, C, S, Γ: $\Gamma \not\vdash_{\beta\Pi} (\Pi_{x:A}.B)C : S$.* $\Box$

Lemma 3.9 *(Legal terms and contexts for $\vdash_\beta$ and $\twoheadrightarrow_\beta$)*
$\vdash_\beta$-legal terms and contexts contain no Π-redexes. $\Box$

Lemma 3.10 *(Legal terms and contexts for $\vdash_{\beta\Pi}$ and $\twoheadrightarrow_{\beta\Pi}$)*

1. *If $\Gamma \vdash_{\beta\Pi} A : B$ then A and Γ are free of Π-redexes, and either B contains no Π-redexes or B is the only Π-redex in B.*

2. *If $(\Pi_{x:D}.E)B$ is $\vdash_{\beta\Pi}$-legal, then $E[x := B]$ contains no Π-redexes.* $\Box$

Theorem 3.11 *(Subject Reduction SR, for $\vdash_\beta$ and $\twoheadrightarrow_\beta$)*
$\Gamma \vdash_\beta A : B \wedge A \twoheadrightarrow_\beta A' \Rightarrow \Gamma \vdash_\beta A' : B$ $\Box$

As explained in Example 3.7, SR fails for $\vdash_{\beta\Pi}$. A weak form of SR holds however. First we need the following definition which removes all Π-redexes of a $\vdash_{\beta\Pi}$-legal term (see Lemma 3.10):

Definition 3.12 *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 3.13 *(Weak Subject Reduction for $\vdash_{\beta\Pi}$ and $\twoheadrightarrow_{\beta\Pi}$)*
$\Gamma \vdash_{\beta\Pi} A : B \wedge A \twoheadrightarrow_{\beta\Pi} A' \Rightarrow \Gamma \vdash_{\beta\Pi} A' : \hat{B}$ $\Box$

Lemma 3.14 *(Unicity of Types for $\vdash_r$ and $\twoheadrightarrow_r$)*
$\Gamma \vdash_r A : B_1 \wedge \Gamma \vdash_r A : B_2 \Rightarrow B_1 =_r B_2$ $\Box$

Theorem 3.15 *(Strong Normalisation with respect to $\vdash_r$ and $\rightarrow_r$)*
If A is $\vdash_r$-legal then $SN_{\rightarrow_r}(A)$; i.e. A is strongly normalising with respect to $\rightarrow_r$. $\Box$

4 Extending the Cube with definitions

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

- One rule, the (*def rule*), is added.
- The use of $\Gamma \vdash B =_{\texttt{def}} B'$ in the conversion rule really has an effect now, rather than simply postulating $B =_r B'$.
- Not only declarations but also definitions are allowed in contexts. Note that in the case $r = \beta$, definitions do not contain Π-redexes and declarations are only $\lambda_{x:A}$ and not $\Pi_{x:A}$.

Note that the intended scope of $\lambda_{x:A}$ in $\Gamma.(\lambda_{x:A}.-)B.\Delta \vdash_r C : D$ is Δ, C and D. This is what should be expected since the scope in $\Gamma.\lambda_{x:A}.\Delta \vdash_r C : D$ is the same.

Definition 4.1 *(Axioms and rules of the Cube extended with definitions: d ranges over declarations and definitions)*

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

$$(\textit{def rule}) \qquad \frac{\Gamma.(\pi_{x:A}.-)B \vdash_{re} C : D}{\Gamma \vdash_{re} (\pi_{x:A}.C)B : D[x := B]}$$

The (*def rule*) says that if $C : D$ can be deduced using a definition $d \equiv (\pi_{x:A}.-)B$, then $(\pi_{x:A}.C)B$ will be of type D where d has been unfolded in D. Recall that in the case $r = \beta$, π is only λ everywhere.

Remark 4.2 A natural alternative for the *(def rule)* would be

$$\text{(alternative def rule)} \qquad \frac{\Gamma.(\pi_{x:A}.-)B \vdash_{re} C : D}{\Gamma \vdash_{re} (\pi_{x:A}.C)B : (\pi_{x:A}.D)B}$$

We didn't choose the alternative def rule because of the problem that correctness of types (cf. 4.11) will no longer hold: using (alternative def rule) we can derive $\alpha : * \vdash_{re} (\lambda_{\beta:*}.*)\alpha : (\lambda_{\beta:*}.\Box)\alpha$, but $(\lambda_{\beta:*}.\Box)\alpha$ is of course not typable.

The only way to prevent this problem is by unfolding all definitions in the type of the judgement as is done in the (def rule).

Note that the alternative def rule is a derived rule whenever $D \not\equiv \Box$ because then if $\Gamma.(\pi_{x:A}.-)B \vdash_{re} C : D$ then by correctness of types, $\Gamma.(\pi_{x:A}.-)B \vdash_{re} D : S$ hence by the def rule $\Gamma \vdash_{re} (\pi_{x:A}.D)B : S$. But $D[x := B] =_{\beta\Pi} (\pi_{x:A}.D)B$. Hence by conversion and the def rule, we get $\Gamma \vdash_{re} (\pi_{x:A}.C)B : (\pi_{x:A}.D)B$.

When considering a definition in a term to be syntactically equal to a redex, the (def rule) is quite natural: for instance, deriving a type for $(\lambda_{x:\alpha}.x)y$ via defining y to be x gives the same type as the derivation via abstraction followed by application (let $\Gamma \equiv \lambda_{\alpha:*}.\lambda_{y:\alpha}$):

$$\text{(def rule)} \quad \frac{\Gamma.(\lambda_{x:\alpha}.-)y \vdash_{re} x : \alpha}{\Gamma \vdash_{re} (\lambda_{x:\alpha}.x)y : \alpha[x := y]}$$

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

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

Example 4.3 $\lambda_{\beta:*}.\lambda_{y:\beta} \not\vdash_r (\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)y)\beta : \beta$. The reason for this is that we need $y : \alpha$ to be able to give $(\lambda_{x:\alpha}.x)y$ a type. Looking carefully however, we find that $(\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)y)\beta$ is defining α to be β. So here is how the above derivation can be obtained using definitions (we present a short-cut and do not mention all the steps, nor the names of the rules):

$$\begin{array}{l}
\lambda_{\beta:*}.\lambda_{y:\beta}.(\lambda_{\alpha:*}.-)\beta \vdash_{re} \lambda_{x:\alpha}.x : \Pi_{x:\alpha}.\alpha \\
\lambda_{\beta:*}.\lambda_{y:\beta}.(\lambda_{\alpha:*}.-)\beta \vdash_{re} y : \beta \\
\lambda_{\beta:*}.\lambda_{y:\beta}.(\lambda_{\alpha:*}.-)\beta \vdash_{re} \alpha =_{\mathtt{def}} \beta \\
\lambda_{\beta:*}.\lambda_{y:\beta}.(\lambda_{\alpha:*}.-)\beta \vdash_{re} y : \alpha \\
\lambda_{\beta:*}.\lambda_{y:\beta}.(\lambda_{\alpha:*}.-)\beta \vdash_{re} (\lambda_{x:\alpha}.x)y : \alpha \\
\lambda_{\beta:*}.\lambda_{y:\beta} \vdash_{re} (\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)y)\beta : \alpha[\alpha := \beta] \\
\lambda_{\beta:*}.\lambda_{y:\beta} \vdash_{re} (\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)y)\beta : \beta
\end{array}$$

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

Lemma 4.4 *(Free variable lemma for $\vdash_{re}$)*
Let Γ be a legal context such that $\Gamma \vdash_{re} B : C$. Then the following holds:

1. *If d and d' are two different elements of $\Gamma\text{-}\mathtt{decl} \cup \Gamma\text{-}\mathtt{def}$, then $\mathtt{subj}(d) \not\equiv \mathtt{subj}(d')$.*

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 $\Gamma \vdash_{re} B : C$.* $\Box$

Lemma 4.5 *(Start Lemma for $\vdash_{re}$)*
Let Γ be a legal context. Then $\Gamma \vdash_{re} * : \Box$ and $\forall d \in \Gamma[\Gamma \vdash_{re} d]$.
Proof: *Γ legal $\Rightarrow \exists B, C[\Gamma \vdash_{re} B : C]$; now use induction on $\Gamma \vdash_{re} B : C$.* $\Box$

Lemma 4.6 *(Transitivity Lemma for $\vdash_{re}$)*
Let Γ and Δ be legal contexts. Then: $[\Gamma \vdash_{re} \Delta \wedge \Delta \vdash_{re} A : B] \Rightarrow \Gamma \vdash_{re} A : B$.
Proof: *Induction on the derivation $\Delta \vdash_{re} A : B$.* $\Box$

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

Lemma 4.7 *(Thinning for $\vdash_{re}$)*

1. *If $\Gamma_1.\Gamma_2 \vdash_{re} A =_{\mathtt{def}} B$, $\Gamma_1.\Delta.\Gamma_2$ is a legal context, then $\Gamma_1.\Delta.\Gamma_2 \vdash_{re} A =_{\mathtt{def}} B$.*

2. *If Γ and Δ are legal contexts such that $\Gamma \subseteq' \Delta$ and if $\Gamma \vdash_{re} A : B$, then $\Delta \vdash_{re} A : B$.*

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

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

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

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

Lemma 4.8 *(Substitution lemma for $\vdash_{re}$)*

1. *If $\Gamma.(\pi_{x:C}.-)D.\Delta \vdash_{re} A =_{\mathtt{def}} B$, A and B are $\Gamma.(\pi_{x:C}.-)D.\Delta$-legal, then $\Gamma.\Delta[x := D] \vdash_{re} A[x := D] =_{\mathtt{def}} B[x := D]$.*

2. *If B is a $\Gamma.(\pi_{x:C}.-)D$-legal term, then $\Gamma.(\pi_{x:C}.-)D \vdash_{re} B =_{\mathtt{def}} B[x := D]$.*

3. *If $\Gamma.(\pi_{x:A}.-)B.\Delta \vdash_{re} C : D$, then $\Gamma.\Delta[x := B] \vdash_{re} C[x := B] : D[x := B]$.*

4. *If $\Gamma.\pi_{x:A}.\Delta \vdash_{re} C : D$ and $\Gamma \vdash_{re} B : A$, then $\Gamma.\Delta[x := B] \vdash_{re} C[x := B] : D[x := B]$.*

Proof:

1. *Induction on the derivation rules of $=_{\mathtt{def}}$.*

2. *Induction on the structure of B.*

3. *Induction on the derivation rules, using 1., 2. and thinning.*

4. *Idem.* $\Box$

Lemma 4.9 *(Generation Lemma for $\vdash_{re}$)*

1. $\Gamma \vdash_{re} S : C \Rightarrow S \equiv *, \Gamma \vdash_{re} C =_{\mathtt{def}} \Box$, *and if* $C \not\equiv \Box$ *then* $\Gamma \vdash_{re} C : S'$ *for some sort* S'.
2. *If* $\Gamma \vdash_{re} x : A$ *then for some* $d \in \Gamma$, $x \equiv \mathtt{subj}(d)$, $\Gamma \vdash_{re} A =_{\mathtt{def}} \mathtt{pred}(d)$ *and* $\Gamma \vdash_{re} A : S$ *for some sort* S.
3. *If* $\Gamma \vdash_{re} \lambda_{x:A}.B : C$ *then for some* D *and sort* S: $\Gamma.\lambda_{x:A} \vdash_{re} B : D$, $\Gamma \vdash_{re} \Pi_{x:A}.D : S$, $\Gamma \vdash_{re} \Pi_{x:A}.D =_{\mathtt{def}} C$ *and if* $\Pi_{x:A}.D \not\equiv C$ *then* $\Gamma \vdash_{re} C : S'$ *for some sort* S'.
4. *If* $\Gamma \vdash_{re} \Pi_{x:A}.B : C$ *then for some sorts* S_1, S_2: $\Gamma \vdash_{re} A : S_1$, $\Gamma.\lambda_{x:A} \vdash_{re} B : S_2$, (S_1, S_2) *is a rule,* $\Gamma \vdash_{re} C =_{\mathtt{def}} S_2$ *and if* $S_2 \not\equiv C$ *then* $\Gamma \vdash_{re} C : S$ *for some sort* S.
5. *If* $\Gamma \vdash_{re} Fa : C$, $F \not\equiv \pi_{x:A}.B$, *then for some* D, E: $\Gamma \vdash_{re} a : D$, $\Gamma \vdash_{re} F : \Pi_{x:D}.E$, $\Gamma \vdash_{re} T =_{\mathtt{def}} C$ *and if* $T \not\equiv C$ *then* $\Gamma \vdash_{re} C : S$ *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* $\Gamma \vdash_{re} (\pi_{x:A}.D)B : C$, *then* $\Gamma.(\pi_{x:A}.-)B \vdash_{re} D : C$

Proof: *1., 2., 3., 4. and 5. follow by a tedious but straightforward induction on the derivations (use the thinning lemma). As to 6., an easy induction on the derivation rules shows that one of the following two cases is applicable:*

- $\Gamma.(\pi_{x:A}.-)B \vdash_{re} D : C'$, $\Gamma \vdash_{re} C'[x := B] =_{\mathtt{def}} C$ *and if* $C'[x := B] \not\equiv C$ *then* $\Gamma \vdash_{re} C : S$ *for some sort* S.
- $\Gamma \vdash_{re} B : F$, $\Gamma \vdash_{re} \lambda_{x:A}.D : \Pi_{y:F}.G$, $\Gamma \vdash_{re} C =_{\mathtt{def}} T$ *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* $\Gamma \vdash_{re} C : S$ *for some sort* S.

In the first case use thinning and conversion; in the second case use thinning, conversion and 3. □

Now, recall again that correctness of types fails for $\vdash_{\beta\Pi}$ but holds for $\vdash_{\beta}$. Here we show that it holds for $\vdash_{re}$ for $r = \beta$ or $\beta\Pi$. First, we need the following exchange lemma for $\vdash_{\beta\Pi e}$.

Lemma 4.10 *($\lambda\Pi$-exchanging)*

$\Gamma.\lambda_{x:A}.\Delta \vdash_{\beta\Pi e} C : D \Longleftrightarrow \Gamma.\Pi_{x:A}.\Delta \vdash_{\beta\Pi e} C : D$
and $\Gamma.(\lambda_{x:A}.-)B.\Delta \vdash_{\beta\Pi e} C : D \Longleftrightarrow \Gamma.(\Pi_{x:A}.-)B.\Delta \vdash_{\beta\Pi e} C : D$

Proof: *induction on the derivation rules. We treat one case of the* start rule*:*
$\Gamma.(\lambda_{x:A}.-)B \vdash_{\beta\Pi e} x : A$ *as a consequence of* $\Gamma \prec (\lambda_{x:A}.-)B$. *Then also* $\Gamma \prec (\Pi_{x:A}.-)B$, *so* $\Gamma.(\Pi_{x:A}.-)B \vdash_{\beta\Pi e} x : A$. □

In what follows, we show the correctness of types for both $\vdash_{\beta e}$ and $\vdash_{\beta\Pi e}$.

Corollary 4.11 *(Correctness of Types)*

If $\Gamma \vdash_{re} A : B$ *then* $B \equiv \Box$ *or* $\Gamma \vdash_{re} B : S$ *for some sort* S.

Proof: *By induction on the derivation rules. The interesting cases are the definition and application rules.*

- *In case* $\Gamma \vdash_{re} (\pi_{x:A}.D)B : C[x := B]$ *as a consequence of* $\Gamma.(\pi_{x:A}.-)B \vdash_{re} D : C$, *then by IH* $C \equiv \Box$ *or* $\Gamma.(\pi_{x:A}.-)B \vdash_{re} C : S$ *for some sort* S. *In the first case also* $C[x := B] \equiv \Box$, *in the second case by the Substitution Lemma* $\Gamma \vdash_{re} C[x := B] : S[x := B] \equiv S$.

- *In case* $\Gamma \vdash_{\beta\Pi e} Fa : (\Pi_{x:A}.B)a$ *as a consequence of* $\Gamma \vdash_{\beta\Pi e} F : \Pi_{x:A}.B$, $\Gamma \vdash_{\beta\Pi e} a : A$, *then by the induction hypothesis* $\Gamma \vdash_{\beta\Pi e} \Pi_{x:A}.B : S$ *for some sort* S *and hence by Generation* $\Gamma.\lambda_{x:A} \vdash_{\beta\Pi e} B : S$. *Then by Thinning* $\Gamma.(\lambda_{x:A}.-)a \vdash_{\beta\Pi e} B : S$, *so by* $\lambda\Pi$*-exchanging* $\Gamma.(\Pi_{x:A}.-)a \vdash_{\beta\Pi e} B : S$ *and by the definition rule* $\Gamma \vdash_{\beta\Pi e} (\Pi_{x:A}.B)a : S[x := a] \equiv S$. $\square$

¿From correctness of types for $\vdash_{\beta\Pi e}$, we can establish its Subject Reduction.

Theorem 4.12 *(Subject Reduction for* $\vdash_{\beta\Pi e}$ *and* $\twoheadrightarrow_{\beta\Pi}$*)*
If $\Gamma \vdash_{\beta\Pi e} A : B$ *and* $A \twoheadrightarrow_{\beta\Pi} A'$ *then* $\Gamma \vdash_{\beta\Pi e} A' : B$.
Proof: *We prove by simultaneous induction on the derivation rules:*

1. *If* $\Gamma \vdash_{\beta\Pi e} A : B$ *and* Γ' *results from contracting one of the terms in the declarations and definitions of* Γ *by a one step* $\beta\Pi$*-reduction, then* $\Gamma' \vdash_{\beta\Pi e} A : B$
2. *If* $\Gamma \vdash_{\beta\Pi e} A : B$ *and* $A \rightarrow_{\beta\Pi} A'$ *then* $\Gamma \vdash_{\beta\Pi e} A' : B$

- *(axiom): nothing to prove*
- *(start rule): We consider the case* $d \equiv (\lambda_{x:A}.-)B$, $A \rightarrow_{\beta\Pi} A'$. *The other cases are similar or easy.*

  We have: $\Gamma.(\lambda_{x:A}.-)B \vdash_{\beta\Pi e} x : A$ *as a consequence of* $\Gamma \prec (\lambda_{x:A}.-)B$, *i.e.* $\Gamma \vdash_{\beta\Pi e} B : A : S$. *By the induction hypothesis* $\Gamma \vdash_{\beta\Pi e} A' : S$ *and by the induction hypothesis and conversion* $\Gamma \vdash_{\beta\Pi e} B : A'$. *Hence* $\Gamma.(\lambda_{x:A'}.-)B \vdash_{\beta\Pi e} x : A'$ *and again by conversion* $\Gamma.(\lambda_{x:A'}.-)B \vdash_{\beta\Pi e} x : A$.
- *(weak), (formation), (conversion): use the induction hypothesis.*
- *(abstraction): use the induction hypothesis and conversion.*
- *(definition):* $\Gamma \vdash_{\beta\Pi e} (\pi_{x:A}.D)B : C[x := B]$ *as a consequence of* $\Gamma.(\pi_{x:A}.-)B \vdash_{\beta\Pi e} D : C$. *Now* $\Gamma' \vdash_{\beta\Pi e} (\pi_{x:A}.D)B : C[x := B]$, $\Gamma \vdash_{\beta\Pi e} (\pi_{x:A}.D')B : C[x := B]$ *and* $\Gamma \vdash_{\beta\Pi e} (\pi_{x:A'}.D)B : C[x := B]$ *by the induction hypothesis.*

  Furthermore, if $B \rightarrow_{\beta\Pi} B'$ *then* $\Gamma \vdash_{\beta\Pi e} C[x := B] =_{\mathtt{def}} C[x := B']$ *and by the induction hypothesis and definition rule we get* $\Gamma \vdash_{\beta\Pi e} (\pi_{x:A}.D)B' : C[x := B']$. *Now by Lemma 4.11,* $C \equiv \square$ *or* $\Gamma.(\pi_{x:A}.-)B \vdash_{\beta\Pi e} C : S$ *for some sort* S. *In the first case,* $C[x := B] \equiv C \equiv C[x := B']$ *and we are done, in the second case by the Substitution Lemma* $\Gamma \vdash_{\beta\Pi e} C[x := B] : S[x := B] \equiv S$, *so by conversion* $\Gamma \vdash_{\beta\Pi e} (\pi_{x:A}.D)B' : C[x := B]$.

  For the last possibility, $(\pi_{x:A}.D)B \rightarrow_{\beta\Pi} D[x := B]$, *we remark that by the Substitution Lemma we get out of* $\Gamma.(\pi_{x:A}.-)B \vdash_{\beta\Pi e} D : C$ *that* $\Gamma \vdash_{\beta\Pi e} D[x := B] : C[x := B]$.
- *(application):* $\Gamma \vdash_{\beta\Pi e} Fa : (\Pi_{x:A}.B)a$ *as a consequence of* $\Gamma \vdash_{\beta\Pi e} F : \Pi_{x:A}.B$ *and* $\Gamma \vdash_{\beta\Pi e} a : A$. *Then* $\Gamma' \vdash_{\beta\Pi e} Fa : (\Pi_{x:A}.B)a$ *and* $\Gamma \vdash_{\beta\Pi e} F'a : (\Pi_{x:A}.B)a$ *by the induction hypothesis, and* $\Gamma \vdash_{\beta\Pi e} Fa' : (\Pi_{x:A}.B)a$ *because by the induction hypothesis* $\Gamma \vdash_{\beta\Pi e} Fa' : (\Pi_{x:A}.B)a'$, *by Corollary 4.11 (Correctness of Types)* $\Gamma \vdash_{\beta\Pi e} (\Pi_{x:A}.B)a : S$ *for some sort* S, *so by conversion* $\Gamma \vdash_{\beta\Pi e} Fa' : (\Pi_{x:A}.B)a$.

  Now the crucial case: $F \equiv (\pi_{y:C}.D)$, $Fa \rightarrow_{\beta\Pi} D[y := a]$. *Then* $\Gamma \vdash_{\beta\Pi e} (\pi_{y:C}.D)a : (\Pi_{x:A}.B)a$ *so by the Generation Lemma* $\Gamma.(\pi_{y:C}.-)a \vdash_{\beta\Pi e} D : (\Pi_{x:A}.B)a$, *now by the Substitution Lemma* $\Gamma \vdash_{\beta\Pi e} D[y := a] : ((\Pi_{x:A}.B)a)[y := a]$, *but by the Barendregt convention* $((\Pi_{x:A}.B)a)[y := a] \equiv (\Pi_{x:A}.B)a$ *so we are done.* $\square$

Subject Reduction for $\vdash_{\beta e}$ is easier:

Theorem 4.13 *(Subject Reduction for $\vdash_{\beta e}$ and $\twoheadrightarrow_\beta$)*
$\Gamma \vdash_{\beta e} A : B$ and $A \twoheadrightarrow A'$ then $\Gamma \vdash_{\beta e} A' : B$.

Proof: *We only need to consider $A \to_\beta A'$. Suppose $\Gamma \vdash_{\beta e} (\lambda_{x:B}.C)A : D$. Then by generation, $\Gamma.(\lambda_{x:B}.-)A \vdash_{\beta e} C : D$, and by substitution we get $\Gamma \vdash_{\beta e} C[x := A] : D[x := A]$, but as $x \notin FV(D)$, $D[x := A] \equiv D$. The compatibility cases are easy.* □

The proof of Strong Normalisation is based on Strong Normalisation of the λ-cube extended with definitions as in [BKN 9y].

Theorem 4.14 *(Strong Normalisation for the Cube with respect to $\vdash_{\beta e}$ and $\to_\beta$)*
If A is a $\vdash_{\beta e}$-legal term then A is strongly normalising with respect to $\twoheadrightarrow_\beta$.

Proof: *$\vdash_{\beta e}$ in this paper is a subset of $\vdash_e$ of [BKN 9y] in that if $\Gamma \vdash_{\beta e} A : B$ then $\Gamma \vdash_e A : B$ (see subsection 5.5). Now, Strong normalisation for $\vdash_{\beta e}$ follows from that of $\vdash_e$ (again see [BKN 9y] for the lengthy but standard (similar to the proof of Geuvers for λC in [Geuvers 94]) proof of SN for $\vdash_e$ which can be adapted to $\vdash_{\beta e}$).* □

Now, Strong Normalisation of $\vdash_{\beta\Pi e}$ is a consequence of that of $\vdash_{\beta e}$. First we change Π-redexes into λ-redexes.

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 lambda-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 $\widetilde{\pi_{x:A}} \equiv \lambda_{x:\widetilde{A}}$ and $\widetilde{(\pi_{x:A}.-)B} \equiv (\lambda_{x:\widetilde{A}}.-)\widetilde{B}$.*

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

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

Now suppose $\Gamma \vdash_{\beta\Pi e} Fa : (\Pi_{x:A}.B)a$ as a consequence of $\Gamma \vdash_{\beta\Pi e} F : \Pi_{x:A}.B$ and $\Gamma \vdash_{\beta\Pi e} a : A$. Then by the induction hypothesis $\widetilde{\Gamma} \vdash_{\beta e} \widetilde{F} : \Pi_{x:\widetilde{A}}.\widetilde{B}$ and $\widetilde{\Gamma} \vdash_{\beta e} \widetilde{a} : \widetilde{A}$, so by the application rule of $\vdash_{\beta e}$, $\widetilde{\Gamma} \vdash_{\beta e} \widetilde{F}\widetilde{a} : \widetilde{B}[x := \widetilde{a}]$.

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

But F cannot contain a Π-symbol which will mix with a in Fa to form a Π-redex. Otherwise, one can show by the generation lemma that $\Pi_{x:A}.B =_{def} S$ for some S. But this is impossible, hence $\widetilde{Fa} \equiv \widetilde{F}\widetilde{a}$. □

Theorem 4.17 *(Strong Normalisation for the Cube with respect to $\vdash_{\beta\Pi e}$ and $\to_{\beta\Pi}$)*
If A is a $\vdash_{\beta\Pi e}$-legal term then A is strongly normalising with respect to $\twoheadrightarrow_{\beta\Pi}$.

Proof: *If A is $\vdash_{\beta\Pi e}$-legal then $\widetilde{A}$ is $\vdash_{\beta e}$-legal by Lemma 4.16 and hence $\widetilde{A}$ is strongly normalising with respect to $\twoheadrightarrow_\beta$ (Theorem 4.14). But then also A is strongly normalising with respect to $\twoheadrightarrow_{\beta\Pi}$.* □

5 Comparing our system of definitions with other systems

In this section we will compare the type systems generated by $\vdash_{\beta e}$ with the one generated by $\vdash_\beta$ and with the type systems with definitions of Severi and Poll [SP 93]. First we prove a conservativity result saying that in a certain sense, definitions are harmless. That is, even though we can type more terms using $\vdash_{\beta e}$ than using $\vdash_\beta$, whenever a judgement is derivable in a theory $\mathcal{L}$ using definitions and $\vdash_{\beta e}$, it is also derivable in the theory $\mathcal{L}$ without definitions, using only $\vdash_\beta$ and where all the definitions are unfolded. Second, we say something about the effectiveness of derivations and type-checking of the extended systems. More work has to be done yet but we believe that there is a gain in using definitions.

5.1 Conservativity

In the following we shall write $\mathcal{L}_{\mathtt{def}}$ for any system $\mathcal{L}$ of the Cube, to denote the system $\mathcal{L}$ extended with definitions. We shall furthermore write $\vdash$ for $\vdash_\beta$.

Let us start by noting that all derivable judgements in a type system of the λ-cube are derivable in the same type system extended with definitions as we only extended, not changed, the derivation rules. In other words, if $\Gamma \vdash^{\mathcal{L}}_{\beta} A : B$ then $\Gamma \vdash^{\mathcal{L}}_{\beta e} A : B$.

The converse however is not true even if Γ contains no definitions. Look again at Example 4.3 where $\Gamma \vdash_{\beta e} A : B$, Γ contains no definitions yet $\Gamma \not\vdash A : B$. Hence, in the type systems with definitions there are more legal terms. Therefore, it has to be investigated to what extent the set of legal terms has changed.

The first noticeable change with definitions is the bypassing of the *formation rule* by using the *weakening* and *definition rule* instead. The following example shows that in $\lambda_{\to\mathtt{def}}$ we have more legal judgements than in $\lambda_\to$:

Example 5.1 *In $\lambda 2$ without definitions we can derive the following by using the formation rules $(*,*)$ and $(\Box,*)$ (take $\Gamma \equiv \lambda_{\beta:*}.\lambda_{y:\beta}$):*

$\Gamma \vdash^{\lambda 2} y : \beta : * : \Box$	
$\Gamma.\lambda_{\alpha:*} \vdash^{\lambda 2} \alpha : * : \Box$	*(start)*
$\Gamma.\lambda_{\alpha:*}.\lambda_{x:\alpha} \vdash^{\lambda 2} x : \alpha : *$	*(start resp weakening)*
$\Gamma.\lambda_{\alpha:*} \vdash^{\lambda 2} \Pi_{x:\alpha}.\alpha : *$	*(formation rule $(*,*)$)*
$\Gamma.\lambda_{\alpha:*} \vdash^{\lambda 2} \lambda_{x:\alpha}.x : \Pi_{x:\alpha}.\alpha$	*(abstraction)*
$\Gamma \vdash^{\lambda 2} \Pi_{\alpha:*}.\Pi_{x:\alpha}.\alpha : *$	*(formation rule $(\Box,*)$)*
$\Gamma \vdash^{\lambda 2} \lambda_{\alpha:*}.\lambda_{x:\alpha}.x : \Pi_{\alpha:*}.\Pi_{x:\alpha}.\alpha$	*(abstraction)*
$\Gamma \vdash^{\lambda 2} (\lambda_{\alpha:*}.\lambda_{x:\alpha}.x)\beta : \Pi_{x:\beta}.\beta$	*(application)*
$\Gamma \vdash^{\lambda 2} ((\lambda_{\alpha:*}.\lambda_{x:\alpha}.x)\beta)y : \beta$	*(application)*

It is not possible to derive this judgement in $\lambda_\to$ as the formation rule $(\Box,*)$ *is needed. Now we observe that the term $(\lambda_{\alpha:*}.-)\beta$ is a definition. Using this observation we can derive the judgement in a type system with definitions without having to use the formation rule $(\Box,*)$:*

$$\begin{array}{ll}
\Gamma \vdash^{\lambda_\rightarrow}_{\beta e} y : \beta : * : \Box & \\
\Gamma.(\lambda_{\alpha:*}.-)\beta \vdash^{\lambda_\rightarrow}_{\beta e} \alpha : * & \textit{(start)} \\
\Gamma.(\lambda_{\alpha:*}.-)\beta.\lambda_{x:\alpha} \vdash^{\lambda_\rightarrow}_{\beta e} x : \alpha : * & \textit{(start resp weakening)} \\
\Gamma.(\lambda_{\alpha:*}.-)\beta \vdash^{\lambda_\rightarrow}_{\beta e} \Pi_{x:\alpha}.\alpha : * & \textit{(formation rule } (*,*) \textit{)} \\
\Gamma.(\lambda_{\alpha:*}.-)\beta \vdash^{\lambda_\rightarrow}_{\beta e} \lambda_{x:\alpha}.x : \Pi_{x:\alpha}.\alpha & \textit{(abstraction)} \\
\Gamma \vdash^{\lambda_\rightarrow}_{\beta e} (\lambda_{\alpha:*}.\lambda_{x:\alpha}.x)\beta : (\Pi_{x:\alpha}.\alpha)[\alpha := \beta] \equiv \Pi_{x:\beta}.\beta & \textit{(definition)} \\
\Gamma \vdash^{\lambda_\rightarrow}_{\beta e} ((\lambda_{\alpha:*}.\lambda_{x:\alpha}.x)\beta)y : \beta & \textit{(application)}
\end{array}$$

The following example shows an even stronger statement. It shows that some terms which were only typable in the highest system of the Cube λC, become typable even in the lowest system $\lambda_\rightarrow$ in the presence of definitions:

Example 5.2 *Take a look at the judgement* $\Gamma \vdash (\lambda_{\alpha:*}.\lambda_{x:M}.x)\beta : \Pi_{x:M}.M$ *where* $M \equiv (\lambda_{z:\beta}.(\lambda_{\gamma:*}.\gamma)\beta)y$ *and* $\Gamma \equiv \lambda_{\beta:*}.\lambda_{y:\beta}$. *This judgement can be derived in* λC *using the formation rules* $(\Box,\Box)$, $(\Box,*)$, $(*,\Box)$ *and* $(*,*)$ *(here for simplicity,* $\vdash$ *stands for* $\vdash_\beta$ *rather than* $\vdash_{\beta\Pi}$*):*

$$\begin{array}{ll}
\Gamma \vdash^{\lambda C} \beta : * : \Box & \\
\Gamma.\lambda_{\alpha:*} \vdash^{\lambda C} \beta : * : \Box & \textit{(weakening)} \\
\Gamma.\lambda_{\alpha:*}.\lambda_{z:\beta} \vdash^{\lambda C} z : \beta : * : \Box & \textit{(start resp. weakening)} \\
\Gamma.\lambda_{\alpha:*}.\lambda_{z:\beta}.\lambda_{\gamma:*} \vdash^{\lambda C} \gamma : * : \Box & \textit{(start resp. weakening)} \\
\Gamma.\lambda_{\alpha:*}.\lambda_{z:\beta} \vdash^{\lambda C} \Pi_{\gamma:*}.* : \Box & \textit{(formation rule } (\Box,\Box) \textit{)} \\
\Gamma.\lambda_{\alpha:*}.\lambda_{z:\beta} \vdash^{\lambda C} \lambda_{\gamma:*}.\gamma : \Pi_{\gamma:*}.* & \textit{(abstraction)} \\
\Gamma.\lambda_{\alpha:*}.\lambda_{z:\beta} \vdash^{\lambda C} (\lambda_{\gamma:*}.\gamma)\beta : * & \textit{(application)} \\
\Gamma.\lambda_{\alpha:*} \vdash^{\lambda C} \Pi_{z:\beta}.* : \Box & \textit{(formation rule } (*,\Box) \textit{)} \\
\Gamma.\lambda_{\alpha:*} \vdash^{\lambda C} \lambda_{z:\beta}.(\lambda_{\gamma:*}.\gamma)\beta : \Pi_{z:\beta}.* & \textit{(abstraction)} \\
\Gamma.\lambda_{\alpha:*} \vdash^{\lambda C} M : * & \textit{(application, } M \equiv (\lambda_{z:\beta}.(\lambda_{\gamma:*}.\gamma)\beta)y \\
\Gamma.\lambda_{\alpha:*}.\lambda_{x:M} \vdash^{\lambda C} x : M : * & \textit{(start resp. weakening)} \\
\Gamma.\lambda_{\alpha:*} \vdash^{\lambda C} \Pi_{x:M}.M : * & \textit{(formation rule } (*,*) \textit{)} \\
\Gamma.\lambda_{\alpha:*} \vdash^{\lambda C} \lambda_{x:M}.x : \Pi_{x:M}.M & \textit{(abstraction)} \\
\Gamma \vdash^{\lambda C} \Pi_{\alpha:*}.\Pi_{x:M}.M : * & \textit{(formation rule } (\Box,*) \textit{)} \\
\Gamma \vdash^{\lambda C} \lambda_{\alpha:*}.\lambda_{x:M}.x : \Pi_{\alpha:*}.\Pi_{x:M}.M & \textit{(abstraction)} \\
\Gamma \vdash^{\lambda C} (\lambda_{\alpha:*}.\lambda_{x:M}.x)\beta : \Pi_{x:M}.M & \textit{(application)}
\end{array}$$

It is impossible to derive this judgement in any other system of the cube than λC *since all* *four formation rules are needed. We can however derive this judgement in* $\lambda_{\to\mathtt{def}}$*:*

$\Gamma \vdash_{\beta e}^{\lambda\to} \beta : * : \Box$	
$\Gamma.(\lambda_{\alpha:*}.-)\beta \vdash_{\beta e}^{\lambda\to} \beta : * : \Box$	*(weakening)*
$\Gamma(\lambda_{\alpha:*}.-)\beta.(\lambda_{z:\beta}.-)y \vdash_{\beta e}^{\lambda\to} \beta : * : \Box$	*(weakening)*
$\Gamma.(\lambda_{\alpha:*}.-)\beta.(\lambda_{z:\beta}.-)y.(\lambda_{\gamma:*}.-)\beta \vdash_{\beta e}^{\lambda\to} \gamma : *$	*(weakening)*
$\Gamma.(\lambda_{\alpha:*}.-)\beta.(\lambda_{z:\beta}.-)y \vdash_{\beta e}^{\lambda\to} (\lambda_{\gamma:*}.\gamma)\beta : *$	*(definition rule)*
$\Gamma.(\lambda_{\alpha:*}.-)\beta \vdash_{\beta e}^{\lambda\to} (\lambda_{z:\beta}.(\lambda_{\gamma:*}.\gamma)\beta)y : *$	*(definition rule)*
$\Gamma.(\lambda_{\alpha:*}.-)\beta.\lambda_{x:M} \vdash_{\beta e}^{\lambda\to} x : M : *$	*(start resp. weakening)*
$\Gamma.(\lambda_{\alpha:*}.-)\beta \vdash_{\beta e}^{\lambda\to} \Pi_{x:M}.M : *$	*(formation rule* $(*,*)$ *)*
$\Gamma.(\lambda_{\alpha:*}.-)\beta \vdash_{\beta e}^{\lambda\to} \lambda_{x:M}.x : \Pi_{x:M}.M$	*(abstraction)*
$\Gamma \vdash_{\beta e}^{\lambda\to} (\lambda_{\alpha:*}.\lambda_{x:M}.x)\beta : (\Pi_{x:M}.M)[\alpha := \beta] \equiv \Pi_{x:M}.M$	*(definition rule)*

This example shows that in every system of the λ-cube (except λC), adding definitions gives more derivable judgements. As was shown in Example 4.3, $\lambda_{\beta:*}.\lambda_{y:\beta} \vdash_{\beta e}^{\lambda\to} (\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)y)\beta : \beta$ is derivable in $\lambda_{\to\mathtt{def}}$ and hence is also derivable in $\lambda C_{\mathtt{def}}$, but this judgement cannot be derived in λC as y is of type β and not of type α. At first sight this might cause the reader to suspect type systems with definitions of having too much derivable judgements. However, we have a conservativity result stating that a judgement that can be derived in $\mathcal{L}_{\mathtt{def}}$ can be derived in $\mathcal{L}$ when all definitions in the whole judgement have been unfolded.

Definition 5.3 *For* $\Gamma \vdash_{\beta e} A : B$ *a judgement we define the unfolding of* $\Gamma \vdash_{\beta e} A : B$, $[\Gamma \vdash_{\beta e} A : B]^u$ *to be the judgement obtained from* $\Gamma \vdash_{\beta e} A : B$ *in the following way:*

- *first, mark all visible redexes in* Γ, A *and* B,
- *second, contract in* Γ, A *and* B *all these marked redexes.*

When $\Gamma \equiv \cdots(\lambda_{x:D}.-)C\cdots$, *contracting* $(\lambda_{x:D}.-)C$ *amounts to substituting all free occurrences of* x *in the scope of* λ_x *by* C*; these free occurrences may also be in one of the terms* A *and* B*. The result is independent of the order in which the redexes are contracted, as one can see this unfolding as a* complete development *(see [Barendregt 84]) in a certain sense.*

Example 5.4 $[\lambda_{\beta:*}.\lambda_{y:\beta}.(\lambda_{\alpha:*}.-)\beta.(\lambda_{x:\alpha}.-)y.\lambda_{z:\alpha} \vdash_{\beta e} (\lambda_{v:\Pi_{u:\alpha}.\beta}.vx)\lambda_{u:\alpha}.u : \alpha]^u$ is $\lambda_{\beta:*}.\lambda_{y:\beta}.(\lambda_{z:\alpha}[x := y][\alpha := \beta]) \vdash_{\beta e} ((vx)[v := \lambda_{u:\alpha}.u])[x := y][\alpha := \beta] : \alpha[x := y][\alpha := \beta]$, which is $\lambda_{\beta:*}.\lambda_{y:\beta}.\lambda_{z:\beta} \vdash_{\beta e} (\lambda_{u:\beta}.u)y : \beta$. Note that the resulting context contains only declarations of the form $\lambda_{x:A}$ and that the resulting subject and predicate need not be in normal form.

Theorem 5.5 *Let* $\mathcal{L}$ *be one of the systems of the Cube,* Γ *a context with definitions and* A, B *pseudoterms. If* $\Gamma \vdash_{\beta e}^{\mathcal{L}} A : B$ *then:*

1. $[\Gamma \vdash_{\beta e}^{\mathcal{L}} A : B]^u$
2. $\Gamma' \vdash_{\beta}^{\mathcal{L}} A' : B'$, *where* $[\Gamma \vdash_{\beta e}^{\mathcal{L}} A : B]^u \equiv \Gamma' \vdash_{\beta e}^{\mathcal{L}} A' : B'$.

Proof: *use induction on the derivation of* $\Gamma \vdash_{\beta e}^{\mathcal{L}} A : B$. Axiom, abstraction *and* formation rules *are easy, we treat the other cases.*

- *The last rule applied is the* start rule. *Then* $\Gamma.d \vdash^{\mathcal{L}}_{\beta e} \mathtt{subj}(d) : \mathtt{pred}(d)$ *as a consequence of* $\Gamma \prec d$. *Now if* $d \equiv \lambda_{x:A}$ *then by IH* $[\Gamma \vdash^{\mathcal{L}}_{\beta e} A : S]^u$ *and* $\Gamma' \vdash^{\mathcal{L}}_{\beta} A' : S$ *(S a sort, x fresh) so by the* start rule $\Gamma'.\lambda_{x:A'} \vdash^{\mathcal{L}}_{\beta} x : A'$ *and* $[\Gamma \vdash^{\mathcal{L}}_{\beta e} x : A]^u$ *On the other hand, if d is a definition, say* $d \equiv (\lambda_{x:B}.-)A$, *then by IH* $\Gamma' \vdash^{\mathcal{L}}_{\beta} A' : B' : S$ *(S a sort) and* $[\Gamma \vdash^{\mathcal{L}}_{\beta e} A : B]^u$, *and the unfolding of* $\Gamma.d \vdash^{\mathcal{L}}_{\beta e} \mathtt{subj}(d) : \mathtt{pred}(d)$ *is* $\Gamma' \vdash^{\mathcal{L}}_{\beta} \mathtt{def}(d)' : \mathtt{pred}(d)'$ *which is* $\Gamma' \vdash^{\mathcal{L}}_{\beta} A' : B'$ *so we are done.*

- *The last rule applied is the* weakening rule, *say* $\Gamma.d \vdash^{\mathcal{L}}_{\beta e} D : E$ *as a consequence of* $\Gamma \prec d$ *and* $\Gamma \vdash^{\mathcal{L}}_{\beta e} D : E$. *Because* $\mathtt{subj}(d)$ *is fresh we have that* $(\Gamma.d)' \vdash^{\mathcal{L}}_{\beta} D' : E'$ *is the same as* $\Gamma' \vdash^{\mathcal{L}}_{\beta} D' : E'$ *so by IH we are done.*

- *The last rule applied is the* application rule. *Then* $\Gamma \vdash^{\mathcal{L}}_{\beta e} Fa : B[x := a]$ *as a consequence of* $\Gamma \vdash^{\mathcal{L}}_{\beta e} F : \Pi_{x:A}.B$ *and* $\Gamma \vdash^{\mathcal{L}}_{\beta e} a : A$. *By IH and the* application rule *we get* $\Gamma' \vdash^{\mathcal{L}}_{\beta} F'a' : B'[x := a']$. *Now by subject reduction also* $\Gamma' \vdash^{\mathcal{L}}_{\beta} (F'a')' : B'[x := a']$. *If* $B'[x := a'] \equiv (B'[x := a'])'$ *then we are done, otherwise, by the Generation Corollary* $\Gamma' \vdash^{\mathcal{L}}_{\beta} B'[x := a'] : S$ *for some sort S, so by subject reduction* $\Gamma' \vdash^{\mathcal{L}}_{\beta} (B'[x := a'])' : S$ *and as* $B'[x := a'] =_{\beta} (B'[x := a'])'$ *by* conversion *we are done.*

- *The last rule applied is the* conversion rule. *Then* $\Gamma \vdash^{\mathcal{L}}_{\beta e} A : B_2$ *as a consequence of* $\Gamma \vdash^{\mathcal{L}}_{\beta e} A : B_1$, $\Gamma \vdash^{\mathcal{L}}_{\beta e} B_2 : S$ *and* $\Gamma \vdash^{\mathcal{L}}_{\beta e} B_1 =_{\mathtt{def}} B_2$. *Now* $\Gamma \vdash^{\mathcal{L}}_{\beta e} B_1 =_{\mathtt{def}} B_2$ *implies* $B_1' =_{\beta} B_2'$ *because if C results from D by locally unfolding a definition of* Γ *then* $C' \equiv D'$, *so the result follows by IH.*

- *The last rule applied is the* definition rule. *Then* $\Gamma \vdash^{\mathcal{L}}_{\beta e} (\lambda_{x:A}.C)B : [D]_d$ *as a consequence of* $\Gamma.d \vdash_{\beta} C : D$ *where* $d \equiv (\lambda_{x:A}.-)B$. *By IH,* $\Gamma' \vdash^{\mathcal{L}}_{\beta} [C']_d : [D']_d$ *which is the unfolding of* $\Gamma \vdash^{\mathcal{L}}_{\beta e} (\lambda_{x:A}.C)B : [D]_d$. *Here we wrote* $[D]_d$ *instead of* $D[\mathtt{subj}(d) := \mathtt{def}(d)]$

Remark 5.6 It is not sufficient in theorem 5.5 to unfold all the definitions in the context only. Look again at Example 5.1 to see that $\lambda_{\beta:*}.\lambda_{y:\beta} \vdash^{\lambda_{\to}}_{\beta e} ((\lambda_{\alpha:*}.\lambda_{x:\alpha}.x)\beta)y : \beta$, the context $\Gamma \equiv \lambda_{\beta:*}.\lambda_{y:\beta}$ contains no definitions and yet $\Gamma' \equiv \Gamma \not\vdash^{\lambda_{\to}}_{\beta} ((\lambda_{\alpha:*}.\lambda_{x:\alpha}.x)\beta)y : \beta$. The reason for this is that a redex in the subject may have been used to change the type when it was still in the context. Note that if all the definitions are unfolded in context, subject and predicate, then the judgement $\lambda_{\beta:*}.\lambda_{y:\beta} \vdash^{\beta}_{\lambda_{\to}} y : \beta$ is derivable.

5.2 Length of derivations and type checking

As can be noted from the examples in Section 5.1, derivations using the definition mechanism seem to need considerably less derivation steps to derive a judgement that can also be derived without definitions. This is mainly due to the fact that redexes in the term to be derived can be introduced by the *def rule* which bypasses the formation rule.

Type checking in the extended systems at first sight seems to be more difficult than in the systems of the λ-cube of Barendregt. Consider for instance the type-checking problem $\Gamma \vdash_{re} (\lambda_{\alpha:*}.P\alpha x)\sigma : ?$ where $\Gamma \equiv \lambda_{\sigma:\alpha}.\lambda_{P:\Pi_{\alpha:*}.\alpha \to *}.\lambda_{x:\sigma}$.

Note that this problem is not solvable in the non-extended systems, since $P\alpha : \alpha \to *$ and $x : \sigma$, so $P\alpha x$ is not typable. In the extended systems, the only thing a typechecking algorithm can do is trying to solve $\Gamma.(\lambda_{\alpha:*}.-)\sigma \vdash_{re} P\alpha x : ?$, which is equivalent to finding A, B, y such that $\begin{cases} \Gamma.(\lambda_{\alpha:*}.-)\sigma \vdash_{re} P\alpha : \Pi_{y:A}.B \\ \Gamma.(\lambda_{\alpha:*}.-)\sigma \vdash_{re} x : A \end{cases}$

which again is equivalent to finding z, C such that $\begin{cases} \Gamma.(\lambda_{\alpha:*}.-)\sigma \vdash_{re} P : \Pi_{z:C}.(\Pi_{x:A}.B) \\ \Gamma.(\lambda_{\alpha:*}.-)\sigma \vdash_{re} \alpha : C \\ \Gamma.(\lambda_{\alpha:*}.-)\sigma \vdash_{re} x : A \end{cases}$

Now $\Gamma.(\lambda_{\alpha:*}.-)\sigma \vdash_{re} P : \Pi_{\alpha:*}.\alpha \to \alpha$ and $\Gamma.(\lambda_{\alpha:*}.-)\sigma \vdash_{re} \alpha : *$, hence $\Gamma.(\lambda_{\alpha:*}.-)\sigma \vdash_{re} P\alpha : \alpha \to *$ and $\Gamma.(\lambda_{\alpha:*}.-)\sigma \vdash_{re} x : \sigma$.

Now we face the problem of converting $\alpha \to *$ to $\sigma \to *$ or σ to α in context $\Gamma.(\lambda_{\alpha:*}.-)\sigma$ and this is easily done by unfolding the definition $(\lambda_{\alpha:*}.-)\sigma$ in $\alpha \to *$, giving $\Gamma.(\lambda_{\alpha:*}.-)\sigma \vdash_{re} P\alpha : \sigma \to *$ and hence $\Gamma \vdash_{re} (\lambda_{\alpha:*}.P\alpha x)\sigma : *$.

We saw that typechecking gave rise to locally unfolding a definition in the type $\alpha \to *$. This is something new in comparison with typechecking in the λ-cube of Barendregt where only reduction to (weak head-) normal form is needed. Now if we want to typecheck a redex it appears to be a reasonable strategy to consider it as a definition since it is not easy to see whether a redex in a term can be typed without the (def rule).

So when typechecking $(\lambda x : \sigma.P\sigma xQ)t$ in our extended system with $\lambda_{P:\Pi_{\alpha:*}.\alpha\to\alpha\to*}.\lambda_{\sigma:*}.\lambda_{t:\sigma}$ being the context Γ, an automated type checker will try to solve

$\Gamma.(\lambda_{x:\sigma}.-)t \vdash_{re} P\sigma xQ : ?$

instead of $\begin{cases} \Gamma \vdash_{re} \lambda_{x:\sigma}.P\sigma xQ : \Pi_{x:\sigma}.A \\ \Gamma \vdash_{re} t : \sigma \end{cases}$

As a result, something like $\begin{cases} \Gamma.(\lambda_{x:\sigma}.-)t \vdash_{re} P\sigma x : \Pi_{y:A}.B \\ \Gamma.(\lambda_{x:\sigma}.-)t \vdash_{re} Q : A' \end{cases}$

will be derived and now it has to be checked whether $\Gamma.(\lambda_{x:\sigma}.-)t \vdash_{re} A =_{\mathtt{def}} A'$. In case the original redex was not a definition, $A =_{\mathtt{def}} A'$ can be established without using the context definition $(\lambda_{x:\sigma}.-)t$. Hence we conjecture that an intelligent typecheck algorithm can avoid needless extra work by unfolding definitions only as a last resort. Further research has yet to be done in this direction.

5.3 Comparison with the systems of the Barendregt cube

Here we discuss the (dis)advantages of our extended typing systems to the typing systems of the λ-cube.

In the extended typing systems we can reason with definitions in the context (which is very natural to do): we can add definitions to the context in which we reason (the *start rule* and *weakening rule*), we can eliminate definitions in the context (the *def rule*) and we can unfold a definition in the context locally in the type (the *conversion rule*).

If one considers one of the seven lower systems in the λ-cube, some abstractions are forbidden, for instance in $\lambda P\underline{\omega}$ the abstraction of a term over a type is not allowed (this abstraction corresponds to universal quantification in logic). Intuitively such a quantification need not be forbidden if it is immediately being instantiated by an application, as is the case in the term $(\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x))\beta$ in context $\lambda_{\beta:*}$. However, in the system $\lambda P\underline{\omega}$ this term is untypable.

Now in our extended typing system $\lambda P\underline{\omega}_{\mathtt{def}}$ we can type the term $(\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x))\beta$ by using the *def rule*: from $\lambda_{\beta:*}.(\lambda_{\alpha:*}.-)\beta \vdash_{\beta e} \lambda_{x:\alpha}.x : \Pi_{x:\alpha}.x$ we may conclude $\lambda_{\beta:*} \vdash_{\beta e} (\lambda_{\alpha:*}.\lambda_{x:\alpha}.x)\beta : \Pi_{x:\beta}.x$. Note that the use of the formation rule $(\Box, *)$ is avoided.

By this property, the extended type systems are closer to intuition than the systems of the λ-cube of Barendregt as there are more (intuitively correct) derivable inhabitants of certain types.

5.4 Comparison with the type systems of Severi and Poll

When we compare the extended type systems to those of Severi and Poll (see [SP 93]), we observe the following differences.

1. In the systems of [SP 93], the definition of pseudoterms has been adapted, not only the usual variables, abstractions and applications are pseudoterms, but definitions, i.e. terms of the form $x = a : A \texttt{ in } B$ are added. A new reduction relation has to be introduced to be able to unfold these definitions (locally in the predicate of a judgement). This means Church Rosser had to be shown again.

   In our approach, we treat definitions not much different from redexes, hence the syntax of pseudoterms remains the same. We only need to change the syntax of contexts and extend the notion of β-equality in a natural way to be able to use the definitions in the context and unfold them in the predicate of a judgement. Church Rosser remains unchanged.

2. [SP 93] have a rule that takes a definition out of the context and puts it in front of the term and type. In our extended system however, we only put the definition in front of the term and unfold it in the type. As we already noted in Remark 4.2, if the type is not $\Box$, it is a derived rule in our system that the definition need not be unfolded in the type.

3. [SP 93] do not demand the predicate of a definition to have some sort as type. This only leads to being able to abbreviate kinds, which is impossible in our extended systems. We consider this to be a minor disadvantage which might very well be easily overcome by leaving this demand.

5.5 Comparison with the generalised definitions

In [BKN 9y], we introduced a notion of generalised definitions which is similar to the one introduced here in that definitions are a kind of redexes which only occur in the context and can be unfolded via a def rule similar to the one presented here. In [BKN 9y] however, definitions were nested. That is, we could have $(\lambda_{y:B}.(\lambda_{x:A}.-)a)b$ and hence the def rule had to be changed to take this nesting into account. Such nesting however, is unnecessary for the reductions we are using in the present paper. In [BKN 9y], reduction was generalised due to the use of a useful 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. All that work on generalising reduction and nesting definitions is irrelevant to the present paper. It must be noted however that with nested definitions one can get yet shorter derivations due to the fact that many nested definitions may be treated as a single definition and hence the def rule will only be applied a single time. We should close here by saying that any definition in the sense of the present paper is also a definition in the sense of [BKN 9y] when the notation is changed. Furthermore any type derivation with definitions in this paper (not involving

$$\begin{tikzcd}
& C(\text{CR, SN, SR}) \arrow[dl] \arrow[dr] & \\
C_{\Pi}(\text{CR, SN}) \arrow[dr] & & C_{def}(\text{CR, SN, SR}) \arrow[dl] \\
& C_{\Pi,def}\ (\text{CR, SN, SR}) &
\end{tikzcd}$$

Figure 1: Properties of the Cube with various extensions

Π-reduction) is also a type derivation with definitions in [BKN 9y]. That is, if $\Gamma \vdash_{\beta e} A : B$ then $\mathcal{I}(\Gamma) \vdash_e \mathcal{I}(A) : \mathcal{I}(B)$ where $\vdash_e$ is the type derivation of [BKN 9y] and $\mathcal{I}$ translates terms to the notation of [KN 95a].

6 Conclusion

In this paper, we studied the addition of explicit definitions and Π-reduction to the Cube. In particular, we discussed various typing relations in the Cube, mainly the known relation $\vdash_\beta$, its extension with Π-reduction $\vdash_{\beta\Pi}$, its extension with definition $\vdash_{\beta e}$ and its extension with both Π-reduction and definitions $\vdash_{\beta\Pi e}$. Our addition of definitions (which are different in this paper from the existing notions of definition in the literature), is simple and worth studying. Furthermore, this addition enabled us to solve a problem we left open in [KN 95b] on Subject Reduction in the Cube with Π-conversion. There are many arguments why Π-reduction and explicit definitions must be considered and a system combining both of them without losing any of the nice properties of the cube is certainly worth considering. Moreover, we find it intreaguing that so far in the literature, definitions have been added for reasons of efficiency of implementation and not because they solve theoretical problems. In this paper, we have shown that definitions do indeed solve the theoretical problem of Subject Reduction in the extended version of the cube with Π-reduction. In [BKN 9y], we show that definitions solve the problem of Subject Reduction in the cube extended with a notion of *generalised reduction*. In a work in progress, we show that definitions solve the problem of type preservation for a certain operation on terms.

This is all puzzling as to why definitions really have that power. What definitions do however to solve these problems is that they keep information in the context on the defined values of some variables. This information might have been removed when some reductions in the term take place and so keeping the definition in the context preserves this information.

Hence, our paper contributes to other work on definitions not only in that it offers a simple and attractive account of definitions which keeps all the original properties of the cube, but also shows that definitions are theoretically important and should hence be introduced in the cube. Figure 1 summarizes our results in this paper.

7 Acknowledgements

We are grateful for discussions with Henk Barendregt, Thierry Coquand, Herman Geuvers, Erik Poll and Randy Pollack.

References

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

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

[BKN 9y] Bloo, R., Kamareddine, F., Nederpelt, R., The Barendregt Cube with Definitions and Generalised Reduction, Computing Science Note 94/08, University of Glasgow, Computing Science department, 1994.

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

[Geuvers 94] Geuvers, H., A short and flexible proof of Strong Normalization for the Calculus of Constructions, Computing Science Note 94/50, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1994.

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

[GM 93] Gordon M.J.C. and Melham, T.F. (eds), *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*, Cambridge University Press, 1993.

[KN 94] Kamareddine, F., and Nederpelt, R.P., Canonical Typing and Π-Conversion, Computing Science Note 94/02, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1994.

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

[KN 95b] Kamareddine, F., and Nederpelt, R.P., Canonical Typing and Π-Conversion in the Barendregt Cube, *Journal of Functional Programming*, 1995.

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

[NGV 95] R.P. Nederpelt, J.H. Geuvers, and R.C. de Vrijer, eds., *Selected Papers on Automath* , North-Holland, 1994.