The lambda-cube with classes of terms modulo conversion

Citation for published version (APA):

Document status and date:
Published: 01/01/1994

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 11. Nov. 2019
The $\lambda$-cube with classes of terms modulo conversion

by

R. Bloo, F. Kamareddine and R. Nederpelt

94/46
On II-conversion in Type Theory*†

Roel Bloor‡ Fairouz Kamareddine § and Rob Nederpelt ¶

October 7, 1994

1 Introduction

Type theory has almost always been studied without II-conversion (which is the analogue of β-conversion on product type level). That is, \((\lambda_{x:A}.b)C \rightarrow_{\beta} b[x := C]\) is always assumed but not \((\Pi_{x:A}.B)C \rightarrow_{\Pi} B[x := C]\). The exception for this are some Automath languages in [de Bruijn 74] and the current work of [KN 94] and [KN 9x].

Moreover, the conversion rule of the usual typing relation of the systems of the λ-Cube \(\vdash_{\beta}\), is given in terms of \(=_{\beta}\) rather than \(=_{\beta\Pi}\) and the application axiom is as follows:

\[
\frac{\Gamma \vdash_{\beta} F : \Pi_{x:A}.B \quad \Gamma \vdash_{\beta} a : A}{\Gamma \vdash_{\beta} Fa : B[x := a]}
\]

The fact that the type of \(Fa\) is taken to be \(B[x := a]\) rather than \((\Pi_{x:A}.B)a\), implies the loss of the compatibility property for the typing of application. So whereas in the λ-Cube one has compatibility for the typing of abstraction (i.e. \(A : B \Rightarrow \lambda_{x:C}.A : \Pi_{x:C}.B\)), one does not have \(A : B \Rightarrow AC : BC\).

The extended metatheory of the systems of the λ-Cube with II-conversion was first studied in detail in [KN 94] and [KN 9x]. The aim of that study was to create a canonical typing operator which can easily find the type of a term, and to make the judgement \(A\) is typable in \(\Gamma\) independent of the type of \(A\). That is, the aim was to split \(\Gamma \vdash A : B\) into \(\Gamma \vdash A\) and \(\tau(\Gamma, A) =_{\Pi\Pi} B\) where \(\tau(A)\) may indeed contain II-redexes. In fact, the \(\Gamma\)-canonical type of a term \(A\), \(\tau(\Gamma, A)\), was almost an exact copy of \(A\) where the heart of \(A\) is replaced by its type in \(\Gamma\), the II's are removed and the \(\lambda\)-redexes are changed to II-redexes. For example:

\[
\tau(\Pi_{x:A}.(\lambda_{y:B}.(\lambda_{z:C}.x)y)(\Pi_{w:D}.(\lambda_{z:E}.x)y)) \equiv
\]

\[
(\Pi_{y:B}.(\Pi_{z:C}.x)y)(\Pi_{w:D}.(\lambda_{z:E}.x)y)
\]
In [KN 9x], II-reduction was needed as a relation between the canonical type of a pseudo-term and any other of its types. It was shown however, that including II-reduction in the typing relation of the Cube is rather redundant for the splitting of $\Gamma \vdash A : B$ in two parts. In fact, if $\vdash_{\beta\Pi}$ is the relation which is defined as $\vdash_{\beta}$ but where the conversion rule uses $=_{\beta\Pi}$ rather than $=_{\beta}$ and where the application rule takes the type of $Fa$ to be $(\Pi x : A).B)a$ rather than $B[x := a]$, then it was shown that $\Gamma \vdash_{\beta} A : B \iff \Gamma \vdash A \land \tau(\Gamma, A) =_{\beta\Pi} B \land B$ is a $\vdash_{\beta}$-legal type. Hence, really $\vdash_{\beta\Pi}$ is not needed yet $=_{\beta\Pi}$ is as it compares the canonical type of $A$ (which may contain many II-redexes) with the $\vdash_{\beta}$-type of $A$.

A further result was shown in [KN 9x]. That is: if $\Gamma \vdash_{\beta\Pi} A : B$ then $\Gamma, A$ contain no II-redexes and $B$ is either II-redex free or itself the unique II-redex in $B$. This means that $\vdash_{\beta\Pi}$ has almost the same legal terms as $\vdash_{\beta}$. We do however, think that the small change in legal types which allows them now to be themselves II-redexes, and the elegance that results from writing $\Gamma \vdash_{\beta\Pi} Fa : (\Pi x : A).B)a$ rather than $B[x := a]$ in the application rule are worth studying.

Now we come to the heart of this paper. In [KN 9x], it was shown that Subject Reduction (SR) fails for $\vdash_{\beta\Pi}$ (that is: $\Gamma \vdash_{\beta\Pi} A : B \land A \rightarrow_{\beta\Pi} A' \not\propto \Gamma \vdash_{\beta\Pi} A' : B$) yet holds if $B$ is $\vdash_{\beta}$-legal. The failure of SR was a result of the failure of type correctness. In fact, one may have $\Gamma \vdash_{\beta\Pi} A : B$ without having $B \equiv \Box$ or $\Gamma \vdash_{\beta\Pi} B : S$ for some $S$. We find this failure of type correctness annoying and once we establish it, we can re-establish SR. In this paper, we shall show that adding definitions to the Cube re-establishes type correctness and full Subject Reduction for $\vdash_{\beta\Pi}$. This is quite intriguing. In fact, most implementations of Pure Type Systems such as Coq ([Dow 91]), Lego ([LP 92]) and HOL ([GM 93]) include definitions. Furthermore, our work on generalising reduction in the Cube in [BKN 9y] showed that definitions are necessary. Now, again, definitions play a role in II-reduction.

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

1. In section 2, we introduce the formal machinery of the cube extended with definitions and introduce $\vdash_{\beta\Pi}$ as in [KN 9x] observing that correctness of types and subject reduction do not hold for $\vdash_{\beta\Pi}$.

2. In section 3, we introduce $\vdash_{\beta\Pi e}$ which is $\vdash_{\beta\Pi}$ extended with definitions and show that now, correctness of types and subject reduction hold for $\vdash_{\beta\Pi e}$. We show further that all other properties of the cube remain valid for $\vdash_{\beta\Pi e}$.

2 The Cube with II-reduction

The systems of the Cube (see [Barendregt 92]), are based on a set of pseudo-expressions or terms $T$ defined by the following abstract syntax:

$$T = \ast | \Box | V | TT | \pi V : T . T$$

where $\pi$ ranges over $\Pi$ and $\lambda, V$ is an infinite collection of variables over which $\alpha, \beta, x, y, z, \ldots$ range. $\ast$ 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 $T$.

Bound and free variables and substitution are defined as usual. 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 \( \lambda \)'s have different variables as subscript. Hence, we will not have \( (\lambda x:A.x)x \), but \( (\lambda y:A.y)x \) instead.

Terms can be related via a reduction relation \( \rightarrow_{r} \). We assume the usual definition of the compatibility of a reduction relation, and define \( \rightarrow_{r} \) to be its reflexive transitive closure and \( =_{r} \) to be its equivalence closure. We use in this paper two reduction relations: \( \rightarrow_{\beta} \) generated by the axiom \( (\lambda x:A.B)C \rightarrow_{\beta} B[x := C] \) and \( \rightarrow_{\beta\Pi} \) generated by the axiom \( (\pi x:A.B)C \rightarrow_{\beta\Pi} B[x := C] \) (remember that \( \pi \) ranges over both \( \lambda \) and \( \Pi \)).

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

1. A declaration \( d \) is of the form \( \lambda x:A \). We define \( \text{subj}(d) \) and \( \text{pred}(d) \) 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 \( \text{subj}(d) \), \( \text{pred}(d) \) and \( \text{def}(d) \) 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 \( \Gamma \) is a (possibly empty) concatenation of declarations and definitions \( d_1, d_2, \ldots, d_n \) such that if \( i \neq j \), then \( \text{subj}(d_i) \neq \text{subj}(d_j) \). We use \( \Gamma, \Delta, \Delta', \Gamma_1, \Gamma_2, \ldots \) to range over pseudocontexts.

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

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.
   - \( \Gamma.\lambda x:A.\Delta \subseteq' \Gamma.(\lambda x:A.-)B.\Delta \)

**Remark 2.3** We only consider definitions when the reduction relation is \( \rightarrow_{\beta\Pi} \). Then \( \lambda \)-redexes and \( \Pi \)-redexes have —intuitively— equal meanings, therefore we allow definitions to be \( \Pi \)-redexes as well as \( \lambda \)-redexes.

**Definition 2.4 (statements, judgements, \( \preceq \))**

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 \( \Gamma \) is a pseudocontext and \( A : B \) is a statement, we call \( \Gamma \vdash A : B \) a judgement, and write \( \Gamma \vdash A : B : C \) to mean \( \Gamma \vdash A : B \wedge \Gamma \vdash B : C \).

3. For \( \Gamma \) a pseudocontext and \( d \in \Gamma\text{-def} \cup \Gamma\text{-decl} \), we say \( \Gamma \) invites \( d \), notation \( \Gamma \prec d \), iff
• \( \Gamma \cdot d \) is a pseudocontext
• \( \Gamma \vdash \text{pred}(d) : S \) for some sort \( S \).
• if \( d \) is a definition then \( \Gamma \vdash \text{def}(d) : \text{pred}(d) \)

**Definition 2.5** (Definitional \( r \)-equality) Let \( \approx_r \) be one of the relations \( \approx_{\beta} \), \( \approx_{\beta_1} \). For all pseudocontexts \( \Gamma \) we define the binary relation \( \Gamma \vdash \cdot \approx_{\beta} \cdot \) to be the equivalence relation generated by

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

**Remark 2.6** If no definitions are present in \( \Gamma \) then \( \Gamma \vdash A =_{\text{def}} B \) is the same as \( A =_r B \).

**Definition 2.7** Let \( \Gamma \) be a pseudocontext.

1. Let \( d, d_1, \ldots, d_n \) be declarations and definitions. We define \( \Gamma \vdash d \) and \( \Gamma \vdash d_1 \cdots d_n \) simultaneously as follows:
   - If \( d \) is a declaration: \( \Gamma \vdash d \) iff \( \Gamma \vdash \text{subj}(d) : \text{pred}(d) \).
   - If \( d \) is a definition: \( \Gamma \vdash d \) iff \( \Gamma \vdash \text{subj}(d) : \text{pred}(d) \land \Gamma \vdash \text{def}(d) : \text{pred}(d) \land \Gamma \vdash \text{subj}(d) =_{\text{def}} \text{def}(d) \).
   - \( \Gamma \vdash d_1 \cdots d_n \) iff \( \Gamma \vdash d_i \) for all \( 1 \leq i \leq n \).

2. \( \Gamma \) is called legal if \( \exists P, Q \in T \) such that \( \Gamma \vdash P : Q \).

3. \( A \in T \) is called a \( \Gamma \)-term if \( \exists B \in T[\Gamma \vdash A : B \lor \Gamma \vdash B : A] \).
   We take \( \Gamma \)-terms = \( \{ A \in T \mid \exists B \in T[\Gamma \vdash A : B \lor \Gamma \vdash B : A] \} \).
   \( A \in T \) is called legal if \( \exists \Gamma[A \in \Gamma \cdot \text{terms}] \).

In the Cube as presented in [Barendregt 92], the only declarations allowed are of the form \( \lambda x : A \). Hence there are no definitions. Therefore, \( \Gamma \vdash d \) is of the form \( \Gamma \vdash \lambda x : A \) and means that \( \Gamma \vdash A : S \) for some \( S \) and that \( x \) is fresh in \( \Gamma, A \). Moreover, for any \( d \equiv \lambda x : A \), remember that \( \text{subj}(d) \equiv x \) and \( \text{pred}(d) \equiv A \). Moreover, II-reduction is not allowed. Hence, in the following definition, \( d \) is a meta-variable for declarations only and \( \approx_{\text{def}} \) is the same as \( \approx_{\beta} \) (which is independent of \( \vdash \)).

**Definition 2.8** (Axioms and rules of the Cube: \( d \) is a declaration, \( \approx_{\text{def}} \) is \( \approx_{\beta} \))

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

<table>
<thead>
<tr>
<th>System</th>
<th>Set of specific rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\lambda_\ast)</td>
<td>((\ast, \ast)) (□, *), (\ast, □)</td>
</tr>
<tr>
<td>(\lambda 2)</td>
<td>((\ast, \ast)) (□, *), (\ast, □)</td>
</tr>
<tr>
<td>(\lambda P)</td>
<td>((\ast, \ast)) (□, *), (\ast, □)</td>
</tr>
<tr>
<td>(\lambda P2)</td>
<td>((\ast, \ast)) (□, *), (\ast, □)</td>
</tr>
<tr>
<td>(\lambda \omega)</td>
<td>((\ast, \ast)) (□, *), (\ast, □)</td>
</tr>
<tr>
<td>(\lambda P\omega)</td>
<td>((\ast, \ast)) (□, *), (\ast, □)</td>
</tr>
<tr>
<td>(\lambda P\omega = \lambda C)</td>
<td>((\ast, \ast)) (□, *), (\ast, □)</td>
</tr>
</tbody>
</table>

[KN 9x] extended the above Cube by changing \(\rightarrow_\beta\) to \(\rightarrow_\beta \Pi\) and by changing \(\vdash_\beta\) to \(\vdash_\beta \Pi\):

**Definition 2.9 (\(\vdash_\beta \Pi\))** We define \(\vdash_\beta \Pi\) as \(\vdash_\beta\) with the difference that the application and conversion rules change as follows:

\[(\text{new application rule})\]
\[
\Gamma \vdash_\beta \Pi F : \Pi \chi A : B \quad \Gamma \vdash_\beta A : A \quad \Gamma \vdash_\beta \Pi Fa : (\Pi \chi A \cdot B) a
\]

\[(\text{new conversion rule})\]
\[
\Gamma \vdash_\beta \Pi A : B \quad \Gamma \vdash_\beta \Pi B' : S \quad \Gamma \vdash_\beta \Pi B = \text{det} B'
\]

Note that \(\Gamma \vdash_\beta \Pi B = \text{det} B'\) is the same as \(B = \beta \Pi B'\), as no definitions are allowed in the context.

Now we list the properties of \(\vdash_\beta \Pi\) without proofs (see [KN 9x]).

**Theorem 2.10 (The Church Rosser Theorem CR, for \(\rightarrow_\beta \Pi\))**
If \(A \rightarrow_\beta \Pi B\) and \(A \rightarrow_\beta \Pi C\) then there exists \(D\) such that \(B \rightarrow_\beta \Pi D\) and \(C \rightarrow_\beta \Pi D\). □

**Lemma 2.11 (Free variable lemma for \(\vdash_\beta \Pi\))**
Let \(\Gamma\) be a \(\vdash_\beta \Pi\)-legal context such that \(\Gamma \vdash_\beta \Pi B : C\). Then we have:

1. For all \(d, d' \in \Gamma\)-decl, \(\text{subj}(d) \neq \text{subj}(d')\).
2. \(FV(B), FV(C) \subseteq \text{dom}(\Gamma)\).
3. If $\Gamma \equiv \Gamma_1 \cdot \Gamma_2$ then $FV(d) \subseteq dom(\Gamma_1)$. □

Lemma 2.12 (Start Lemma for $\vdash_{\beta \Pi}$)
Let $\Gamma$ be a $\vdash_{\beta \Pi}$-legal context. Then $\Gamma \vdash_{\beta \Pi} * : \Box$ and $\forall d \in \Gamma[\Gamma \vdash_{\beta \Pi} d]$. □

Lemma 2.13 (Transitivity Lemma for $\vdash_{\beta \Pi}$)
Let $\Gamma$ and $\Delta$ be $\vdash_{\beta \Pi}$-legal contexts. Then: $[\Gamma \vdash_{\beta \Pi} \Delta \land \Delta \vdash_{\beta \Pi} A : B] \Rightarrow \Gamma \vdash_{\beta \Pi} A : B$. □

Lemma 2.14 (Substitution Lemma for $\vdash_{\beta \Pi}$)
If $\Gamma \cdot \lambda x : A . B \vdash_{\beta \Pi} C$ and $\Gamma \vdash_{\beta \Pi} D : A$ then $\Gamma. (\Delta[x := D]) \vdash_{\beta \Pi} B[x := D] : C[x := D]$. □

Lemma 2.15 (Thinning Lemma for $\vdash_{\beta \Pi}$)
Let $\Gamma$ and $\Delta$ be $\vdash_{\beta \Pi}$-legal contexts such that $\Gamma \vdash_{\beta \Pi} A : B$.

Lemma 2.16 (Generation Lemma for $\vdash_{\beta \Pi}$)
1. $\Gamma \vdash_{\beta \Pi} S : C \Rightarrow S \vdash_{\beta \Pi} C =_{\Pi} S'$ for some sort $S'$.
2. $\Gamma \vdash_{\beta \Pi} x : C \Rightarrow \exists B =_{\beta \Pi} C [\lambda x : B \in \Gamma \land if C \neq B then \Gamma \vdash_{\beta \Pi} C : S for some sort S]$.
3. $\Gamma \vdash_{\beta \Pi} \Pi_{x : A . B} : C \Rightarrow \exists (S_1, S_2)[\Gamma \vdash_{\beta \Pi} A : S_1 \land \Gamma. \lambda x : A \vdash_{\beta \Pi} B : S_2 \land (S_1, S_2) \text{ is a rule} \land C =_{\beta} S_2 \land [C \neq S_2 \Rightarrow \exists S[\Gamma \vdash_{\beta \Pi} C : S]]$.
4. $\Gamma \vdash_{\beta \Pi} \lambda x : A . B : C \Rightarrow \exists (S, B)[\Gamma \vdash_{\beta \Pi} \Pi_{x : A . B} : S \land \Gamma. \lambda x : A \vdash_{\beta \Pi} B : B \land C =_{\beta} \Pi_{x : A . B} \land [C \neq \Pi_{x : A . B} \Rightarrow \exists S'[\Gamma \vdash_{\beta \Pi} C : S']]$.
5. $\Gamma \vdash_{\beta \Pi} F a : C \Rightarrow \exists A, B, x[\Gamma \vdash_{\beta \Pi} F : \Pi_{x : A . B} \land \Gamma \vdash_{\beta \Pi} a : A \land C =_{\beta} (\Pi_{x : A . B})a \land ([\Pi_{x : A . B}]a \neq C \Rightarrow \exists S'[\Gamma \vdash_{\beta \Pi} C : S'])]$. □

Remark 2.17 (Correctness of types does not hold for $\vdash_{\beta \Pi}$)
The new legal terms of the form $(\Pi_{x : A . B} : C)$ imply the failure of type correctness for $\vdash_{\beta \Pi}$. That is, even in $\lambda_{a_\rightarrow}, \Gamma \vdash_{\beta \Pi} A : B \neq (B \equiv \Box \lor \Box \vdash_{\beta \Pi} B : S$ for some sort $S$). For example, if $\Gamma \equiv \lambda x_{a_\rightarrow} \cdot \lambda x_{a_\rightarrow} \vdash_{\beta \Pi} \lambda y_{a_\rightarrow} . \lambda x_{a_\rightarrow} \vdash_{\beta \Pi} y_{a_\rightarrow} x : (\Pi_{x_{a_\rightarrow} . x} : S)$, or $\Gamma \vdash_{\beta \Pi} (\Pi_{x_{a_\rightarrow} . x} : S)$ from Lemma 2.19. Type correctness of course holds for $\vdash_{\beta}$.

Failure of correctness of types implies failure of Subject Reduction even in $\lambda_{a_\rightarrow}$:

Example 2.18 In $\lambda_{a_\rightarrow}, \lambda x_{a_\rightarrow} \vdash_{\beta \Pi} x : (\Pi_{y_{a_\rightarrow} . x} : x)$. Otherwise, by generation: $\lambda x_{a_\rightarrow} \cdot \lambda x_{a_\rightarrow} \vdash_{\beta \Pi} (\Pi_{y_{a_\rightarrow} . x} : x), \lambda x_{a_\rightarrow} \vdash_{\beta \Pi} (\Pi_{y_{a_\rightarrow} . x} : x) \vdash_{\beta} (\Pi_{y_{a_\rightarrow} . x} : x)$. We do have however, a weak subject reduction which we will prove after we show the relationship between $\vdash_{\beta \Pi}$ and $\vdash_{\beta}$.

Lemma 2.19 For any $A, B, C, S, \Gamma$: $\Gamma \vdash_{\beta \Pi} (\Pi_{x : A . B} : C) : S$. □

We do have the following lemma which is a sort of weak generation corollary:

Lemma 2.20 $\Gamma \vdash_{\beta \Pi} A : B \land B$ is not a $\Pi$-redex $\Rightarrow (B \equiv \Box \lor \exists S[\Gamma \vdash_{\beta \Pi} B : S])$. □

Lemma 2.21 (Legal terms and contexts for $\vdash_{\beta \Pi}$ and $\vdash_{\beta}$)
1. If $\Gamma \vdash_{\beta \Pi} A : B$ then $A$ and $\Gamma$ are free of $\Pi$-redexes, and either $B$ contains no $\Pi$-redexes or $B$ is the only $\Pi$-redex in $B$. □
2. If \((\Pi x:D.E)B\) is \(\vdash_{\beta_{\Pi}}\)legal, then \(E[x := B]\) contains no \(\Pi\)-redexes.

To relate \(\vdash_{\beta}\) and \(\vdash_{\beta_{\Pi}}\), we introduce a notation which removes the unique \(\Pi\)-redex in a \(\vdash_{\beta_{\Pi}}\)legal term (if it exists):

**Definition 2.22** For \(A : B\) \(\vdash_{\beta_{\Pi}}\)-legal, let \(\hat{A}\) be \(C[x := D]\) if \(A \equiv (\Pi x:B.C)D\) and \(A\) otherwise.

**Lemma 2.23**

1. If \(\Gamma \vdash_{\beta} A : B\) then \(\Gamma \vdash_{\beta} A : \hat{B}\).

2. If \(\Gamma \vdash_{\beta} A : B\) then \(\Gamma \vdash_{\beta_{\Pi}} A : B\).

**Lemma 2.24** (Weak Subject Reduction for \(\vdash_{\beta_{\Pi}}\) and \(\rightarrow_{\beta_{\Pi}}\))

\[\Gamma \vdash_{\beta_{\Pi}} A : B \wedge A \rightarrow_{\beta_{\Pi}} A' \Rightarrow \Gamma \vdash_{\beta_{\Pi}} A' : \hat{B}\]

**Corollary 2.25** (WSR Corollary for \(\vdash_{\beta_{\Pi}}\) and \(\rightarrow_{\beta_{\Pi}}\))

If \(\Gamma \vdash_{\beta_{\Pi}} A : B_1\) and \(B_1 \rightarrow_{\beta_{\Pi}} B_2\) then \(\Gamma \vdash_{\beta_{\Pi}} A : B_2\).

**Remark 2.26** We cannot replace Corollary 2.25 by: If \(\Gamma \vdash_{\beta_{\Pi}} A : B\) and \(B \rightarrow_{\beta_{\Pi}} B'\) then \(\Gamma \vdash_{\beta_{\Pi}} A : B'\). For example, take \(\Gamma \equiv \lambda z:A.\lambda y:A, A \equiv (\lambda z:A)z\), \(B \equiv (\Pi x:A)z\) and \(B' \equiv (\Pi x:A)z\). Then, \(\Gamma \vdash_{\beta_{\Pi}} A : B\) but \(\Gamma \vdash_{\beta_{\Pi}} A : B'\) because if otherwise, we get by generation, \(\Gamma \vdash_{\beta_{\Pi}} (\Pi x:A)y : S\), absurd by Lemma 2.19.

**Lemma 2.27** (Unicity of Types for \(\vdash_{\beta_{\Pi}}\) and \(\rightarrow_{\beta_{\Pi}}\))

\[\Gamma \vdash_{\beta_{\Pi}} A : B_1 \wedge \Gamma \vdash_{\beta_{\Pi}} A : B_2 \Rightarrow B_1 =_{\beta_{\Pi}} B_2\]

**Theorem 2.28** (Strong Normalisation with respect to \(\vdash_{\beta_{\Pi}}\) and \(\rightarrow_{\beta_{\Pi}}\))

If \(A\) is \(\vdash_{\beta_{\Pi}}\)-legal then \(SN_{\beta_{\Pi}}(A)\); i.e. \(A\) is strongly normalising with respect to \(\rightarrow_{\beta_{\Pi}}\).

## 3 Extending \(\vdash_{\beta_{\Pi}}\) with definitions

We shall extend the derivation rules of \(\vdash_{\beta_{\Pi}}\) 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 \(\Gamma \vdash B =_{\beta_{\Pi}} B'\) in the conversion rule really has an effect now, rather than simply postulating \(B =_{\beta_{\Pi}} B'\).

- Not only declarations but also definitions are allowed in contexts.

**Definition 3.1** (Axioms and rules of the Cube extended with definitions: \(d\) ranges over declarations and definitions)

We extend the relation \(\vdash_{\beta_{\Pi}}\) to \(\vdash_{\beta_{\Pi}e}\) by adding the following definition rule:

\[
\text{(def rule)} \quad \frac{\Gamma, (\pi x:A)B \vdash_{\beta_{\Pi}e} C : D}{\Gamma \vdash_{\beta_{\Pi}e} (\pi x:A,C)B : D[x := B]}
\]
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 \( D \). We do not get type \((\pi_{x:A.D})B\) in order to avoid things like \((\pi_{x:A.-})B\). Note that the (def rule) does global substitution in the predicate of all the occurrences of \( \text{subj}(d) \). The reason is that \( d \) no longer remains in the context. In the conversion rule however, substitution is local as \( \Gamma \) keeps all its information (see Definition 2.5).

Remark 3.2 Our approach to definitions is slightly different from that of [SP 93], for details we refer to [BKN 9y].

Lemma 3.3 (Free variable lemma for \( \beta_{\Pi_e} \))
Let \( \Gamma \) be a legal context such that \( \Gamma \vdash_{\beta_{\Pi_e}} B : C \). Then the following holds:

1. If \( d \) and \( d' \) are two different elements of \( \Gamma \)-\text{decl} \cup \Gamma \text{-} \text{def} \), then \( \text{subj}(d) \neq \text{subj}(d') \).
2. \( \text{FV}(B), \text{FV}(C) \subseteq \text{dom}(\Gamma) \).
3. If \( \Gamma \equiv \Gamma_1.d.\Gamma_2 \) then \( \text{FV}(d) \subseteq \text{dom}(\Gamma_1) \).

Proof: All by induction on the derivation of \( \Gamma \vdash_{\beta_{\Pi_e}} B : C \).

Lemma 3.4 (Start Lemma for \( \beta_{\Pi_e} \))
Let \( \Gamma \) be a legal context. Then \( \Gamma \vdash_{\beta_{\Pi_e}} * : * \) and \( \forall d \in \Gamma[\Gamma \vdash_{\beta_{\Pi_e}} d] \).

Proof: \( \Gamma \) legal \( \Rightarrow \exists B, C[\Gamma \vdash_{\beta_{\Pi_e}} B : C] \); now use induction on \( \Gamma \vdash_{\beta_{\Pi_e}} B : C \).

Lemma 3.5 (Transitivity Lemma for \( \beta_{\Pi_e} \))
Let \( \Gamma \) and \( \Delta \) be legal contexts. Then: \([\Gamma \vdash_{\beta_{\Pi_e}} \Delta \land \Delta \vdash_{\beta_{\Pi_e}} A : B] \Rightarrow \Gamma \vdash_{\beta_{\Pi_e}} A : B \).

Proof: Induction on the derivation \( \Delta \vdash_{\beta_{\Pi_e}} A : B \).

Lemma 3.6 (Thinning for \( \beta_{\Pi_e} \))
1. If \( \Gamma_1.\Gamma_2 \vdash_{\beta_{\Pi_e}} A =_{\text{def}} B \), \( \Gamma_1.\Delta.\Gamma_2 \vdash_{\beta_{\Pi_e}} A =_{\text{def}} B \).
2. If \( \Gamma \) and \( \Delta \) are legal contexts such that \( \Gamma \subseteq \Delta \) and if \( \Gamma \vdash_{\beta_{\Pi_e}} A : B \), then \( \Delta \vdash_{\beta_{\Pi_e}} A : B \).

Proof: 1. is by induction on the derivation \( \Gamma_1.\Gamma_2 \vdash_{\beta_{\Pi_e}} A =_{\text{def}} B \). 2. is as follows:

- If \( \Gamma.\Delta \vdash_{\beta_{\Pi_e}} A : B \), \( \Gamma \vdash_{\beta_{\Pi_e}} C : S_x \) x is fresh, then also \( \Gamma.\lambda_{x:C}.\Delta \vdash_{\beta_{\Pi_e}} A : B \). We show this by induction on the \( \Delta \vdash_{\beta_{\Pi_e}} A : B \) using 1. for conversion.
- If \( \Gamma.\Delta \vdash_{\beta_{\Pi_e}} A : B \), \( \Gamma \vdash_{\beta_{\Pi_e}} C : D : S_x \) x is fresh, then also \( \Gamma.(\pi_{x:D.-})C.\Delta \vdash_{\beta_{\Pi_e}} A : B \). We show this by induction on the \( \Delta \vdash_{\beta_{\Pi_e}} A : B \).
- If \( \Gamma.\lambda_{x:A}.\Delta \vdash_{\beta_{\Pi_e}} B : C \), \( \Gamma \vdash_{\beta_{\Pi_e}} D : A \), then \( \Gamma.(\lambda_{x:A.-})D.\Delta \vdash_{\beta_{\Pi_e}} B : C \) is shown by induction on the \( \Delta \vdash_{\beta_{\Pi_e}} B : C \) (for conversion, use 1.).

Lemma 3.7 (Substitution lemma for \( \beta_{\Pi_e} \))
1. If \( \Gamma.(\pi_{x:C.-})D.\Delta \vdash_{\beta_{\Pi_e}} A =_{\text{def}} B \), \( A \) and \( B \) are \( \Gamma.(\pi_{x:C.-})D.\Delta \)-legal, then \( \Gamma.\Delta[x := D] \vdash_{\beta_{\Pi_e}} A[x := D] =_{\text{def}} B[x := D] \).
2. If \( B \) is a \( \Gamma.(\pi_{x:C.-})D \)-legal term, then \( \Gamma.(\pi_{x:C.-})D \vdash_{\beta_{\Pi_e}} B =_{\text{def}} B[x := D] \).
3. If \( \Gamma.(\pi_{x:A}.--)B.\Delta \vdash \beta\Pi C : D \), then \( \Gamma.\Delta[x := B] \vdash \beta\Pi C[x := B] : D[x := B] \).

4. If \( \Gamma.\lambda_{x:A}.\Delta \vdash \beta\Pi C : D \), \( \Gamma \vdash \beta\Pi B : A \), then \( \Gamma.\Delta[x := B] \vdash \beta\Pi C[x := B] : D[x := B] \).

Proof:

1. Induction on the derivation rules of \( \vdash \beta\Pi \).

2. Induction on the structure of \( B \).

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

4. Idem.

Lemma 3.8 (Generation Lemma for \( \vdash \beta\Pi \))

1. If \( \Gamma \vdash \beta\Pi x : A \) then for some \( d \in \Gamma \), \( x \equiv \text{subj}(d) \), \( \Gamma \vdash \beta\Pi A =\text{def} \ \text{pred}(d) \) and \( \Gamma \vdash \beta\Pi A : S \) for some sort \( S \).

2. If \( \Gamma \vdash \beta\Pi \lambda_{x:A}.B : C \) then for some \( D \) and sort \( S \): \( \Gamma.\lambda_{x:A} \vdash \beta\Pi B : D \), \( \Gamma \vdash \beta\Pi \Pi_{x:A}.D : S \), \( \Gamma \vdash \beta\Pi B : D \) and if \( \Pi_{x:A}.D \not\equiv C \) then \( \Gamma \vdash \beta\Pi C : S' \) for some sort \( S' \).

3. If \( \Gamma \vdash \beta\Pi \Pi_{x:A}.B : C \) then for some sorts \( S_1, S_2 \): \( \Gamma \vdash \beta\Pi A : S_1 \), \( \Gamma \vdash \beta\Pi B : S_2 \), \( (S_1, S_2) \) is a rule, \( \Gamma \vdash \beta\Pi C =\text{def} \ S_2 \) and if \( S_2 \not\equiv C \) then \( \Gamma \vdash \beta\Pi C : S \) for some \( S \).

4. If \( \Gamma \vdash \beta\Pi Fa : C \), \( F \not\equiv \lambda_{x:A}.B \), then for some \( D, E \): \( \Gamma \vdash \beta\Pi a : D \), \( \Gamma \vdash \beta\Pi F : \Pi_{x:D}.E \), \( \Gamma \vdash \beta\Pi C =\text{def} \ C \) and if \( \Pi_{x:D}.E \not\equiv C \) then \( \Gamma \vdash \beta\Pi C : S \) for some \( S \).

5. If \( \Gamma \vdash \beta\Pi (\pi_{x:A}.D)B : C \), then \( \Gamma.(\pi_{x:A}.--)B \vdash \beta\Pi D : C \)

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

- \( \Gamma.(\pi_{x:A}.--)B \vdash \beta\Pi D : C' \), \( \Gamma \vdash \beta\Pi C'[x := B] =\text{def} \ C \) and if \( C'[x := B] \not\equiv C \) then \( \Gamma \vdash \beta\Pi C : S \) for some sort \( S \).

- \( \Gamma \vdash \beta\Pi B : F \), \( \Gamma \vdash \beta\Pi \lambda_{x:A}.D : \Pi_{y:F}.G \), \( \Gamma \vdash \beta\Pi C =\text{def} \ (\Pi_{y:F}.G)B \) and if \( (\Pi_{y:F}.G)B \not\equiv C \) then \( \Gamma \vdash \beta\Pi C : S \) for some sort \( S \).

In the first case we know by thinning that \( \Gamma.(\pi_{x:A}.--)B \vdash \beta\Pi C'[x := B] =\text{def} \ C \) and also \( \Gamma.(\pi_{x:A}.--)B \vdash \beta\Pi C'[x := B] =\text{def} \ C'[x := B] \), hence by conversion \( \Gamma.(\pi_{x:A}.--)B \vdash \beta\Pi D : C \).

In the second case 2. tells us \( \Gamma.\lambda_{x:A} \vdash \beta\Pi D : H \), \( \Gamma \vdash \beta\Pi \Pi_{x:A}.H : S \), \( \Gamma \vdash \beta\Pi \Pi_{x:A}.H =\text{def} \ \Pi_{y:F}.G \) and if \( \Pi_{x:A}.H \not\equiv \Pi_{y:F}.G \) then \( \Gamma \vdash \beta\Pi \Pi_{y:F}.G : S' \) for some sort \( S' \).

This means that \( x \equiv y \), \( \Gamma \vdash \beta\Pi A =\text{def} \ F \) and \( \Gamma \vdash \beta\Pi H =\text{def} \ G \). Out of \( \Gamma \vdash \beta\Pi \Pi_{x:A}.H : S \) we get by 3. that \( \Gamma \vdash \beta\Pi A : S'' \) for some sort \( S'' \), hence by conversion \( \Gamma \vdash \beta\Pi B : A \).

Now by thinning we get \( \Gamma.(\lambda_{x:A}.--)B \vdash \beta\Pi D : H \). As we know \( \Gamma \vdash \beta\Pi C =\text{def} \ (\Pi_{x:F}.G)B \), \( \Gamma \vdash \beta\Pi G =\text{def} \ H \) and \( \Gamma.(\lambda_{x:A}.--)B \vdash \beta\Pi H =\text{def} \ H[x := B] \), we get (use thinning)

\( \Gamma.(\lambda_{x:A}.--)B \vdash \beta\Pi H =\text{def} \ C \) so by conversion \( \Gamma.(\lambda_{x:A}.--)B \vdash \beta\Pi D : C \).
Lemma 3.9 (λΠ-exchanging)

\[ \Gamma, \Delta \vdash \beta_{\Pi} A : C \iff \Gamma, \Pi[A] \Delta \vdash \beta_{\Pi} A : C \]
and \[ \Gamma, (\lambda x : A. -) B, \Delta \vdash \beta_{\Pi} A : C \iff \Gamma, (\Pi[A]) B, \Delta \vdash \beta_{\Pi} A : C \]

**Proof:** induction on the derivation rules. We treat one case of the start rule:
\[ \Gamma, (\lambda x : A. -) B \vdash \beta_{\Pi} A \Rightarrow x : A \text{ as a consequence of } \Gamma \vdash (\lambda x : A. -) B. \]
Then also \( \Gamma \vdash (\Pi[A]) B \), so \( \Gamma, (\Pi[A]) B \vdash \beta_{\Pi} A : A \).

Corollary 3.10 (Correctness of Types)

If \( \Gamma \vdash \beta_{\Pi} A : B \) then \( B = \Box \) or \( \Gamma \vdash \beta_{\Pi} B : S \) for some sort \( S \).

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

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

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

Theorem 3.11 (Subject Reduction for \( \vdash \beta_{\Pi} \) and \( \rightarrow_{\Pi} \))

If \( \Gamma \vdash \beta_{\Pi} A : B \) and \( A \rightarrow_{\Pi} A' \) then \( \Gamma \vdash \beta_{\Pi} A' : B \).

**Proof:** We prove by simultaneous induction on the derivation rules:

1. If \( \Gamma \vdash \beta_{\Pi} A : B \) and \( \Gamma' \) results from contracting one of the terms in the declarations and definitions of \( \Gamma \) by a one step \( \Pi\Pi\)-reduction, then \( \Gamma' \vdash \beta_{\Pi} A : B \)

2. If \( \Gamma \vdash \beta_{\Pi} A : B \) and \( A \rightarrow_{\Pi} A' \) then \( \Gamma \vdash \beta_{\Pi} A' : B \)

• (axiom): nothing to prove

• (start rule): We consider the case \( d = (\lambda x : A. -) B, A \rightarrow_{\Pi} A' \). The other cases are similar or easy.

We have: \( \Gamma, (\lambda x : A. -) B \vdash \beta_{\Pi} x : A \) as a consequence of \( \Gamma \vdash (\lambda x : A. -) B, \) i.e. \( \Gamma \vdash \beta_{\Pi} B : A : S \). By the induction hypothesis \( \Gamma \vdash \beta_{\Pi} A' : S \) and by the induction hypothesis and conversion \( \Gamma \vdash \beta_{\Pi} B : A' \). Hence \( \Gamma, (\lambda x : A. -) B \vdash \beta_{\Pi} x : A' \) and again by conversion \( \Gamma, (\lambda x : A. -) B \vdash \beta_{\Pi} x : A \).

• (weak), (formation), (conversion): use the induction hypothesis.

• (abstraction): use the induction hypothesis and conversion.

• (definition): \( \Gamma \vdash \beta_{\Pi} (\tau x : A. D) B : C[x := B] \) as a consequence of \( \Gamma \vdash \beta_{\Pi} D : C \). Now \( \Gamma' \vdash \beta_{\Pi} (\tau x : A. D) B : C[x := B], \Gamma \vdash \beta_{\Pi} (\tau x : A. D') B : C[x := B] \) and \( \Gamma \vdash \beta_{\Pi} (\tau x : A. D') B : C[x := B] \) by the induction hypothesis.

Furthermore, if \( B \rightarrow_{\Pi} B' \) then \( C[x := B] \equiv_{\Pi} C[x := B'] \) and by the induction hypothesis and definition rule we get \( \Gamma \vdash \beta_{\Pi} (\tau x : A. D) B' : C[x := B'] \). Now by Lemma
Definition 3.12

For all pseudo-expressions\( A \) we define \( \tilde{A} \) to be the term \( A \) where all partnered \( \Pi \)-symbols have been changed into \( \lambda \)-symbols.

The proof of Strong Normalisation is based on Strong Normalisation of the \( \lambda \)-cube extended with definitions as in [BKN 9y]. First we change II-redexes into \( \lambda \)-redexes.

**Lemma 3.13** If \( \Gamma \vdash_{\beta_{\Pi}} A : B \) then \( \tilde{\Gamma} \vdash_{\tau} \tilde{A} : \tilde{B} \), where \( \vdash_{\tau} \) is the typing relation of systems of the \( \lambda \)-cube extended with definitions (no \( \Pi \)-definitions are allowed in the context, the reduction relation is \( \beta \)-reduction only, not \( \beta_{\Pi} \)-reduction, and the abstraction rule has the old format, i.e. \( \Gamma \vdash_{\tau} F : \Pi_{x:A} B \quad \Gamma \vdash_{\tau} a : A \) \( \) see [BKN 9y]).

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

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

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

But \( F \) cannot contain a \( \Pi \)-symbol which will mix with \( a \) in \( Fa \) to form a \( \Pi \)-redex. Otherwise, one can show by the generation lemma that \( \Pi_{x:A} B \equiv_{def} S \) for some \( S \) But this is impossible, hence \( \tilde{F} a \equiv_{def} \tilde{F} \tilde{a} \).

**Theorem 3.14** (Strong Normalisation for the Cube with respect to \( \beta_{\Pi} \) and \( \rightarrow_{\beta} \))

If \( A \) is a \( \beta_{\Pi} \)-legal term then \( A \) is strongly normalising with respect to \( \rightarrow_{\beta} \).

**Proof:** If \( A \) is \( \beta_{\Pi} \)-legal then \( \tilde{A} \) is \( \tau \)-legal by Lemma 3.13 and hence \( \tilde{A} \) is strongly normalising with respect to \( \rightarrow_{\beta} \) (see [BKN 9y]). But then also \( A \) is strongly normalising with respect to \( \rightarrow_{\beta_{\Pi}} \).

\[ \square \]
References


## Computing Science Reports

### Department of Mathematics and Computing Science
Eindhoven University of Technology

**In this series appeared:**

<table>
<thead>
<tr>
<th>Number</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>91/02</td>
<td>R.P. Nederpelt, H.C.M. de Swart</td>
<td>Implication. A survey of the different logical analyses &quot;if...,then...&quot;, p. 26.</td>
</tr>
<tr>
<td>91/03</td>
<td>J.P. Kataen, L.A.M. Schoenmakers</td>
<td>Parallel Programs for the Recognition of P-invariant Segments, p. 16.</td>
</tr>
<tr>
<td>91/05</td>
<td>D. de Reus</td>
<td>An Implementation Model for GOOD, p. 18.</td>
</tr>
<tr>
<td>91/06</td>
<td>K.M. van Hee</td>
<td>SPECIFICATIEMETHODEN, een overzicht, p. 20.</td>
</tr>
<tr>
<td>91/07</td>
<td>E. Poll</td>
<td>CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.</td>
</tr>
<tr>
<td>91/12</td>
<td>E. van der Sluis</td>
<td>A parallel local search algorithm for the travelling salesman problem, p. 12.</td>
</tr>
<tr>
<td>91/14</td>
<td>P. Lemmens</td>
<td>The PDB Hypermedia Package. Why and how it was built, p. 63.</td>
</tr>
<tr>
<td>91/16</td>
<td>A.J.J.M. Marcelis</td>
<td>An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.</td>
</tr>
</tbody>
</table>
91/17  A.T.M. Aerts  
P.M.E. de Bra  
K.M. van Hee  
Transforming Functional Database Schemes to Relational  
Representations, p. 21.

91/18  Rik van Geldrop  
Transformational Query Solving, p. 35.

91/19  Erik Poll  
Some categorical properties for a model for second order  
lambda calculus with subtyping, p. 21.

91/20  A.E. Eiben  
R.V. Schuwer  

91/21  J. Coenen  
W.-P. de Roever  
J. Zwiers  
Assertional Data Reification Proofs: Survey and  
Perspective, p. 18.

91/22  G. Wolf  
Schedule Management: an Object Oriented Approach, p.  
26.

91/23  K.M. van Hee  
L.J. Somers  
M. Voorhoeve  
Z and high level Petri nets, p. 16.

91/24  A.T.M. Aerts  
D. de Reus  
Formal semantics for BRM with examples, p. 25.

91/25  P. Zhou  
J. Hooman  
R. Kuiper  
A compositional proof system for real-time systems based  
on explicit clock temporal logic: soundness and complete  
ness, p. 52.

91/26  P. de Bra  
G.J. Houben  
J. Paredaens  
The GOOD based hypertext reference model, p. 12.

91/27  F. de Boer  
C. Palamidessi  
Embedding as a tool for language comparison: On the  
CSP hierarchy, p. 17.

91/28  F. de Boer  
A compositional proof system for dynamic proces  
creation, p. 24.

91/29  H. Ten Eikelder  
R. van Geldrop  
Correctness of Acceptor Schemes for Regular Languages,  
p. 31.

91/30  J.C.M. Baeten  
F.W. Vaandrager  
An Algebra for Process Creation, p. 29.

91/31  H. ten Eikelder  
Some algorithms to decide the equivalence of recursive  

91/32  P. Struik  
Techniques for designing efficient parallel programs, p.  
14.

91/33  W. v.d. Aalst  
The modelling and analysis of queueing systems with  
QNM-ExSpec, p. 23.

91/34  J. Coenen  
Specifying fault tolerant programs in deontic logic,  
p. 15.
Asynchronous communication in process algebra, p. 20.

A note on compositional refinement, p. 27.

A compositional semantics for fault tolerant real-time systems, p. 18.

Real space process algebra, p. 42.

Program derivation in acyclic graphs and related problems, p. 90.

Conservative fixpoint functions on a graph, p. 25.

Discrete time process algebra, p. 45.

The fine-structure of lambda calculus, p. 110.

On stepwise explicit substitution, p. 30.


Composition and decomposition in a CPN model, p. 55.

Demonic operators and monotype factors, p. 29.


Set theory and nominalisation, Part II, p. 22.

The total order assumption, p. 10.

A system at the cross-roads of functional and logic programming, p. 36.

Integrity checking in deductive databases: an exposition, p. 32.

Interval timed coloured Petri nets and their analysis, p. 20.

A unified approach to Type Theory through a refined lambda-calculus, p. 30.

Axiomatizing Probabilistic Processes: ACP with Generative Probabilities, p. 36.

Are Types for Natural Language? P. 32.
Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.

A useful lambda notation, p. 17.

Nominalization, Predication and Type Containment, p. 40.

Bottom-up Abstract Interpretation of Logic Programs, p. 33.

A Programming Logic for Fo, p. 15.

A modelling method using MOVIE and SimCon/ExSpect, p. 15.

A taxonomy of keyword pattern matching algorithms, p. 50.

Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.

A continuous version of the Prisoner's Dilemma, p. 17

Quicksort for linked lists, p. 8.

Deterministic and randomized local search, p. 78.

A congruence theorem for structured operational semantics with predicates, p. 18.

On the unavoidability of metastable behaviour, p. 29

Exercises in Multiprogramming, p. 97

A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.


Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.


<table>
<thead>
<tr>
<th>Paper Number</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>93/16</td>
<td>H. Schepers, J. Hooman</td>
<td>A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27.</td>
</tr>
<tr>
<td>93/17</td>
<td>D. Alstein, P. van der Stok</td>
<td>Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.</td>
</tr>
<tr>
<td>93/18</td>
<td>C. Verhoef</td>
<td>A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.</td>
</tr>
<tr>
<td>93/19</td>
<td>G-J. Houben</td>
<td>The Design of an Online Help Facility for ExSpect, p. 21.</td>
</tr>
<tr>
<td>93/22</td>
<td>E. Poll</td>
<td>A Typechecker for Bijective Pure Type Systems, p. 28.</td>
</tr>
<tr>
<td>93/23</td>
<td>E. de Kogel</td>
<td>Relational Algebra and Equational Proofs, p. 23.</td>
</tr>
<tr>
<td>93/24</td>
<td>E. Poll and Paula Severi</td>
<td>Pure Type Systems with Definitions, p. 38.</td>
</tr>
<tr>
<td>93/26</td>
<td>W.M.P. van der Aalst</td>
<td>Multi-dimensional Petri nets, p. 25.</td>
</tr>
<tr>
<td>93/27</td>
<td>T. Kloks and D. Kratsch</td>
<td>Finding all minimal separators of a graph, p. 11.</td>
</tr>
<tr>
<td>93/28</td>
<td>F. Kamareddine and R. Nederpelt</td>
<td>A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.</td>
</tr>
<tr>
<td>93/29</td>
<td>R. Post and P. De Bra</td>
<td>GOLD, a Graph Oriented Language for Databases, p. 42.</td>
</tr>
<tr>
<td>93/30</td>
<td>J. Deogun, T. Kloks, D. Kratsch, H. Müller</td>
<td>On Vertex Ranking for Permutation and Other Graphs, p. 11.</td>
</tr>
<tr>
<td>93/31</td>
<td>W. Körver</td>
<td>Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.</td>
</tr>
<tr>
<td>Paper ID</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>93/33</td>
<td>L. Loyens and J. Moonen</td>
<td>ILIAS, a sequential language for parallel matrix computations, p. 20.</td>
</tr>
<tr>
<td>93/34</td>
<td>J.C.M. Baeten and J.A. Bergstra</td>
<td>Real Time Process Algebra with Infinitesimals, p.39.</td>
</tr>
<tr>
<td>93/36</td>
<td>J.C.M. Baeten and J.A. Bergstra</td>
<td>Non Interleaving Process Algebra, p. 17.</td>
</tr>
<tr>
<td>93/38</td>
<td>C. Verhoef</td>
<td>A general conservative extension theorem in process algebra, p. 17.</td>
</tr>
<tr>
<td>93/41</td>
<td>A. Bijlsma</td>
<td>Temporal operators viewed as predicate transformers, p. 11.</td>
</tr>
<tr>
<td>93/42</td>
<td>P.M.P. Rambags</td>
<td>Automatic Verification of Regular Protocols in P/T Nets, p. 23.</td>
</tr>
<tr>
<td>93/43</td>
<td>B.W. Watson</td>
<td>A taxonomy of finite automata construction algorithms, p. 87.</td>
</tr>
<tr>
<td>93/44</td>
<td>B.W. Watson</td>
<td>A taxonomy of finite automata minimization algorithms, p. 23.</td>
</tr>
<tr>
<td>93/48</td>
<td>R. Gerth</td>
<td>Verifying Sequentially Consistently Memory using Interface Refinement, p. 20.</td>
</tr>
</tbody>
</table>
94/01 P. America  
M. van der Kamm  
R.P. Nederpelt  
O.S. van Roosmalen  
H.C.M. de Swart  
The object-oriented paradigm, p. 28.

94/02 F. Kamareddine  
R.P. Nederpelt  
Canonical typing and Π-conversion, p. 51.

94/03 L.B. Hartman  
K.M. van Hee  
Application of Markov Decision Processes to Search  
Problems, p. 21.

94/04 J.C.M. Baeten  
J.A. Bergstra  
Graph Isomorphism Models for Non Interleaving Process  
Algebra, p. 18.

94/05 P. Zhou  
J. Hooman  
Formal Specification and Compositional Verification of  
an Atomic Broadcast Protocol, p. 22.

94/06 T. Basten  
T. Kunz  
J. Black  
M. Coffin  
D. Taylor  
Time and the Order of Abstract Events in Distributed  
Computations, p. 29.

94/07 K.R. Apt  
R. Bol  

94/08 O.S. van Roosmalen  
A Hierarchical Diagrammatic Representation of Class  
Structure, p. 22.

94/09 J.C.M. Baeten  
J.A. Bergstra  
Process Algebra with Partial Choice, p. 16.

94/10 T. Verhoeff  
The testing Paradigm Applied to Network Structure.  
p. 31.

94/11 J. Peleska  
C. Huizing  
C. Petersohn  
A Comparison of Ward & Mellor's Transformation  

94/12 T. Kloks  
D. Kratsch  
H. Müller  

94/13 R. Seljée  
A New Method for Integrity Constraint checking in  
Deductive Databases, p. 34.

94/14 W. Peremans  
Ups and Downs of Type Theory, p. 9.

94/15 R.J.M. Vaessens  
E.H.L. Aarts  
J.K. Lenstra  
Job Shop Scheduling by Local Search, p. 21.

94/16 R.C. Backhouse  
H. Doornbos  
Mathematical Induction Made Calculational, p. 36.

94/17 S. Mauw  
M.A. Reniers  
An Algebraic Semantics of Basic Message  
Sequence Charts, p. 9.
94/18 F. Kamareddine
   R. Nederpelt  Refining Reduction in the Lambda Calculus, p. 15.

94/19 B.W. Watson  The performance of single-keyword and multiple-keyword pattern matching algorithms, p. 46.

94/20 R. Bloo
   F. Kamareddine
   R. Nederpelt  Beyond β-Reduction in Church's λ→, p. 22.


94/22 B.W. Watson  The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and regular Expressions.

94/23 S. Mauw and M.A. Reniers An algebraic semantics of Message Sequence Charts, p. 43.

94/24 D. Dams
   O. Grumberg

94/25 T. Kloks  K_{1,3}-free and W_4-free graphs, p. 10.


94/28 C.W.A.M. van Overveld
   M. Verhoeven  Stars or Stripes: a comparative study of finite and transfinite techniques for surface modelling, p. 20.

94/29 J. Hooman  Correctness of Real Time Systems by Construction, p. 22.

94/30 J.C.M. Baeten
   J.A. Bergstra
   Gh. Ştefănescu Process Algebra with Feedback, p. 22.

94/31 B.W. Watson
   R.E. Watson  A Boyer-Moore type algorithm for regular expression pattern matching, p. 22.


94/33 T. Laan  A formalization of the Ramified Type Theory, p.40.

94/34 R. Bloo
   F. Kamareddine
   R. Nederpelt  The Barendregt Cube with Definitions and Generalised Reduction, p. 37.

94/35 J.C.M. Baeten
   S. Mauw  Delayed choice: an operator for joining Message Sequence Charts, p. 15.

94/36 F. Kamareddine
   R. Nederpelt  Canonical typing and Π-conversion in the Barendregt Cube, p. 19.
Simulating and Analyzing Railway Interlockings in ExSpect, p. 30.

Point-free substitution, p. 10.

On the equivalence covering number of splitgraphs, p. 4.

Distributed Consensus and Hard Real-Time Systems, p. 34.

Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph, p. 6.

Concatenation of Graphs, p. 7.

Category Theory as Coherently Constructive Lattice Theory: An Illustration, p. 35.

Verifying Sequentially Consistent Memory, p. 160

Tutorial voor de ExSpect-bibliotheek voor "Administratieve Logistiek", p. 43.