Canonical typing and pi-conversion in the Barendregt cube

Citation for published version (APA):

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

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 author's 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.

Take down policy
If you believe that this document breaches copyright please contact us:
openaccess@tue.nl
providing details. We will immediately remove access to the work pending the investigation of your claim.
Eindhoven University of Technology
Department of Mathematics and Computing Science

Canonical typing and Π-conversion in
the Barendregt Cube

by

F. Kamareddine and R. Nederpelt

94/36

ISSN 0926-4515

All rights reserved
editors: prof.dr. J.C.M. Baeten
prof.dr. M. Rem

Computing Science Report 94/36
Eindhoven, September 1994
Canonical typing and $\Pi$-conversion in the Barendregt Cube*

Fairouz Kamareddine
Department of Computing Science
17 Lilybank Gardens
University of Glasgow
Glasgow G12 8QQ, Scotland
email: fairouz@cs.gla.ac.uk

and

Rob Nederpelt
Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven, the Netherlands
email: wsinrpn@win.tue.nl

September 3, 1994

---

*First of all, we are very grateful to our colleague Bert van Benthem Jutting who has read draft versions of the manuscript, and who has made very useful suggestions. Furthermore, we are grateful for the discussions with Henk Barendregt, Roel Bloo, Tijn Borghuis, Herman Geuvers, Kevin Hammond, Simon Peyton-Jones, Erik Poll and Phil Wadler, and for the helpful remarks received from them. Last but not least, we are grateful to the anonymous referees for their constructive comments and criticisms.

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 in 1993 and 1994. Furthermore, Kamareddine is grateful to the Department of Mathematics and Computer Science, University of Amsterdam, and in particular to Jan Bergstra and Inge Bethke for their hospitality during the preparation of this article, and to the Dutch organisation of research (NWO) for its financial support. Last but not least, Kamareddine is grateful to the ESPRIT Basic Action for Research project “Types for Proofs and Programming” for its financial support.
Abstract

In this article, we extend the Barendregt Cube with Π-conversion (which is the analogue of β-conversion, on product type level) and study its properties. We use this extension to separate the problem of whether a term is typable from the problem of what is the type of a term.

Keywords: Barendregt Cube, Π-conversion, Canonical Typing, Typable Terms, Subject Reduction, Church Rosser, Strong Normalisation.

Contents

1 Introduction 3
2 The formal machinery of the Cube 6
3 The ordinary typing relation ⊢_β and its properties 7
4 The extended typing relation ⊢_Π and its properties 10
5 The canonical typing operator τ and its properties 13
6 The typability relation ⊢ and its properties 15
1 Introduction

At the end of the nineteenth century, types did not play a role in mathematics or logic, unless at the meta-level, in order to distinguish between different ‘classes’ of objects. Frege’s formalization of logical reasoning, as explained in the Begriffsschrift ([Frege 1879]), was untyped. Only after the discovery of Russell’s paradox, undermining Frege’s work, one may observe various formulations of typed theories. Types could explain away the paradoxical instances. The first theory which aimed at doing so, was that of Russell and Whitehead, as exposed in their famous Principia Mathematica ([Whitehead and Russell 1910]). Their ‘ramified theory of types’ has later been adapted and simplified by Hilbert and Ackermann ([Hilbert and Ackermann 1928]).

Church was the first to define a type theory ‘as such’, almost a decade after he developed a theory of functionals which is nowadays called λ-calculus ([Church 1932]). This calculus was used for defining a notion of computability that turned out to be of the same power as Turing-computability or general recursiveness. However, the original, untyped version did not work as a foundation for mathematics. In order to come round the inconsistencies in his proposal for logic, Church developed the ‘simple theory of types’ ([Church 1940]).

From then till the present day, research on the area has grown and one can find various reformulations of type theories. A taxonomy of type systems has recently been given by Barendregt ([Barendregt 92]). A version of Church’s simple theory of types is found in this taxonomy under the name $\lambda$. This $\lambda$ has, apart from type variables, so-called arrow-types of the form $A \rightarrow B$. In higher type theories, arrow-types 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. We propose reduction rules which treat alike types and terms. That is, not only we have $(\lambda_{x:A}.b)C \rightarrow_{\beta} b[x := C]$, but also $(\Pi_{x:A}.B)C \rightarrow_{\beta} B[x := C]$.

This strategy of permitting $\Pi$-application $(\Pi_{x:A}.B)C$ in term construction is not commonly used, yet is desirable for the following reasons:

1. **$\Pi$-reduction behaves like $\beta$-reduction.** One may say that $\beta$-reduction has been invented as an expedient in order to forebode a possible substitution. So why does one use a direct substitution as in equation 1 below, (which is used almost everywhere) if $\beta$-reduction can be used to do the job, as shown in equation 2? (We omit the contexts, for the sake of simplicity):

   If $f : \Pi_{x:A}.B$ and $a : A$, then $fa : B[x := a]$  

   (1)

   If $f : \Pi_{x:A}.B$ and $a : A$, then $fa : (\Pi_{x:A}.B)a$ (which $\beta$-reduces to $fa : B[x := a]$).  

   (2)

   In fact, it is more elegant and uniform to use the second notation instead of the first one.

2. **Compatibility.** With $\Pi$-reduction, one introduces a compatibility property for the typing of applications:

   $M : N \Rightarrow MP : NP$.  

   This is in line with the compatibility property for the typing of abstractions, which does hold in general:

   $M : N \Rightarrow \lambda_{y:P}M : \Pi_{y:P}N$. 


As an example, we give a simple derivation with the above-described compatible application rule and with conversion on II-application:

\[
\begin{align*}
A : *, b : A, a : A & \vdash a : A \quad \text{(start)} \\
A : *, b : A & \vdash (\lambda_{a:A} a) : (\Pi_{a:A} A) \quad \text{(abstraction)} \\
A : *, b : A & \vdash (\lambda_{a:A} a)b : (\Pi_{a:A} A)b \quad \text{(application)} \\
A : *, b : A & \vdash (\lambda_{a:A} a)b : A \quad \text{(conversion)}
\end{align*}
\]

3. Unified treatment of terms and types. It is our belief that with II-reduction it is simpler to treat terms and types in a unified manner. Such a treatment provides a step towards the generalisation of type systems which is an important topic of research at the present time. For example, Barendregt's taxonomy of type systems in [Barendregt 92], but also Pure Type Systems (PTSs) introduced by Terlouw and Berardi (see [Ter 89]), and our generalised system in [NK 94] are attempts at combining all the important results of type systems in a compact and elegant way. As a step towards this goal, we believe that conversion should apply to both types and terms. In fact, II is indeed a kind of A, hence eligible for an application. This is a quite natural approach and one may interpret \((\Pi x,A.B)a\) as the wish to select the “axis” \(B(a)\) in the Cartesian product \(\Pi x,A.B\). One might argue that implicit II-reduction (as is the case of the ordinary Cube) is closer to the intuition in the most usual applications. However, experiences with the Automath-languages ([de Bruijn 74]), containing explicit II-reduction, demonstrated that there exists no formal or informal objection against the use of this explicit II-reduction in natural applications of type systems.

4. The ability to divide two important questions of typing. Introducing explicit II-reduction gives an elegant way to divide two important questions which are usually answered together via the judgement \(\Gamma \vdash A : B\). These questions are:

1. Is \(A\) typable in \(\Gamma\)? (Below we use the simplified judgement \(\Gamma \vdash A\) for this question.)

2. Is \(B\) the type of \(A\) in \(\Gamma\)? (Below we use a canonical type \(\tau(\Gamma, A)\) for \(A\) and compare this canonical type with \(B\), for this question.)

II-reduction is needed in order to split elegantly these two questions. In particular, we require for an application \(\tau(\Gamma, F)a \equiv \tau(\Gamma, F)a\) on the condition that \(\tau(\Gamma, F) = \Pi x,A.B\), hence we obtain \((\Pi x,A.B)a\), a II-redex.

There are reasons why separating the questions “what is the type of a term” (via \(\tau\)) and “is the term typable” (via \(\vdash\)), is advantageous. Here are some:

1. The canonical type of \(A\) is easy to calculate. The canonical type of \(A\), \(\tau(\Gamma, A)\) is defined by just scanning through \(A\), removing all so called main II-items \(\Pi x,B\), replacing all main \(\lambda\)-items \(\lambda x:B\) by \(\Pi x:B\) and replacing the heart of \(A\) by its obvious type in \(A\). For example: if \(A \equiv \Pi z.(\lambda y.(\lambda z:x^2) y)(\Pi w.(\lambda z:x) y)\), then \(\Pi z\) is the main II-item of \(A\), \(\lambda y\) and \(\lambda z\) are the main \(\lambda\)-items and \(x^2\) is the heart of \(A\). Hence, \(\tau(\Gamma, A) \equiv (\Pi y.(\Pi z:*) y)(\Pi w.(\lambda z:x) y)\).

A consequence is that the mapping algorithm (in order to find a type for a term) is extremely simple. This contrasts with the mapping algorithm in the usual setting, which needs intermediate applications of the conversion rule. This is caused by the fact that \(Fa\) is only typable if \(F\) has an appropriate II-type. If \(F\) has not (yet) a II-type, then the conversion rule must be used to find one. Of course we will need a conversion rule in order to check whether \(A\) has type \(B\) in context \(\Gamma\) (by establishing that \(\tau(\Gamma, A) = B\)). Note, however, that we use only typing for the calculation of the canonical type, and only conversion for the second part (“\(\tau(\Gamma, A) = B?\)”). This is clearly a separation of concerns.
2. $\tau(A)$ plays the role of a preference type for $A$. To define the type of a term, in the traditional Cube, one starts with the types of variables, and subsequently deduces other statements of the form $\Gamma \vdash A : B$, by regarding more complex terms and their types. Finally, a conversion rule expresses that the types of terms are given modulo conversion; i.e., if $A : B$ and $B =_{\beta} C$, then $A : C$. The typing relation is the smallest relation satisfying these rules.

In our opinion, the approach in the traditional frameworks is, in a sense, ambiguous. Note that with each variable $x$ and pseudo-context $\Gamma$, there is associated a preference type, which is $B$ for $x : B \in \Gamma$. For terms in general no preference type has been given, but a whole collection of types, which are typeable by themselves and linked by means of $\beta$-reduction.

We define however, the canonical type of $A$, $\tau(A)$, which plays the role of a preference type. For example, the preference type of $A = \lambda^x\cdot(\lambda^y\cdot y)x$ is $\tau(\lambda^x\cdot(\lambda^y\cdot y))x$. This type indeed reduces with the relation $\rightarrow_{\beta II}$ to $\Pi x.\top$, the type traditionally given to $A$.

3. The conversion rule is no longer needed as a separate rule in the definition of $\vdash$. In our approach, $\beta$-conversion finds its place in the application condition of the rules of $\vdash$, where it naturally belongs. The conversion rule of the cube is redundant in our system. It is accommodated in our application rule:

$$
\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash AB} \quad \text{if } \tau(\Gamma, A) =_{\beta II} \Pi x:C.D \text{ and } \tau(\Gamma, B) =_{\beta II} C
$$

It will be the case that $\tau(\Gamma, AB) \equiv \tau(\Gamma, A)B =_{\beta II} (\Pi x:C.D)B \rightarrow_{\beta II} D[x := B]$ and so indeed $\tau(\Gamma, AB) =_{\beta II} D[x := C]$.

4. Higher degrees If we use $\lambda^1$ for II and $\lambda^2$ for $\lambda$ then we can aim for a possible generalization. In fact, we can extend our system by incorporating more different $\lambda$'s. For example, with an infinity of $\lambda$'s, viz. $\lambda^0, \lambda^1, \lambda^2, \lambda^3 \ldots$, we replace $\tau(\Gamma, \lambda x:A.B) \equiv \Pi x:A.\tau(\Gamma, x:A, B)$ and $\tau(\Gamma, \Pi x:A.B) \equiv \tau(\Gamma, x:A, B)$ by the following:

$$
\tau(\Gamma, \lambda^{i+1} x:A.B) \equiv \lambda^{i+1} x:A.\tau(\Gamma, x:A, B), \text{for } i = 0, 1, 2, \ldots \text{ where } \lambda^{i+1} x:A.B \equiv B
$$

There is no reason why one cannot use as many $\lambda^i$ as possible in a type system. In fact, even though in the Cube there are only two, there are other systems with more. There may be circumstances in which one desires to have more “layers” of $\lambda$’s. As an example we refer to [de Bruijn 74].

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

- In Section 2, we introduce the formal machinery.
- In Section 3, we introduce the usual properties of the Cube for $\vdash_{\beta}$ and $\rightarrow_{\beta}$ which will be studied for our extensions.
- In Section 4, we study in detail the properties of the Barendregt Cube extended with II-conversion and show that $\vdash_{\beta II}$ satisfies all the essential properties of $\vdash_{\beta}$ except for Subject Reduction. That is: $\Gamma \vdash_{\beta II} A : B \land A \rightarrow_{\beta II} A' \not= \Gamma \vdash_{\beta II} A' : B$. Subject Reduction however holds for the case $B \equiv \square$ or $\Gamma \vdash_{\beta II} B : S$. This Weak Subject Reduction is sufficient to obtain the desirable typing properties such as unicity of typing. The explanation for this is that, this $B$ which is not $\square$ or of type $S$, reduces via $\rightarrow_{\beta II}$ to $B'$ which is itself either $\square$ or of type $S$, and hence $\Gamma \vdash_{\beta II} A : B$ implies $\Gamma \vdash_{\beta} B'$ where $B \rightarrow_{\beta II} B'$ and $B'$ has no II-redexes.
- In Sections 5 and 6 we study the properties of the two separate typing questions regarding $\tau$ and $\vdash$.
2 The formal machinery of the Cube

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 (let $\pi$ range over both $\Pi$ and $\lambda$):

$$T = \ast | \Box | V | TT | \pi_{V;T}.T$$

where $V$ is an infinite collection of variables over which $x, y, z, \ldots$ range. $\ast$ and $\Box$ are called sorts over which $S, S_1, S_2, \ldots$ are used to range. We take $A, B, C, a, b, \ldots$ 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. An example is $\beta$-reduction (see Section 3). We say that a reduction relation $\rightarrow$ on terms is compatible iff the following holds:

$$\frac{A_1 \rightarrow A_2}{A_1B \rightarrow A_2B} \quad \frac{B_1 \rightarrow B_2}{AB_1 \rightarrow AB_2} \quad \frac{A_1 \rightarrow A_2}{\pi_{x:A_1}.B \rightarrow \pi_{x:A_2}.B} \quad \frac{B_1 \rightarrow B_2}{\pi_{x:A}.B_1 \rightarrow \pi_{x:A}.B_2}$$

A statement is of the form $A : B$ with $A, B \in T$. $A$ is the subject and $B$ is the predicate of $A : B$. A declaration is of the form $\lambda_{x:A}$ with $A \in T$ and $x \in V$. A pseudo-context is a finite ordered sequence of declarations, all with distinct subjects. The empty context is denoted by $<>$. If $\Gamma = \lambda_{x_1:A};_1, \ldots, \lambda_{x_n:A_n}$ then $\Gamma, \lambda_{x:A} = \lambda_{x_1:A_1}, \ldots, \lambda_{x_n:A_n}.\lambda_{x:A}$ and $\text{dom}(\Gamma) = \{x_1, \ldots, x_n\}$. We use $\Gamma, \Delta, \Gamma', \Gamma_1, \Gamma_2, \ldots$ to range over pseudo-contexts.

A typability relation $\vdash$ is a relation between pseudo-contexts and pseudo-expressions written as $\Gamma \vdash A$. The rules of typability establish which judgements $\Gamma \vdash A$ can be derived. A judgement $\Gamma \vdash A$ states that $A$ is typable in the pseudo-context $\Gamma$.

A type assignment relation $\vdash$ is a relation between a pseudo-context and two pseudo-expressions written as $\Gamma \vdash A : B$. The rules of type assignment establish which judgements $\Gamma \vdash A : B$ can be derived. A judgement $\Gamma \vdash A : B$ states that $A : B$ can be derived from the pseudo-context $\Gamma$.

When $\Gamma \vdash A$ or $\Gamma \vdash A : B$ then $A$ and $B$ are called (legal) expressions and $\Gamma$ is a (legal) context.

We write $\Gamma \vdash A : B : C$ for $\Gamma \vdash A : B \land \Gamma \vdash B : C$. If $\Delta \equiv \lambda_{x_1:A_1}, \ldots, \lambda_{x_n:A_n}$ with $n \geq 0$ is a pseudo-context, then $\Gamma \vdash \Delta$, for $\Gamma$ a type assignment, means $\Gamma \vdash x_i : A_i$ for $1 \leq i \leq n$. If $A \rightarrow B$ then we also say $\Gamma_1, \lambda_{x:A}.\Gamma_2 \rightarrow \Gamma_1, \lambda_{x:B}.\Gamma_2$ and define $\rightarrow$ on pseudo-contexts to be the reflexive transitive closure of $\rightarrow$. 
Remark 2.2 Note that we differ from [Barendregt 92] in that we take a declaration to be $\lambda x:A$ rather than $x:A$. The reason for this is that we want pseudo-contexts to be as close as possible to terms. In fact the context $\Gamma$ can be mapped to the term $\Gamma.\star$ for example, and definitions of boundness/freeness of variables in a term and the Barendregt convention are thus easily extended to pseudo-contexts.

Definition 2.3 (Type of Bound Variables, $\triangleright$)

- If $x$ occurs free in $B$, then all its occurrences are bound with type $A$ in $\pi_{x:A}.B$.
- If an occurrence of $x$ is bound with type $A$ in $B$, then it is also bound with type $A$ in $\pi_{y:C}.B$ for $y \neq x$, in $BC$, and in $CB$.
- Define $\triangleright(x) = x$, $\triangleright(\pi_{x:A}.B) = \triangleright(B)$ and $\triangleright(AB) = \triangleright(A)$.

In this paper (Section 6) we introduce a system where the type information $B$ of a judgement $\Gamma \vdash A : B$ is no longer needed. Hence, judgements obtain the form $\Gamma \vdash A$ (a simple judgement). In the following definition, we include these simple judgements.

Definition 2.4 Let $\Gamma$ be a pseudo-context, $A$ be a pseudo-expression and $\vdash$ be a typability or a type assignment relation.

1. $\Gamma$ is called legal if $\exists P, Q \in T$ such that $\Gamma \vdash P(: Q)$.
2. $A \in T$ is called a $\Gamma$-term if $\Gamma \vdash A(\exists B \in T[\Gamma \vdash A : B \lor \Gamma \vdash B : A])$.
   We take $\Gamma$-terms $= \{A \in T \mid \Gamma \vdash A(\exists B \in T[\Gamma \vdash A : B \lor \Gamma \vdash B : A])\}$.
3. $A \in T$ is called legal if $\exists \Gamma[A \in \Gamma$-terms$]$.
4. We say that $A$ is strongly normalising with respect to a reduction relation $\rightarrow$ (written $\text{SN}_{\rightarrow}(A)$) iff every $\rightarrow$-reduction path starting at $A$ terminates.

Definition 2.5 Let $\Gamma \equiv \lambda_{x_1:A_1} \ldots \lambda_{x_n:A_n}$ and $\Delta \equiv \lambda_{y_1:B_1} \ldots \lambda_{y_m:B_m}$ be pseudo-contexts.

1. We write $\lambda x:A \in \Gamma$ if $x \equiv x_i$ and $A \equiv A_i$ for some $i$.
2. $\Gamma$ is part of $\Delta$, notation $\Gamma \subseteq \Delta$, if every $\lambda x:A$ in $\Gamma$ is also in $\Delta$.
3. Let $X$ be a set of variables. Then $\Gamma \vdash X$ is $\Gamma$ where $\lambda x_i:A_i$ is removed for every $x_i \notin X$.

3 The ordinary typing relation $\vdash_\beta$ and its properties

Definition 3.1 ($\beta$-reduction $\rightarrow_\beta$ for the Cube)

$\beta$-reduction $\rightarrow_\beta$, is the least compatible relation generated out of the following axiom:

$$(\beta) \quad (\lambda x:B.A)C \rightarrow_\beta A[x := C]$$

We take $\rightarrow_\beta$ to be the reflexive transitive closure of $\rightarrow_\beta$ and we take $=_\beta$ to be the least equivalence relation generated by $\rightarrow_\beta$. 
Definition 3.2 \( \vdash _\beta \) The type assignment relation \( \vdash _\beta \) is defined by the following inference rules:

(axiom) \( \vdash _\beta \ast : \Box \)

(start rule) \[
\Gamma \vdash _\beta A : S \\
\Gamma \vdash _\beta x : A \quad x \notin \Gamma
\]

(weakening rule) \[
\Gamma \vdash _\beta A : S \\
\Gamma \vdash _\beta D : E \quad x \notin \Gamma
\]

(application rule) \[
\Gamma \vdash _\beta F : \Pi_{x:A} B \\
\Gamma \vdash _\beta a : A
\]

(abstraction rule) \[
\Gamma \vdash _\beta \lambda_{x:A} b : B \\
\Gamma \vdash _\beta \Pi_{x:A} B : S
\]

(conversion rule) \[
\Gamma \vdash _\beta A : B \\
\Gamma \vdash _\beta A' : B' \quad B =_\beta B'
\]

(formation rule) \[
\Gamma \vdash _\beta A : S_1 \\
\Gamma \vdash _\beta \Pi_{x:A} B : S_2
\]

Each of the eight systems of the Cube is obtained by taking its set of \((S_1, S_2)\) rules allowed in the formation rule out of \{\((\ast, \ast), (\ast, \Box), (\Box, \ast), (\Box, \Box)\)\}. 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 \((\ast, \Box), (\Box, \ast)\) and \((\Box, \Box)\) for \((S_1, S_2)\).

Here is the table which presents the eight systems of the Cube (see also Figure 1):

<table>
<thead>
<tr>
<th>System</th>
<th>Allowed ((S_1, S_2)) rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\lambda)</td>
<td>((\ast, \ast))</td>
</tr>
<tr>
<td>(\lambda 2)</td>
<td>((\ast, \ast))</td>
</tr>
<tr>
<td>(\lambda P)</td>
<td>((\ast, \ast))</td>
</tr>
<tr>
<td>(\lambda P2)</td>
<td>((\ast, \ast))</td>
</tr>
<tr>
<td>(\lambda \omega)</td>
<td>((\ast, \ast))</td>
</tr>
<tr>
<td>(\lambda \omega)</td>
<td>((\ast, \ast))</td>
</tr>
<tr>
<td>(\lambda P \omega)</td>
<td>((\ast, \ast))</td>
</tr>
<tr>
<td>(\lambda P \omega = \lambda \omega)</td>
<td>((\ast, \ast))</td>
</tr>
</tbody>
</table>

Now, we list the properties of the Cube without proofs (see [Barendregt 92]). These properties will be studied in Section 4 for the Cube extended with \(\Pi\)-conversion and will be discussed for the two different subjects of canonical typing and typability in Sections 5 and 6 respectively.

Theorem 3.3 (The Church Rosser Theorem CR, for \(\rightarrow_\beta\))
If \(A \rightarrow_\beta B\) and \(A \rightarrow_\beta C\) then there exists \(D\) such that \(B \rightarrow_\beta D\) and \(C \rightarrow_\beta D\)

Lemma 3.4 (Free variable lemma for \(\vdash _\beta\))
Let \(\Gamma \equiv \lambda_{x_1:A_1} \ldots \lambda_{x_n:A_n}\) be a \(\vdash _\beta\)-legal context such that \(\Gamma \vdash _\beta B : C\). Then we have:

1. The \(x_1 \ldots x_n\) are all distinct.
2. \( FV(B), FV(C) \subseteq \{x_1, \ldots, x_n\} \).

3. \( FV(A_i) \subseteq \{x_1, \ldots, x_{i-1}\} \) for \( 1 \leq i \leq n \).

**Lemma 3.5** (Start Lemma for \( \vdash_\beta \))

Let \( \Gamma \) be a \( \vdash_\beta \)-legal context. Then \( \Gamma \vdash_\beta * : \square \) and \( \forall \lambda x : C \in \Gamma[\Gamma \vdash_\beta x : C] \).

**Lemma 3.6** (Transitivity Lemma for \( \vdash_\beta \))

Let \( \Gamma \) and \( \Delta \) be \( \vdash_\beta \)-legal contexts. Then: \( \Gamma \vdash_\beta \Delta \wedge \Delta \vdash_\beta A : B \Rightarrow \Gamma \vdash_\beta A : B \).

**Lemma 3.7** (Substitution Lemma for \( \vdash_\beta \))

Assume \( \Gamma, A, x, A, B \vdash_\beta C =_\beta S_1 \neg \neg \wedge \neg \neg C \neq S_2 \Rightarrow \exists S[\Gamma \vdash_\beta C : S] \).

**Lemma 3.8** (Thinning Lemma for \( \vdash_\beta \))

Let \( \Gamma \) and \( \Delta \) be \( \vdash_\beta \)-legal contexts such that \( \Gamma \subseteq \Delta \). Then \( \Gamma \vdash_\beta A : B \Rightarrow \Delta \vdash_\beta A : B \).

**Lemma 3.9** (Generation Lemma for \( \vdash_\beta \))

1. \( \Gamma \vdash_\beta S : C \Rightarrow S \equiv *, C =_\beta \square \), and if \( C \neq \square \) then \( \Gamma \vdash_\beta C : S' \) for some sort \( S' \).

2. \( \Gamma \vdash_\beta x : C \Rightarrow \exists B =_\beta C[\lambda x : B \in \Gamma \wedge \Gamma \vdash_\beta C : S \text{ for some sort } S] \).

3. \( \Gamma \vdash_\beta \Pi_{x : A}.B : C \Rightarrow \exists (S_1, S_2)[\Gamma \vdash_\beta A : S_1 \land \Gamma \vdash_\beta B : S_2 \land (S_1, S_2) \text{ is a rule } \land C =_\beta S_2 \land \Gamma \vdash_\beta C : S]] \).

4. \( \Gamma \vdash_\beta \lambda x : A.b : C \Rightarrow \exists (S, B)[\Gamma \vdash_\beta \Pi_{x : A}.B : S \land \Gamma \vdash_\beta b : B \land C =_\beta \Pi_{x : A}.B[\exists S[\Gamma \vdash_\beta C : S]] \).

5. \( \Gamma \vdash_\beta F a : C \Rightarrow \exists A, B, x[\Gamma \vdash_\beta F : \Pi_{x : A}.B \land \Gamma \vdash_\beta a : A \land C =_\beta B[x := a] \land (B[x := a] \neq C \Rightarrow \exists S[\Gamma \vdash_\beta C : S]) \).

**Corollary 3.10** (Correctness of types for \( \vdash_\beta \))

If \( \Gamma \vdash_\beta A : B \) then \( (B \equiv \square \text{ or } \Gamma \vdash_\beta B : S \text{ for some sort } S) \).
Lemma 3.11 (Legal terms and contexts for $\vdash_\beta$ and $\rightarrow_\beta$) $\vdash_\beta$-legal terms and contexts contain no $\Pi$-redexes.

Theorem 3.12 (Subject Reduction SR, for $\vdash_\beta$ and $\rightarrow_\beta$) $\Gamma \vdash_\beta A : B \land A \rightarrow_\beta A' \Rightarrow \Gamma \vdash_\beta A' : B$

Corollary 3.13 (SR Corollary for $\vdash_\beta$ and $\rightarrow_\beta$) 1. If $\Gamma \vdash_\beta A : B$ and $B \rightarrow_\beta B'$ then $\Gamma \vdash_\beta A : B'$.
2. If $A$ is a $\Gamma^\beta$-term and $A \rightarrow_\beta A'$ then $A'$ is a $\Gamma^\beta$-term.

Lemma 3.14 (Unicity of Types for $\vdash_\beta$ and $\rightarrow_\beta$) 1. $\Gamma \vdash_\beta A : B_1 \land \Gamma \vdash_\beta A : B_2 \Rightarrow B_1 =_\beta B_2$
2. $\Gamma \vdash_\beta A : B \land \Gamma \vdash_\beta A' : B' \land A =_\beta A' \Rightarrow B =_\beta B'$
3. $\Gamma \vdash_\beta B : S, B =_\beta B', \Gamma \vdash_\beta A' : B' \Rightarrow \Gamma \vdash_\beta B' : S$.

Theorem 3.15 (Strong Normalisation with respect to $\vdash_\beta$ and $\rightarrow_\beta$) For all $\vdash_\beta$-legal terms $M$, $SN_{\rightarrow_\beta}(M)$; i.e. $M$ is strongly normalising with respect to $\rightarrow_\beta$.

4 The extended typing relation $\vdash_{\beta\Pi}$ and its properties

Definition 4.1 ($\beta\Pi$-reduction $\rightarrow_{\beta\Pi}$ for the Cube) $\beta\Pi$-reduction $\rightarrow_{\beta\Pi}$, is the least compatible relation generated out of the following axiom:

$\quad (\beta\Pi) \quad (\pi_{x:A}.B)C \rightarrow_{\beta\Pi} A[x := C]$

We take $\rightarrow_{\beta\Pi}$ to be the reflexive transitive closure of $\rightarrow_{\beta\Pi}$ and we take $=_{\beta\Pi}$ to be the least equivalence relation generated by $\rightarrow_{\beta\Pi}$.

Definition 4.2 ($\vdash_{\beta\Pi}$) We define $\vdash_{\beta\Pi}$ as $\vdash_\beta$ of Section 3 with the difference that the application and conversion rules change as follows:

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

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

The following lemmas hold for $\vdash_{\beta\Pi}$ and $\rightarrow_{\beta\Pi}$ and have the same formulation (only change $\beta$ to $\beta\Pi$ everywhere) and proofs as for the case of $\vdash_\beta$ and $\rightarrow_\beta$:

- The Church Rosser Theorem for $\rightarrow_{\beta\Pi}$
- Free variable lemma for $\vdash_{\beta\Pi}$
- Start lemma for $\vdash_{\beta\Pi}$
- Transitivity lemma for $\vdash_{\beta\Pi}$
• Thinning lemma for $\vdash_{\beta\Pi}$
• Substitution lemma for $\vdash_{\beta\Pi}$
• Generation lemma for $\vdash_{\beta\Pi}$

where in clause 5, we replace $B[x := a]$ by $(\Pi_{x:A}.B)a$

Remark 4.3 (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 Corollary 3.10 for $\vdash_{\beta\Pi}$. That is, even in $\lambda_\rightarrow$, $\Gamma \vdash_{\beta\Pi} A : B \neq (B \equiv \Box$ or $\Gamma \vdash_{\beta\Pi} B : S$ for some sort $S$). For example, if $\Gamma \equiv \lambda_{x:.} \lambda_{x:x}$ then $\Gamma \vdash_{\beta\Pi} (\lambda_{y:.} \cdot y)x : (\Pi_{y:x}.z)x$, but $\Gamma \not\vdash_{\beta\Pi} (\Pi_{y:x}.z)x : S$ from Lemma 4.5.

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

Example 4.4 In $\lambda_\rightarrow$, $\lambda_{x:.} \lambda_{x:x} \not\vdash_{\beta\Pi} x : (\Pi_{y:x}.z)x$. Otherwise, by generation: $\lambda_{x:.} \lambda_{x:x} \vdash_{\beta\Pi} (\Pi_{y:x}.z)x : S$, which is absurd by Lemma 4.5. Yet in $\lambda_\rightarrow$, $\lambda_{x:.} \lambda_{x:x} \vdash_{\beta\Pi} (\lambda_{y:.} \cdot y)x : (\Pi_{y:x}.z)x$.

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 4.5 For any $A, B, C, S$, $\Gamma \not\vdash_{\beta\Pi} (\Pi_{x:A}.B)C : S$.

Proof: If $\Gamma \not\vdash_{\beta\Pi} (\Pi_{x:A}.B)C : S$ then by generation, $\Gamma \not\vdash_{\beta\Pi} \Pi_{x:A}.B : \Pi_{x:A'}.B'$ and again by generation, $\Gamma \lambda_{x:A} \vdash_{\beta\Pi} B : S' \land S' =_{\beta\Pi} \Pi_{x:A'}.B'$ which is absurd. \hfill \Box

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

Lemma 4.6 $\Gamma \vdash_{\beta\Pi} A : B \land B$ is not a $\Pi$-redex $\Rightarrow (B \equiv \Box$ or $\Gamma \vdash_{\beta\Pi} B : S$ for some sort $S$).

Proof: By a trivial induction on the derivation of $\Gamma \vdash_{\beta\Pi} A : B$ noting that the application rule does not apply as $(\Pi_{x:A}.B)a$ is not a $\Pi$-redex. \hfill \Box

Lemma 4.7 (Legal terms and contexts for $\vdash_{\beta\Pi}$ and $\rightarrow_{\beta\Pi}$)

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 $A \equiv (\Pi_{x:.}D)E$ is $\vdash_{\beta\Pi}$-legal, then $E[x := B]$ contains no $\Pi$-redexes.

Proof: 1. is by induction on the derivation of $\Gamma \vdash_{\beta\Pi} A : B$. 2. By 1, we only need to show that if $B \equiv \Pi_{y:G}.H$, then $E$ does not contain a subterm $xF$. Now, suppose $B \equiv \Pi_{y:G}.H$ and $E \equiv C[x:G]$, then it is easy to see that $D \equiv \Pi_{x:I}.J$ for some $I, J$, and $\Gamma \vdash_{\beta\Pi} B : D$ for some context $\Gamma$. But $\Gamma \not\vdash_{\beta\Pi} \Pi_{y:G}.H : \Pi_{x:I}.J$ is impossible. \hfill \Box

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 4.8 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 4.9

1. If $\Gamma \vdash_{\beta\Pi} A : B$ then $\Gamma \vdash_{\beta} A : \hat{B}$.

2. If $\Gamma \vdash_{\beta} A : B$ then $\Gamma \vdash_{\beta\Pi} A : B$.
Proof: 1. By a trivial induction on the derivations $\Gamma \vdash_{\beta} A : B$. By induction on the derivation $\Gamma \vdash_{\beta} A : B$. The only interesting cases come from conversion and application. The conversion case is easy as if $B =_{\beta} B'$ then $B =_{\beta_{\Pi}} B'$. The application case is shown as follows: If $\Gamma \vdash_{\beta} F : B[x := a]$ comes from $\Gamma \vdash_{\beta} F : \Pi_{x:A}.B$ and $\Gamma \vdash_{\beta} a : A$, then by $\Pi_{\Pi}$, $\Gamma \vdash_{\beta_{\Pi}} F : \Pi_{x:A}.B$ and $\Gamma \vdash_{\beta_{\Pi}} a : A$. Hence, by application, $\Gamma \vdash_{\beta_{\Pi}} F[a := (\Pi_{x:A}.B)a]$. But $(\Pi_{x:A}.B)a =_{\beta_{\Pi}} B[x := a]$. If $\Gamma \vdash_{\beta_{\Pi}} B[x := a] : S$ for some $S$, then by conversion $\Gamma \vdash_{\beta_{\Pi}} F[a := a] : S$ is shown as follows: $\Gamma \vdash_{\beta_{\Pi}} F : \Pi_{x:A}.B$ then $\Pi_{x:A}.B$ is $\beta_{\Pi}$-legal and $\Gamma \vdash_{\beta_{\Pi}} \Pi_{x:A}.B : S'$ for some $S'$ by Lemma 4.6. Now, by the generation lemma, $\Gamma, \lambda_{x:A} \vdash_{\Pi_{\Pi}} B : S$ for some $S$. But $\Gamma \vdash_{\beta_{\Pi}} a : A$. Hence by the substitution lemma: $\Gamma \vdash_{\beta_{\Pi}} B[x := a] : S$. \hfill $\Box$

Remark 4.10 Note that we may have $\Gamma \vdash_{\beta} A : \hat{B}$ without having $\Gamma \vdash_{\beta_{\Pi}} A : B$, even if $B$ is $\beta_{\Pi}$-legal. Take for example $\Gamma \equiv \lambda_{x:.} \lambda_{y:.}, A \equiv x$ and $B \equiv (\Pi_{y:.})x$. We have $\Gamma \vdash_{\beta_{\Pi}} (\lambda_{y:.} y)x : B$ hence $B$ is $\beta_{\Pi}$-legal. We also have $\Gamma \vdash_{\beta} x : \hat{B}$. Yet $\Gamma \vdash_{\beta_{\Pi}} x : B$.

Lemma 4.11 If $\Gamma \vdash_{\beta_{\Pi}} A : B$ and $A \rightarrow_{\beta_{\Pi}} A'$ then $A'$ has no $\Pi$-redexes.

Proof: We only show this for $A \rightarrow_{\beta_{\Pi}} A'$. Note that $A$ has no $\Pi$-redexes and so $A \rightarrow_{\beta} A'$. Now from $\Gamma \vdash_{\beta_{\Pi}} A : B$, we get by Lemma 4.9, 1, $\Gamma \vdash_{\beta} A : \hat{B}$ and so by Subject Reduction for $\rightarrow_{\beta}$ we get $\Gamma \vdash_{\beta} A' : \hat{B}$. Hence $A'$ has no $\Pi$-redexes by Lemma 4.7. \hfill $\Box$

Lemma 4.12 (Weak Subject Reduction for $\vdash_{\beta_{\Pi}}$ and $\rightarrow_{\beta_{\Pi}}$)

1. $\Gamma \vdash_{\beta_{\Pi}} A : B \land A \rightarrow_{\beta_{\Pi}} A' \Rightarrow \Gamma \vdash_{\beta_{\Pi}} A' : \hat{B}$

2. $\Gamma \vdash_{\beta_{\Pi}} A : B \land A \rightarrow_{\beta_{\Pi}} A' \land B$ is $\beta$-legal $\Rightarrow \Gamma \vdash_{\beta_{\Pi}} A' : B$

Proof: 1. From $\Gamma \vdash_{\beta_{\Pi}} A : B$, and Lemma 4.9, 1, $\Gamma \vdash_{\beta} A : \hat{B}$. Also, from $A \rightarrow_{\beta_{\Pi}} A'$, and $A$ and $A'$ have no $\Pi$-redexes (Lemmas 4.7 and 4.11), $A \rightarrow_{\beta} A'$. Now, from SR for $\rightarrow_{\beta}$ we get $\Gamma \vdash_{\beta} A' : \hat{B}$. Hence, by Lemma 4.9, 2, we get $\Gamma \vdash_{\beta_{\Pi}} A' : \hat{B}$. 2. is a corollary of 1. \hfill $\Box$

Corollary 4.13 (WSR Corollary for $\vdash_{\beta_{\Pi}}$ and $\rightarrow_{\beta_{\Pi}}$)

1. If $\Gamma \vdash_{\beta_{\Pi}} A : B$ and $\Gamma \rightarrow_{\beta_{\Pi}} \Gamma'$ then $\Gamma' \vdash_{\beta_{\Pi}} A : B$.

2. If $\Gamma \vdash_{\beta_{\Pi}} A : B$ and $B_1 \rightarrow_{\beta_{\Pi}} B_2$ then $\Gamma \vdash_{\beta_{\Pi}} A : B_2$.

3. If $\Gamma \vdash_{\beta_{\Pi}} A : B$ and $B \equiv_{\beta_{\Pi}} S$ then $\Gamma \vdash_{\beta_{\Pi}} A : S$.

4. If $A$ is $\Gamma^\ast_{\beta_{\Pi}}$-term and $A \rightarrow_{\beta_{\Pi}} A'$ then $A'$ is a $\Gamma^\ast_{\beta_{\Pi}}$-term.

Proof: 1. By an easy induction on $\Gamma \vdash_{\beta_{\Pi}} A : B$ using Lemma 4.12. 2. Use $\Gamma \vdash_{\beta} A : \hat{B}$, $\hat{B} \rightarrow_{\beta} \hat{B}'$ and SR for $\rightarrow_{\beta}$. 3. is a corollary of 2. 4. Case $\Gamma \vdash_{\beta_{\Pi}} A : B$ and $A \rightarrow_{\beta_{\Pi}} A'$ then it is easy to show $A'$ is $\beta_{\Pi}$-legal using Lemma 4.12. Here we show that if $\Gamma \vdash_{\beta_{\Pi}} B : A$ and $A \rightarrow_{\beta_{\Pi}} A'$ then $A'$ is $\beta_{\Pi}$-legal. We will only consider the case where $A \rightarrow_{\beta_{\Pi}} A'$ as the reflexivity and transitivity of $\rightarrow_{\beta_{\Pi}}$ are easy. There are only three cases to consider:

- Case $A \equiv \hat{A}$ then $A \rightarrow_{\beta} A'$ and by Lemma 4.9, 1, $\Gamma \vdash_{\beta} B : A$. Hence, $\Gamma \vdash_{\beta} B : A'$ by SR for $\rightarrow_{\beta}$ and so $\Gamma \vdash_{\beta_{\Pi}} B : A'$ by Lemma 4.9, 2.

- Case $A \equiv (\Pi_{x:D}.E)A$, $A' \equiv E[x := C]$ then by Lemma 4.9, 1, $\Gamma \vdash_{\beta} B : \hat{A} \equiv A'$. hence, $\Gamma \vdash_{\beta_{\Pi}} B : A'$ by Lemma 4.9, 2.
• Case $A \equiv (\Pi_{\otimes_D}E)C$, $A' \equiv (\Pi_{\otimes_D'}E')C'$, then $C, D, E$ are $\vdash$-legal, $B \equiv FC$, $\Gamma \vdash_{\Pi_1} F : \Pi_{\otimes_D}E$, $\Gamma \vdash_{\Pi_1} C : D$ and hence $\Gamma \vdash_{\Pi_1} F : \Pi_{\otimes_D}E$, $\Gamma \vdash_{\Pi_1} C : D$. So $\Gamma \vdash_{\Pi_1} F : \Pi_{\otimes_D'}E'$, $\Gamma \vdash_{\Pi_1} C' : D'$. Therefore, $\Gamma \vdash_{\Pi_1} F : \Pi_{\otimes_D'}E'$, $\Gamma \vdash_{\Pi_1} C' : D'$. Thus $\Gamma \vdash_{\Pi_1} FC' : (\Pi_{\otimes_D'})C'$.

Remark 4.14 We cannot replace 2 of Corollary 4.13 by: If $\Gamma \vdash_{\Pi_1} A : B$ and $B \rightarrow_{\Pi_1} B'$ then $\Gamma \vdash_{\Pi_1} A : B'$.

For example, take $\Gamma \equiv \lambda_{a.*} \lambda_{y.*} A \equiv (\lambda_{x.*} z)((\lambda_{x.*} z)y)$, $B \equiv (\Pi_{\otimes_D.\alpha})(\lambda_{x.*} z)y$ and $B' \equiv (\Pi_{\otimes_D.\alpha})y$. Then, $\Gamma \vdash_{\Pi_1} A : B$ but $\Gamma \not\vdash_{\Pi_1} A : B'$ because if otherwise, we get by construction, $\Gamma \vdash_{\Pi_1} (\Pi_{\otimes_D.\alpha})y : S$, absurd by Lemma 4.5.

Lemma 4.15 (Unicity of Types for $\vdash_{\Pi_1}$ and $\rightarrow_{\Pi_1}$)

1. $\Gamma \vdash_{\Pi_1} A : B \land \Gamma \vdash_{\Pi_1} A : B \Rightarrow B_1 =_{\Pi_1} B_2$

2. $\Gamma \vdash_{\Pi_1} A : B \land \Gamma \vdash_{\Pi_1} A' : B' \land A =_{\Pi_1} A' \Rightarrow B =_{\Pi_1} B'$

3. $\Gamma \vdash_{\Pi_1} B : S, B =_{\Pi_1} B', \Gamma \vdash_{\Pi_1} A' : B' \land \Gamma \vdash_{\Pi_1} B' : S$.

Proof: 1. by induction on the structure of $A$ using the generation lemma. 2. by Church Rosser, Weak Subject Reduction, 1, and Lemma 4.7. 3. This is the same as $\Gamma \vdash_{\Pi_1} B : S, B =_{\Pi_1} B', \Gamma \vdash_{\Pi_1} A' : B'$ then $\Gamma \vdash_{\Pi_1} B' : S$. Take for example $\Gamma \vdash_{\Pi_1} * : 0, * =_{\Pi_1} (\Pi_{\otimes_D.\alpha})y$ then $\Gamma \vdash_{\Pi_1} (\Pi_{\otimes_D.\alpha})y : S$.

Lemma 4.16 If $SN_{\rightarrow_{\Pi_1}}(B[x := C])$, $SN_{\rightarrow_{\Pi_1}}(A)$, $SN_{\rightarrow_{\Pi_1}}(B)$ and $SN_{\rightarrow_{\Pi_1}}(C)$ then $SN_{\rightarrow_{\Pi_1}}((\Pi_{\otimes_D.B})C)$.

Proof: This is standard.

Theorem 4.17 (Strong Normalisation with respect to $\vdash_{\Pi_1}$ and $\rightarrow_{\beta_\tau}$)

For all $\vdash_{\Pi_1}$-legal terms $A$, $SN_{\rightarrow_{\Pi_1}}(A)$; i.e. $A$ is strongly normalising with respect to $\rightarrow_{\beta_\tau}$.

Proof: Note that if $A$ is $\Pi$-redex free and $SN_{\rightarrow_\beta}(A)$ then $SN_{\rightarrow_{\Pi_1}}(A)$. We show that if $\Gamma \vdash_{\Pi_1} A : B$ then $SN_{\rightarrow_{\Pi_1}}(A)$ and $SN_{\rightarrow_{\Pi_1}}(B)$. By Lemma 4.9, 1, $\Gamma \vdash_{\Pi_1} A : B$. Hence, by Theorem 3.15, $SN_{\rightarrow_\beta}(A)$ and $SN_{\rightarrow_\beta}(B)$. Hence, $SN_{\rightarrow_{\Pi_1}}(A)$ and we only have to show that $SN_{\rightarrow_{\Pi_1}}(B)$.

• Case $B \equiv B$ then $SN_{\rightarrow_{\Pi_1}}(B)$.

• Case $B \equiv (\Pi_{\otimes_B}B_1.B_2)B_3$ then $B \equiv B_2[x := B_3]$, $B_1, B_2, B_3$ are $\vdash_\beta$-legal. By Lemma 4.16, $SN_{\rightarrow_{\Pi_1}}(B_i)$ for $1 \leq i \leq 3$ and $SN_{\rightarrow_{\Pi_1}}(B)$, we get $SN_{\rightarrow_{\Pi_1}}((\Pi_{\otimes_B}B_1.B_2)B_3)$.

5 The canonical typing operator $\tau$ and its properties

Definition 5.1 (Canonical Type Operator) For any pseudo-context $\Gamma$ and pseudo-expression $A$, we define the canonical type of $A$ in $\Gamma$, $\tau(\Gamma, A)$ as follows:

\[
\tau(\Gamma, *) \equiv \Box
\]

\[
\tau(\Gamma, x) \equiv A \text{ if } \lambda_x.A \in \Gamma
\]

\[
\tau(\Gamma, Fa) \equiv \tau(\Gamma, F)a
\]

\[
\tau(\Gamma, \lambda_x.A) \equiv \Pi_{\otimes_A}\tau(\Gamma, \lambda_x.A, B) \text{ if } x \notin \text{dom}(\Gamma)
\]

\[
\tau(\Gamma, \Pi_{\otimes_A}B) \equiv \tau(\Gamma, \lambda_x.A, B) \text{ if } x \notin \text{dom}(\Gamma)
\]
Example 5.2 In usual type theory, the type of $\lambda x_1 : \ldots : x_n : \ldots \lambda y_1 : \ldots : y_m : \ldots$ is $\Pi x_1 : \ldots : \Pi y_m : \ldots$, and the type of $\Pi x_1 : \ldots : \Pi y_m : \ldots$ is $\star$. Now, with our $\tau$, we get the same result:

$$\tau(\langle \rangle, \lambda x_1 : \ldots : x_n : \ldots, \lambda y_1 : \ldots : y_m : \ldots, y) \equiv \Pi x_1 : \ldots : \Pi y_m : \ldots$$

$$\tau(\langle \rangle, \Pi x_1 : \ldots : \Pi y_m : \ldots , x) \equiv \tau(\lambda x_1 : \ldots : x_n : \ldots, \lambda y_1 : \ldots : y_m : \ldots, x) \equiv \star$$

Remark 5.3 Note that $\tau(\Gamma, \Box)$ is undefined. We write $\tau(\Gamma, A)$ for $\tau(\Gamma, A)$ defined. Note also that $\text{FV}(\tau(\Gamma, A)) \neq \text{FV}(\Gamma, A)$. For example, if $\Gamma \equiv \lambda x_1 : \ldots : x_n : \lambda y_1 : \ldots : y_m : \ldots$, then $\tau(\Gamma, y) \equiv x$, $x \in \text{FV}(\tau(\Gamma, y)) \setminus \text{FV}(\Gamma, y)$, and $p \in \text{FV}(\Gamma, y)$.

In what follows, we study the properties of $\tau$.

Lemma 5.4 ($\tau$-weakening)
Let $\Gamma, \Gamma'$ be pseudo-contexts. $\Gamma \subseteq \Gamma' \land \tau(\Gamma, A) \Rightarrow [\uparrow \tau(\Gamma', A) \land \tau(\Gamma, A) \equiv \tau(\Gamma', A)]$.

Proof: By induction on $A$, noting that bound variables in $A$ can always be renamed so that they don't occur in $\text{dom}(\Gamma')$. $\Box$

Lemma 5.5 (Context-reduction for $\tau$)
For $\Gamma, \Gamma'$ be pseudo-contexts, $\Gamma \Rightarrow \beta \Gamma' \land \tau(\Gamma, A) \Rightarrow [\uparrow \tau(\Gamma', A) \land \tau(\Gamma, A) \Rightarrow \beta \tau(\Gamma', A)]$.

Proof: By induction on $\tau(\Gamma, A)$. $\Box$

Lemma 5.6 ($\tau$-restriction)
If $\tau(\Gamma, A)$ then $\tau(\Gamma, FV(A), A) \equiv \tau(\Gamma, A)$.

Proof: By induction on $A$. $\Box$

Lemma 5.7 ($\tau$-Substitution Lemma) Let $\sim$ be $\Rightarrow \beta \Pi, = \beta \Pi$ or $\equiv$.
If $\tau(\Gamma, \lambda x : A, \Delta, B) \equiv C$ and $\tau(\Gamma, D) \sim A$ then $\tau(\Gamma, (\Delta[x := D]), B[x := D]) \sim C[x := D]$.

Proof: By induction on the structure of $A$. $\Box$

Note that when $\Gamma, A$ contain no $\Pi$-redexes, $\tau(\Gamma, A)$ is exactly as $A$ except that:

1. An occurrence of $\pi x : B$ in $A$ which is not an occurrence in some $C$ where $\pi y : C, D$ or $DC$ is a subterm of $A$, disappears in the case $\pi = \Pi$ and becomes $\Pi x : B$ in the case $\pi = \lambda$.

2. $\Box(A)$ is replaced by $\tau(\Gamma', \Box(A))$ where $\Gamma' = \Gamma, \lambda z_1 : A_1 : \ldots : \lambda z_n : A_n$ and $z_1 : A_1$ are those of $\pi y : B$ which have either disappeared or been replaced by $\Pi y : B$, taken in the same order in which they appeared in $A$.

Example 5.8

$$\tau(\langle \rangle, \Pi x_1 : \ldots : \Pi y_m : \ldots , x)(\lambda y_1 : \ldots : y_m : \ldots, y) \equiv$$

$$\Pi y_1 : \ldots : \Pi y_m : \ldots, x$$

$$\tau(\langle \rangle, \Pi x_1 : \ldots : \Pi y_m : \ldots , x)(\lambda x_1 : \ldots : x_n : \ldots, x) \equiv$$

$$(\Pi x_1 : \ldots : \Pi y_m : \ldots , x)C \equiv$$

$$(\Pi x_1 : \ldots : \Pi y_m : \ldots , x)C$$

14
This can be made clearer by using the item notation via a translation function \( T \) where \( T(\pi x : A . B) \equiv (T(A)\pi x)I(B) \) and \( T(AB) \equiv (T(B))\delta T(A) \). Note that for each \( A \), \( T(A) \equiv I_{1}I_{2} \ldots I_{x} \) where each main item \( I_{i} \) is of the form \( (A_{i}\omega) \) for \( \omega \in \{ \delta \} \cup \{ \pi y ; y \in V \} \) and \( x \equiv \nabla (A) \). Moreover, any \( \pi \)-redex \( (\pi x : B . C)D \) in \( A \) will be \( (T(D))\delta (T(B)\pi y)I(C) \). Hence, \( \pi \)-redexes start by a \( \delta \)-item just before a \( \pi \)-item.

With this item notation, it is clearer to evaluate \( r \). In fact, we go through \( I(\pi x : A) \) from left to right and for every \( I_{i} \) we reach, we keep it unchanged if it is a \( \delta \)-item, we remove it if it is a \( \pi \)-item and we change the \( A \) to \( \pi \) if it is a \( \lambda \)-item. Finally, we replace \( \nabla (A) \) which is \( x \) by \( \tau (\Gamma',x) \) where \( \Gamma' \equiv \tau (\Gamma)I_{I_{i}} \) and \( I_{i} \) are all the \( \pi \)-items of \( A \) where \( \pi \) is changed to \( \lambda \). Of course, \( T(\tau (\pi x : A) \equiv \tau (T(\pi x : A)) \).

For example, for \( A \equiv I(\pi x : \lambda y . x \cdot y)I(\pi y : \lambda x y . x y) \),

\[
I(\pi x : A) \equiv (*\Pi x) \quad ((*\Pi y)(y\delta)(*\lambda y)z\delta) \quad (*\lambda y) \quad (y\delta) \quad (*\lambda y) \quad x
\]

\[
\tau (\langle \rangle, I(\pi x : A)) \equiv ((*\Pi y)(y\delta)(*\lambda y)z\delta) \quad (*\Pi y) \quad (y\delta) \quad (*\lambda y) \quad \tau ((*\lambda y)(*\lambda y)(*\lambda y), z)
\]

Note that \( I_{1} \) has disappeared, \( I_{2} \) and \( I_{4} \) remained unchanged whereas the \( \lambda \) in \( I_{3} \) and \( I_{5} \) changed to \( \pi \). Note also that \( I(\tau (\langle \rangle, A)) \equiv \tau (\langle \rangle, I(\pi x : A)) \). In item notation, every term is of the form \( Sx \) where \( S \) is a segment, i.e. a sequence of items. For a segment \( S \), we define \( S\lambda \) as \( S \) where all the main \( \pi \)-items are written as \( \lambda \)-items and where all the main \( \delta \)-items are removed. We define \( S\Pi \) as \( S \) where all the main \( \lambda \)-items are replaced by \( \Pi \)-items, all the main \( \pi \)-items remain unchanged and all the main \( \delta \)-items are removed. For example, if \( S \equiv (x\delta)(y\lambda x)(z\Pi r) \) then \( S\lambda \equiv (y\lambda x)(z\lambda r) \) and \( S\Pi \equiv (x\delta)(y\Pi x) \). With these notations, \( \tau (\pi x : A) \equiv S\Pi \tau (\pi x : Sx) \).

This item notation has been used to study, extend and clarify many notions of the \( \lambda \)-calculus (see [KN 93] and [KN 9y]).

### Remark 5.9
Note that typability of subterms fails for \( r \). That is, \( r \) can be defined for some \( A \) without being defined for all its subterms. For example, \( \tau (\langle \rangle, (\pi x : x \cdot y)) \equiv (\Pi x \cdot \cdot \cdot y) \), but \( \tau (\langle \rangle, y) \) is not defined. Note also that unicity of types fails for \( r \). That is, we can have \( A \rightarrow_{\beta_{1}} A' \) without having \( \tau (\pi x : A) \equiv \beta_{1} \tau (\pi x : A') \). For example, \( A \equiv (\lambda x . x \cdot x)(\lambda y . x y) \rightarrow_{\beta_{1}} \lambda y . x y \equiv A' \) yet \( \tau (\langle \rangle, A) \equiv (\Pi x \cdot \cdot \cdot x)(\lambda y . x y) \neq \beta_{1} \tau (\langle \rangle, \lambda y . x y) \equiv \Pi y \cdot \cdot \cdot y \). Moreover, \( S \rightarrow_{\beta_{1}} (\tau (\pi x : A)) \). For example, take \( \Gamma \equiv \lambda x \cdot (\Pi y \cdot \cdot \cdot x y)(\Pi y \cdot \cdot \cdot x y) \) and \( A \equiv x \). In Lemmas 6.7 and 6.17, we show that typability of subterms and unicity of types hold for \( r \) when \( \Gamma \vdash A \). We conjecture moreover, that if \( \Gamma \vdash A \) then \( \tau (\pi x : A) \) is strongly normalising.

### 6 The typability relation \( \vdash \) and its properties

#### Definition 6.1
\( \vdash \) The Typability relation \( \vdash \) is defined by the following rules:

\[ (\vdash \text{axiom}) \quad \langle \rangle \vdash * \]

\[ (\vdash \text{start rule}) \quad T_{\lambda x : A} \vdash D \quad \text{if} \quad vc \]

\[ (\vdash \text{weakening rule}) \quad T_{\lambda x : A} \vdash D \quad \text{if} \quad vc \]

\[ (\vdash \text{application rule}) \quad T_{\vdash F} \vdash a \quad \text{if} \quad ap \]
(\vdash\text{-}\text{abstraction rule}) \quad \frac{\Gamma, \lambda x:A \vdash b}{\Gamma \vdash \lambda x:A. b} \quad \text{if } ab

(\vdash\text{-}\text{formation}) \quad \frac{\Gamma \vdash A}{\Gamma \vdash \Pi x:A. B} \quad \text{if } fc

\text{vc (variable condition): } x \notin \Gamma \text{ and } \tau(\Gamma, A) \rightarrow_{\beta_1} S \text{ for some } S

\text{ap (application condition): } \tau(\Gamma, F) =_{\beta_1} \Pi x:A. B \text{ and } \tau(\Gamma, a) =_{\beta_1} A \text{ for some } A, B.

\text{ab (abstraction condition): } \tau(\Gamma, \lambda x:A, b) =_{\beta_1} B \text{ and } \tau(\Gamma, \Pi x:A. B) \rightarrow_{\beta_1} S \text{ for some } S.

\text{fc (formation condition): } \tau(\Gamma, A) \rightarrow_{\beta_1} S_1 \text{ and } \tau(\Gamma, \lambda x:A, B) \rightarrow_{\beta_1} S_2 \text{ for some } (S_1, S_2) \text{ rule.}

When \Gamma \vdash A, we say that A is typable in \Gamma.

\text{Lemma 6.2 (Free variable lemma and type-definability for } \vdash \text{ and } \tau)\]
\text{Let } \Gamma \equiv \lambda x_1:A_1, \ldots, \lambda x_n:A_n. \text{ If } \Gamma \vdash A. \text{ Then we have:}

1. The \( x_1 \ldots x_n \) are all distinct.

2. \( \text{FV}(A) \subseteq \{x_1, \ldots, x_n\} \).

3. \( \text{FV}(A_i) \subseteq \{x_1, \ldots, x_{i-1}\} \) for \( 1 \leq i \leq n \).

4. \( \uparrow \tau(\Gamma, A) \text{ and } \text{FV}(\tau(\Gamma, A)) \subseteq \{x_1, \ldots, x_n\} \).

\text{Proof: By induction on } \Gamma \vdash A. \quad \Box

\text{Lemma 6.3 (Start Lemma for } \vdash \text{ and } \tau)\]
\text{If } \Gamma \text{ is } \vdash\text{-}\text{legal, then } \Gamma \vdash \ast \text{ and } \forall \lambda x:C \in \Gamma[\Gamma \vdash z \land \tau(\Gamma, z) \equiv C].

\text{Proof: By induction on the derivation } \Gamma \vdash A. \quad \Box

\text{Lemma 6.4 (Substitution Lemma for } \vdash \text{ and } \tau)\]
\text{If } \Gamma, \lambda x:A, \Delta \vdash B \text{ and } \Gamma \vdash D \text{ and } \tau(\Gamma, D) =_{\beta_1} A \text{, then } \Gamma.(\Delta[x := D]) \vdash B[x := D] \text{ and } \tau(\Gamma.(\Delta[x := D]), B[x := D]) =_{\beta_1} \tau(\Gamma, \lambda x:A, \Delta, B)[x := D].

\text{Proof: By induction on the derivations of } \Gamma, \lambda x:A, \Delta \vdash B. \quad \Box

\text{Lemma 6.5 (Thinning Lemma for } \vdash \text{ and } \tau)\]
\text{If } \Gamma \text{ and } \Delta \text{ be } \vdash\text{-}\text{legal and } \Gamma \subseteq \Delta, \text{ then } \Gamma \vdash A \Rightarrow \Delta \vdash A \text{ (note that } \tau(\Gamma, A) \equiv \tau(\Delta, A)).

\text{Proof: By induction on the length of the derivations } \Gamma \vdash A. \quad \Box

\text{Lemma 6.6 (Generation Lemma for } \vdash \text{ and } \tau)\]
1. \( \Gamma \vdash S \Rightarrow S \equiv \ast. \)

2. \( \Gamma \vdash z \Rightarrow \exists A[\lambda x:A \in \Gamma \land \tau(\Gamma, z) \equiv A]. \)

3. \( \Gamma \vdash \Pi x:A. B \Rightarrow \exists S_1, S_2[\Gamma \vdash A \land \Gamma, \lambda x:A \vdash B \land \tau(\Gamma, A) =_{\beta_1} S_1 \land \tau(\Gamma, \lambda x:A, B) =_{\beta_1} S_2 \land (S_1, S_2) \text{ is a rule}]. \)

4. \( \Gamma \vdash \lambda x:A, b \Rightarrow \exists S, B[\Gamma \vdash \Pi x:A. B \land \Gamma, \lambda x:A \vdash b \land \tau(\Gamma, \lambda x:A, b) =_{\beta_1} B \land \tau(\Gamma, \Pi x:A. B) =_{\beta_1} S]. \)

5. \( \Gamma \vdash F a \Rightarrow \exists A, B, x[\Gamma \vdash F \land \Gamma \vdash a \land \tau(\Gamma, F) =_{\beta_1} \Pi x:A. B \land \tau(\Gamma, a) =_{\beta_1} A]. \)

\text{Proof: By induction on the derivations } \Gamma \vdash A. \quad \Box
Lemma 6.7 (Typability of subterms)
If \( \Gamma \vdash A \) and \( A' \) is a subexpression of \( A \) then (\( \exists \Gamma' \)[\( \Gamma, \Gamma' \vdash A' \)].

Proof: By induction on \( \Gamma \vdash A \).

Lemma 6.8 (Legal terms and contexts for \( \vdash \))
\( \vdash \)-legal terms and contexts are free of IT-redexes.

Proof: By induction on \( \Gamma \vdash A \). The only interesting case is application. Assume \( \Gamma \vdash F, \Gamma \vdash a, \tau(\Gamma, F) =_\Pi \Pi x:A.B \) and \( \tau(\Gamma, a) =_\Pi A \). By IH, \( \Gamma, F, a \) are IT-redexes free. Also, \( F \neq \Pi x:C.D \), otherwise, \( \tau(\Gamma, \lambda x:C.D) =_\Pi \tau(\Gamma, F) =_\Pi S_2 =_\Pi \Pi x:A.B \), absurd.

Note that \( \Gamma \vdash A \) \( \neq (\tau(\Gamma, A) \equiv \Box \lor \Gamma \vdash \tau(\Gamma, A)) \). For example, \( \lambda x.y \vdash (\lambda x.y)x \) and \( \lambda x.y \not\vdash (\Pi x.x)x \), by Lemma 6.8. The property however holds when \( \tau(\Gamma, A) \) is IT-redex free. We need first the following lemma:

Lemma 6.9
If \( \Gamma \vdash A, \Gamma \vdash B \) and \( A =_\beta B \) then \( \tau(\Gamma, A) =_\Pi \tau(\Gamma, B) \).

Proof: By induction on \( A =_\beta B \) using Lemmas 5.5 and 5.7.

Lemma 6.10
If \( \Gamma \vdash A \) and \( \Gamma \vdash F \) and \( \tau(\Gamma, A) \) \( = _\beta \Pi, \) \( \Gamma \vdash F \) \( = _\beta \Pi, \) \( \Gamma \vdash A \) \( = _\beta \Pi, \) \( \Gamma \vdash A \) \( = _\beta \Pi. \)

Proof: By induction on \( \Gamma \vdash A \) using Lemma 6.9 (application cannot apply otherwise, \( \tau(\Gamma, Fa) =_\beta \Pi (\Pi x:A.B)B \) \( =_\beta \Pi \tau(\Gamma, F) \equiv \Pi x:A.B \) \( \tau(\Gamma, Fa) \) \( =_\beta \Pi \) \( \Pi x:A.B \).)

Now, let us study the relationship between \( \vdash _\beta \Pi \) and \( \vdash \)

Lemma 6.11
If \( \Gamma \vdash \vdash _\beta \Pi A : B \) then \( \Gamma \vdash A \) \( \vdash _\beta \Pi B. \)

Proof: By induction on \( \Gamma \vdash A \).}

Definition 6.12
For \( A \) a pseudo-term, we take \( \overline{A} \) to be the \( \beta \Pi \)-normal form of \( A \).

Lemma 6.13
If \( \Gamma \vdash \vdash A \) then \( \vdash \tau(\Gamma, A) \) and \( \Gamma \vdash \vdash A : \tau(\Gamma, A) \).

Proof: By induction on \( \Gamma \vdash A \). We only treat three cases:

application: Assume \( \Gamma \vdash F \) and \( \Gamma \vdash a \) a give \( \Gamma \vdash Fa \) where the application condition (ac) holds and IH holds for the first two derivations. \( \tau(\Gamma, F) =_\Pi \Pi x:A.B \land \tau(\Gamma, a) =_\Pi A \) \( \Rightarrow \exists C, D \) where \( A \rightarrow _\Pi C, B \rightarrow _\Pi D, \tau(\Gamma, F) =_\Pi \Pi x:C.D \) and \( \tau(\Gamma, a) =_\Pi C \).

Moreover, by IH \( \Gamma \vdash _\beta \tau(\Gamma, F) : S \) (otherwise by Corollary 3.10, \( \Pi x:C.D \equiv \Box \) absurd). Note, use application on \( \Gamma \vdash _\beta a : C, \Gamma \vdash _\beta F : \Pi x:C.D \) to get \( \Gamma \vdash _\beta Fa : D[x := a] \).

Hence by Strong Normalisation of \( \vdash _\beta, \vdash D[x := a] \).

But, \( \tau(\Gamma, Fa) =_\Pi \tau(\Gamma, F)a =_\Pi (\Pi x:C.D)a =_\Pi D[x := a] \) and so \( \tau(\Gamma, Fa) \).

Now, by Corollary 3.10, \( \Gamma \vdash _\beta Fa : D[x := a] \equiv \Box \lor \exists S[\Gamma \vdash _\beta D[x := a] : S] \).

- Case \( D[x := a] \equiv \Box \) then \( \tau(\Gamma, Fa) \equiv D[x := a] \equiv D[x := a] \) and \( \Gamma \vdash _\beta Fa : \tau(\Gamma, Fa) \).
- Case \( \Gamma \vdash _\beta D[x := a] : S, \) then by SR for \( \vdash _\beta, \) as \( D[x := a] \rightarrow _\beta D[x := a] \), \( \Gamma \vdash _\beta D[x := a] : S \).

Now, use \( \Gamma \vdash _\beta Fa : D[x := a], \Gamma \vdash _\beta D[x := a] : S \) and \( D[x := a] =_\beta D[x := a] \) and conversion for \( \vdash _\beta \) to get \( \Gamma \vdash _\beta Fa : D[x := a] \). Hence, \( \Gamma \vdash _\beta Fa : \tau(\Gamma, Fa) \).
abstraction: assume $\Gamma \vdash \Pi_{x:A}.B$ and $\Gamma.\lambda_{x:A} b \vdash b$ imply $\Gamma \vdash \lambda_{x:A} b$ where $\tau(\Gamma.\lambda_{x:A} b) =_{\beta_\Pi} B$, and $\tau(\Gamma, \Pi_{x:A}.B) =_{\beta} S$. Hence, $\tau(\Gamma, \Pi_{x:A}.B) \equiv S$.

By IH, $\Gamma \vdash_{\beta} \Pi_{x:A}.B : \tau(\Gamma, \Pi_{x:A}.B) \equiv S$. Moreover, by ab as $\tau(\Gamma.\lambda_{x:A} b) =_{\beta_\Pi} B$, we get $B =_{\beta_\Pi} \tau(\Gamma.\lambda_{x:A} b)$. Hence, $\Pi_{x:A}.B \vdash_{\beta_\Pi} \Pi_{x:A}.\tau(\Gamma.\lambda_{x:A} b)$ and $\Gamma \vdash_{\beta} \Pi_{x:A}.\tau(\Gamma.\lambda_{x:A} b) : S$ by SR for $\vdash_{\beta}$.

Furthermore, by IH, $\Gamma, \lambda_{x:A} b \vdash_{\beta} b : \tau(\Gamma, \lambda_{x:A} b)$.

Now, use $\Gamma, \lambda_{x:A} b \vdash_{\beta} b : \tau(\Gamma, \lambda_{x:A} b)$, $\Gamma \vdash_{\beta} \Pi_{x:A}.\tau(\Gamma.\lambda_{x:A} b) : S$ and abstraction to get $\Gamma \vdash_{\beta} \lambda_{x:A} b : \Pi_{x:A}.\tau(\Gamma.\lambda_{x:A} b)$.

But $\Pi_{x:A}.\tau(\Gamma.\lambda_{x:A} b) \vdash_{\beta} \Pi_{x:A}.\tau(\Gamma.\lambda_{x:A} b) \equiv \tau(\Gamma, \lambda_{x:A} b)$.

Hence by Corollary 3.13, $\Gamma \vdash_{\beta} \lambda_{x:A} b : \tau(\Gamma, \lambda_{x:A} b)$.

Formation: Assume $\Gamma \vdash A$ and $\Gamma.\lambda_{x:A} b \vdash B$ give $\Gamma \vdash \Pi_{x:A}.B$ and IH holds for the first two derivations. Hence, $\Gamma \vdash \tau(\Gamma, A) =_{\beta_\Pi} \tau(\Gamma, A)$ and $\Gamma.\lambda_{x:A} b \vdash B : \tau(\Gamma.\lambda_{x:A} b)$. Hence, as $\tau(\Gamma, \Pi_{x:A}.B) \equiv \tau(\Gamma, \lambda_{x:A} b)$, we get $\tau(\Gamma, \Pi_{x:A}.B)$.

Furthermore, as by fe, $\tau(\Gamma, A) =_{\beta_\Pi} S_1$ and $\tau(\Gamma.\lambda_{x:A} b) =_{\beta_\Pi} S_2$, for some $(S_1, S_2)$ rule, we get $\tau(\Gamma, A) \equiv S_1$ and $\tau(\Gamma.\lambda_{x:A} b) \equiv S_2$.

Now, we use formation to get $\Gamma \vdash_{\beta} \Pi_{x:A}.B : \tau(\Gamma, \Pi_{x:A}.B)$.

\[\square\]

Lemma 6.14 (Subject Reduction for $\vdash$ and $\tau$)

$\Gamma \vdash A \wedge A \rightarrow_{\beta_\Pi} A' \Rightarrow [\Gamma \vdash A' \wedge \tau(\Gamma, A) =_{\beta_\Pi} \tau(\Gamma, A')]$

Proof: Use Lemmas 6.11, 6.13 and SR for $\vdash_{\beta}$.  \[\square\]

Corollary 6.15 (SR corollary for $\vdash$ and $\tau$)

1. If $\Gamma \vdash A$ and $\Gamma \rightarrow_{\beta} \Gamma'$ then $\Gamma' \vdash A$ and $\tau(\Gamma, A) =_{\beta_\Pi} \tau(\Gamma', A)$.

2. If $A$ is a $\Gamma^+$-term and $A \rightarrow_{\beta} A'$ then $A'$ is a $\Gamma^+$-term.

Proof: If $\Gamma \vdash_{\beta} A : \tau(\Gamma, A)$, then $\Gamma \vdash_{\beta} A' : \tau(\Gamma, A')$. Hence, by Lemma 6.11 $\Gamma' \vdash A$ and $\tau(\Gamma', A) =_{\beta_\Pi} \tau(\Gamma, A)$.  \[\square\]

Remark 6.16 Note that $\Gamma \vdash A$ and $A \rightarrow_{\beta} A' \neq \tau(\Gamma, A) \rightarrow_{\beta} \tau(\Gamma, A')$. For example, if $A : (\lambda_{x:w} z)y$ and $\Gamma \equiv \lambda_{x:w} \lambda_{y:(x:w)} y$, then $A \rightarrow_{\beta} y$, $\tau(\Gamma, A) \equiv (\Pi_{x:w} w)y \neq_{\beta} \tau(\Gamma, y)$.

Lemma 6.17 (Unicity of Types for $\vdash$ and $\tau$)

1. $\Gamma \vdash A \wedge \Gamma \vdash B \wedge A =_{\beta} B \Rightarrow \tau(\Gamma, A) =_{\beta_\Pi} \tau(\Gamma, B)$

Proof: Use CR and SR to show $\Gamma \vdash C$, $\tau(\Gamma, A) =_{\beta_\Pi} \tau(\Gamma, C) =_{\beta_\Pi} \tau(\Gamma, B)$.  \[\square\]

Theorem 6.18 (Strong Normalisation for $\vdash$ and $\tau$)

If $A$ is a $\Gamma^+$-legal, then $SN_{\rightarrow_{\beta}}(A)$ and $SN_{\rightarrow_{\beta}}(\tau(\Gamma, A))$.

Proof: By Lemma 6.13, $\Gamma \vdash_{\beta} A : \tau(\Gamma, A)$. Hence, by Theorem 3.15, $SN_{\rightarrow_{\beta}}(A)$ and $SN_{\rightarrow_{\beta}}(\tau(\Gamma, A))$.  \[\square\]

We believe that if $\Gamma \vdash A$ then $SN_{\rightarrow_{\beta}}(\tau(\Gamma, A))$. We leave this as an open problem for the moment.

Finally, note that from Lemmas 6.11, 6.13 and 4.9, $\Pi$-reduction is necessary for splitting $\Gamma \vdash A : B$ into $\Gamma \vdash A$ and $\tau(\Gamma, A) =_{\beta_\Pi} B$, yet $\vdash_{\beta_\Pi}$ is not necessary. This is shown by the following lemma (call $B \vdash_{\beta}$-legal type iff $B \equiv \square$ or $\Gamma \vdash_{\beta} B : S$ for some $\Gamma, S$).
Lemma 6.19
\[ \Gamma \vdash_\beta A : B \iff \Gamma \vdash A \land \tau(\Gamma, A) =_{\beta \Pi} B \land B \text{ is } \vdash_\beta \text{-legal type.} \]

Proof:

\( \Rightarrow \) By Lemma 4.9, \( \Gamma \vdash_{\beta \Pi} A : B \). Hence, by Lemma 6.11, \( \Gamma \vdash A \) and \( \tau(\Gamma, A) =_{\beta \Pi} B \).

Moreover, by Corollary 3.10, as \( \Gamma \vdash_\beta A : B \), \( B \) is \( \vdash_\beta \)-legal type.

\( \Leftarrow \) By Lemma 6.13, \( \uparrow \tau(\Gamma, A) \) and \( \Gamma \vdash_\beta A : \tau(\Gamma, A) \). Moreover, \( B \rightarrow_\beta \tau(\Gamma, A) \).

- Case \( B \equiv \Box \) then \( \tau(\Gamma, A) \equiv \Box \) and \( \Gamma \vdash_\beta A : B \).

- Case \( \Gamma \vdash_\beta B : S \) then by \( \Gamma \vdash_\beta A : \tau(\Gamma, A) \), \( B =_\beta \tau(\Gamma, A) \) and conversion, we get \( \Gamma \vdash_\beta A : B \).

References


19
| Computing Science Reports | Department of Mathematics and Computing Science  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Eindhoven University of Technology</td>
</tr>
</tbody>
</table>

**In this series appeared:**

<table>
<thead>
<tr>
<th>Volume</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>91/01</td>
<td>D. Alstein</td>
<td>Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14</td>
</tr>
<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. Katoen, 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>91/08</td>
<td>H. Schepers</td>
<td>Terminology and Paradigms for Fault Tolerance, p. 25</td>
</tr>
<tr>
<td>91/09</td>
<td>W.M.P.v.d.Aalst</td>
<td>Interval Timed Petri Nets and their analysis, p. 53</td>
</tr>
<tr>
<td>91/11</td>
<td>R.C. Backhouse, P.J. de Bruin, G. Malcolm, E. Voermans, J. van der Woude</td>
<td>Relational Catamorphism, p. 31</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>91/13</td>
<td>F. Rietman</td>
<td>A note on Extensionality, p. 21</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 process
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-ExSpect, 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.
92/21 F. Kamareddine

Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.

92/22 R. Nederpelt
F. Kamareddine

A useful lambda notation, p. 17.

92/23 F. Kamareddine
E. Klein

Nominalization, Predication and Type Containment, p. 40.

92/24 M. Codish
D. Dams
Eyal Yardeni

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

92/25 E. Poll

A Programming Logic for F0, p. 15.

92/26 T. H. W. Beelen
W. J. J. Stut
P. A. C. Verkoulen

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

92/27 B. Watson
G. Zwaan

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

93/01 R. van Geldrop

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

93/02 T. Verhoeff

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

93/03 T. Verhoeff

Quicksort for linked lists, p. 8.

93/04 E. H. L. Aarts
J. H. M. Korst
P. J. Zwietering

Deterministic and randomized local search, p. 78.

93/05 J. C. M. Baeten
C. Verhoef

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

93/06 J. P. Veltkamp

On the unavoidability of metastable behaviour, p. 29

93/07 P. D. Moerland

Exercises in Multiprogramming, p. 97

93/08 J. Verhoosel

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

93/09 K. M. van Hee


93/10 K. M. van Hee

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

93/11 K. M. van Hee


93/12 K. M. van Hee


93/13 K. M. van Hee

Systems Engineering: a Formal Approach


A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27

Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.

A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.

The Design of an Online Help Facility for ExSpect, p. 21.


A Typechecker for Bijective Pure Type Systems, p. 28.

Relational Algebra and Equational Proofs, p. 23.

Pure Type Systems with Definitions, p. 38.


Multi-dimensional Petri nets, p. 25.

Finding all minimal separators of a graph, p. 11.

A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.

GOLD, a Graph Oriented Language for Databases, p. 42.

On Vertex Ranking for Permutation and Other Graphs, p. 11.

Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.

| 93/33 | L. Loyens and J. Moonen | ILIAS, a sequential language for parallel matrix computations, p. 20. |
| 93/34 | J.C.M. Baeten and J.A. Bergstra | Real Time Process Algebra with Infinitesimals, p.39. |
| 93/36 | J.C.M. Baeten and J.A. Bergstra | Non Interleaving Process Algebra, p. 17. |
| 93/38 | C. Verhoef | A general conservative extension theorem in process algebra, p. 17. |
| 93/41 | A. Bijlsma | Temporal operators viewed as predicate transformers, p. 11. |
| 93/42 | P.M.P. Rambags | Automatic Verification of Regular Protocols in P/T Nets, p. 23. |
| 93/43 | B.W. Watson | A taxonomy of finite automata construction algorithms, p. 87. |
| 93/44 | B.W. Watson | A taxonomy of finite automata minimization algorithms, p. 23. |
| 93/48 | R. Gerth | Verifying Sequentially Consistently Consistent Memory using Interface Refinement, p. 20. |
| 94/01 | P. America  
M. van der Kammen  
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  
| 94/04 | J.C.M. Baeten  
J.A. Bergstra | Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18. |
| 94/05 | P. Zhou  
| 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  
| 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  
| 94/12 | T. Kloks  
D. Kratsch  
| 94/13 | R. Seljé | 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. Doombos | Mathematical Induction Made Calculational, p. 36. |
| 94/17 | S. Mauw  
M.A. Reniers | An Algebraic Semantics of Basic Message Sequence Charts, p. 9. |
Refining Reduction in the Lambda Calculus, p. 15.

The performance of single-keyword and multiple-keyword pattern matching algorithms, p. 46.

Beyond $\beta$-Reduction in Church's $\lambda \rightarrow$, p. 22.


The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and regular Expressions.

An algebraic semantics of Message Sequence Charts, p. 43.

Abstract Interpretation of Reactive Systems: Abstractions Preserving $\forall$CTL$^*$, $\exists$CTL$^*$ and CTL$^*$, p. 28.

K$_1$,$\lambda$-free and W$\omega$-free graphs, p. 10.

On the foundations of functional programming: a programmer's point of view, p. 54.


Correctness of Real Time Systems by Construction, p. 22.

Process Algebra with Feedback, p. 22.

A Boyer-Moore type algorithm for regular expression pattern matching, p. 22.


A formalization of the Ramified Type Theory, p. 40.

The Barendregt Cube with Definitions and Generalised Reduction, p. 37.

Delayed choice: an operator for joining Message Sequence Charts, p. 15.