The Barendregt cube with definitions and generalised reduction

The Barendregt Cube with Definitions and Generalised Reduction*

Roel Bloo †
Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O.Box 513
5600 MB Eindhoven, the Netherlands
email: bloo@win.tue.nl

and

Fairouz Kamareddine ‡
Department of Computing Science
17 Lilybank Gardens
University of Glasgow
Glasgow G12 8QQ, Scotland
email: fairouz@dcs.glasgow.ac.uk
fax + 44 41 -3304913

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 2, 1994

*We would like to thank Herman Geuvers, Stefan Khars and Tom Melham for their useful discussions and remarks.

†The first author is supported by the Netherlands Computer Science Research Foundation (SION) with financial support from the Netherlands Organisation for Scientific Research (NWO). He is also grateful to the Department of Computing Science, Glasgow University, for their financial support and hospitality during the work on this paper.

‡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. She is also grateful to the Netherlands Organisation (NWO) for Scientific Research for its financial support.

Abstract

In this paper, we propose to extend the Barendregt Cube by generalising reduction and by adding definition mechanisms. We show that this extension satisfies all the original properties of the Cube including Church Rosser, Subject Reduction and Strong Normalisation.

Keywords: Generalised Reduction, Definitions, Barendregt Cube, Church Rosser, Subject Reduction, Strong Normalisation.

Contents

1 Introduction

In this paper, we introduce and motivate definition mechanisms and generalised reduction, and we study their interrelationship. Most importantly, we show that the Barendregt Cube of [Barendregt 92], extended with these two concepts preserves all its old properties including Church Rosser, Subject Reduction and Strong Normalisation.

1.1 Why generalised reduction

In the classical λ-calculus, the notions of redex and of β-reduction are described as follows:

Definition 1.1 *(Redexes and β-reduction in classical notation)*
A redex is of the form $(\lambda_{x:B}.A)C$. One-step β-reduction $\rightarrow_\beta$ is the compatible relation generated out of the axiom β: $(\lambda_{x:B}.A)C \rightarrow_\beta A[x := C]$. Many step β-reduction $\twoheadrightarrow_\beta$, is the reflexive transitive closure of $\rightarrow_\beta$.

These notions are not as general as one might desire, as the following example shows.

Example 1.2 In the classical term $A \equiv ((\lambda_{x:P}.(\lambda_{y:Q}.\lambda_{z:R}.za)b)c)d$, we have for redexes: $(\lambda_{y:Q}.\lambda_{z:R}.za)b$ and $(\lambda_{x:P}.(\lambda_{y:Q}.\lambda_{z:R}.za)b)c$ (the fact that neither x nor y appear as free variables in their respective scopes does not matter here; this is just to keep the example simple and clear). There is however a third redex which is not immediately visible in the classical term; namely, $(\lambda_{z:R}.za)d$. Such a redex will only be visible after we have contracted the above two redexes (we will not discuss the order here). For example:
$((\lambda_{x:P}.(\lambda_{y:Q}.\lambda_{z:R}.za)b)c)d \rightarrow_\beta ((\lambda_{y:Q}.\lambda_{z:R}.za)b)d \rightarrow_\beta (\lambda_{z:R}.za)d \rightarrow_\beta da$

In classical notation, only the first two redexes are visible at first sight, yet all the three are *needed* to reach the normal form of A. The third could only be seen once we had contracted the first two redexes. There is however a need to make as many needed redexes as possible visible and even though the notion of a needed redex is undecidable, much work has been carried out in order to study some classes of needed redexes (as in [BKKS 87] and [Gardner 94]). Our proposal, is not only to make as many redexes as possible visible, but also to give newly visible redexes the possibility to be contracted before other ones.

Example 1.3 In example 1.2, we may want to contract the redex based on $(\lambda_{z:R}.-)d$ before we have contracted any of the redexes $(\lambda_{x:P}.-)c$ and $(\lambda_{y:Q}.-)b$.

Firstly, this view on reduction gives an appropriate tool for the study of some programming languages. For example, in lazy evaluation ([Launchbury 93]), some redexes get frozen while other ones are being contracted. Now, if we had the ability of choosing which redex to contract out of all visible redexes, rather than waiting for some redex to be evaluated first, then we can say that we have achieved a flexible system where we have control over what to contract rather than letting reductions force themselves in some order. Secondly, we think that an investigation concerning the *complete* class of visible redexes in a term gives a better understanding of reduction strategies, e.g. the optimal reductions as in [Lévy 80].

1.2 Why definition mechanisms

In many type theories and lambda calculi, there is no possibility to introduce definitions which are abbreviations for large expressions and which can be used several times in a program or

a proof. This possibility is essential for practical use, and indeed implementations of Pure Type Systems such as Coq ([Dow 91]), Lego ([LP 92]) and HOL ([GM 93]) do provide this possibility. But what are definitions and why are they attractive? Definitions are name abbreviating expressions and occur in contexts where we reason about terms.

Example 1.4 $\texttt{Let}\ id = (\lambda_{x:A}.x) : A \to A\ \texttt{in}\ (\lambda_{y:A\to A}.id)id$ defines id to be $(\lambda_{x:A}.x)$ in a more complex expression in which id occurs two times.

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

One of the advantages of the definition $\texttt{let}\ x : A\ \texttt{be}\ a\ \texttt{in}\ b$ over $(\lambda_{x:A}.b)a$ is that it is convenient to have the freedom of substituting only some of the occurrences of an expression in a given formula. Another advantage is efficiency; one evaluates a in $\texttt{let}\ x : A\ \texttt{be}\ a\ \texttt{in}\ \ b$ only once, even in lazy languages. A further advantage is that defining x to be a in b can be used to type b efficiently, since the type A of a has to be calculated only once. A disadvantage is that the definition may hide information, as is shown in the following example.

Example 1.5 Without definitions, it is not possible to type $\lambda_{y:x}.\lambda_{f:a\to a}.fy$ even when we somehow know that x is an abbreviation for a. This is because f expects an argument of type a, and y is of type x. Once we make use of the fact that x is defined to be a in our context, then y will have type a and the term will be typable.

Practical experiences with type systems show, however, that definitions are absolutely indispensable for any realistic application. Without defnitions, terms soon become forbiddingly complicated. By using definitions one can avoid such an explosion in complexity. This is, by the way, a very natural thing to do: the apparatus of mathematics, for instance, is unimaginable without definitions.

Introducing definitions in Pure Type Systems is an interesting subject of research at the moment. For example, [SP 93] extended PTS's with definitions. Our approach enables such extension in an elegant way. In fact, the generated type derivations for terms in the Cube with definitions become much shorter than those in the absence of definitions (see Section 7.2). Moreover, we do not have to use complex relations to introduce definitions as in [SP 93]. Rather, the extension will be a natural way to how our terms are written. Basic for our proposed extensions is a new notation: *the item notation.*

1.3 The item notation for definitions and generalised reduction

The *item notation* is a simple variant of the usual notation where the argument is given before the function, the type is given before the abstraction operator, and where the parentheses are grouped differently than those of the classical notation. So that, if $\mathcal{I}$ translates classical terms into our notation, then $\mathcal{I}(AB)$ is written as $(\mathcal{I}(B)\delta)\mathcal{I}(A)$ (here is δ a special symbol used for application) and $\mathcal{I}(\mathcal{O}_{x:A}.B)$ is written as $(\mathcal{I}(A)\mathcal{O}_x)\mathcal{I}(B)$ where $\mathcal{O} = \lambda$ or Π. Both $(t\delta)$ and $(t\mathcal{O}_x)$, t being a term in item notation, are called **items**. For reasons explaining the usefulness of such a notation, the reader is referred to [KN 93] and [KN 9z]. For this paper however, the reader is to notice that redexes and definitions can be easily generalised and

introduced with item notation. A traditional redex is a term that starts with a δ-item next to a λ-item. A definition is itself a certain form of a δ-item next to a λ-item.

Example 1.6 $\mathcal{I}((\lambda_{x:A\to(B\to C)}.\lambda_{y:A}.xy)t) \equiv (t\delta)(A \to (B \to C)\lambda_x)(A\lambda_y)(y\delta)x$. The items are $(t\delta)$, $(A \to (B \to C)\lambda_x)$, $(A\lambda_y)$ and $(y\delta)$. The definition is $(t\delta)(A \to (B \to C)\lambda_x)$ and the redex is the whole term.

Definition 1.7 *(Classical redexes and β-reduction in item notation)*
*In the item notation of the λ-calculus, a classical redex is of the form $(C\delta)(B\lambda_x)A$. We call the pair $(C\delta)(B\lambda_x)$, a $\delta\lambda$-**pair**, or a $\delta\lambda$-**segment**. The classical β-reduction axiom is: $(C\delta)(B\lambda_x)A \to_\beta A[x := C]$. One and many step β-reduction are defined as in Definition 1.1.*

In item notation, term A of Example 1.2 becomes $(d\delta)(c\delta)(P\lambda_x)(b\delta)(Q\lambda_y)(R\lambda_z)(a\delta)z$. Here, the two classical redexes correspond to $\delta\lambda$-pairs as follows:

1. $(\lambda_{y:Q}.\lambda_{z:R}.za)b$ corresponds to $(b\delta)(Q\lambda_y)$. The remainder of the redex, $(R\lambda_z)(a\delta)z$, is the maximal subterm of A to the right of $(Q\lambda_y)$.

2. $(\lambda_{x:P}.(\lambda_{y:Q}.\lambda_{z:R}.za)b)c$ corresponds to $(c\delta)(P\lambda_x)$, the rest being $(b\delta)(Q\lambda_y)(R\lambda_z)(a\delta)z$.

Looking closely at A written in item notation, one sees that the third redex described in Example 1.2 is obtained by just matching δ and λ-items. $(\lambda_{z:R}.za)d$ is visible as it corresponds to the matching $(d\delta)(R\lambda_z)$ where $(d\delta)$ and $(R\lambda_z)$ are separated by the segment $(c\delta)(P\lambda_x)(b\delta)(Q\lambda_y)$. Hence, by extending the notion of a redex from being a δ-item adjacent to a λ-item, to being a matching pair of δ- and λ-items, we can make more redexes visible. Such an extension is simple, as in $(C\delta)\overline{s}(B\lambda_x)$, we say that $(C\delta)$ and $(B\lambda_x)$ match if $\overline{s}$ has the same structure as a matching composite of opening and closing brackets, each δ-item corresponding to an opening bracket and each λ-item corresponding to a closing bracket. For example, in A above, $(d\delta)$ and $(R\lambda_z)$ match as $(c\delta)(P\lambda_x)(b\delta)(Q\lambda_y)$ has the bracketing structure $[\,][\,]$ (see Figure 1). We refine β-reduction by changing (β) from $(C\delta)(B\lambda_x)A \to_\beta A[x := C]$ to

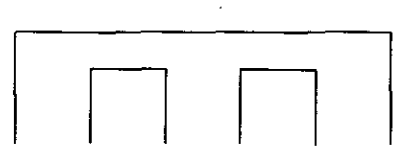

Figure 1: Extended redexes in item notation

$(C\delta)\overline{s}(B\lambda_x)A \hookrightarrow_\beta \overline{s}(A[x := C])$ if $(C\delta)$ and $(B\lambda_x)$ match. It is this generalised reduction that we will put on the top of the Cube and we will investigate its properties.

Now, what about definitions? The first step is to define definitions as matching $\delta\lambda$-couples and to include them in contexts with the condition that if a definition occurs in a context then it can be used anywhere in the term we are reasoning about in that context. Hence, if we look at Example 1.5, then we can type the term now that we allow definitions to occur in contexts and we extend $\vdash$ slightly so that it can see what is in its context.

Example 1.8 We use as context the segment $(a\delta)((A\lambda_x)(x\lambda_y)(a \to a\lambda_f)$, establishing that x of type A is defined as a, that y has type x and that f has type $a \to a$. Then, making use

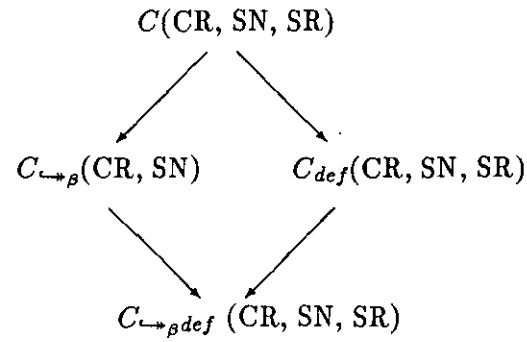

Figure 2: Properties of the Cube with various extensions

of this definition, we have
$(a\delta)(A\lambda_x)(x\lambda_y)(a \to a\lambda_f) \vdash f : a \to a$
$(a\delta)(A\lambda_x)(x\lambda_y)(a \to a\lambda_f) \vdash y : x = a$
$(a\delta)(A\lambda_x)(x\lambda_y)(a \to a\lambda_f) \vdash (y\delta)f : a$
$(a\delta)(A\lambda_x)(x\lambda_y) \vdash (a \to a\lambda_f)(y\delta)f : (a \to a) \to a$
$(a\delta)(A\lambda_x) \vdash (x\lambda_y)(a \to a\lambda_f)(y\delta)f : x \to (a \to a) \to a \;=\; a \to (a \to a) \to a$

Based on the above discussion, we divide the paper into the following sections:

- In Section 2, we introduce the item notation.
- In Section 3, we recall the Cube as in [Barendregt 92], and all its properties.
- In Section 4, we add to the Cube as in [Barendregt 92], generalised reduction $\hookrightarrow_\beta$ and show that $\hookrightarrow\!\!\!\!\to_\beta$ (the reflexive transitive closure of $\hookrightarrow_\beta$) generalises $\twoheadrightarrow_\beta$ (Lemma 4.3) such that $=_\beta$ and $\approx_\beta$ are the same (Lemma 4.5). This means that almost all the original properties still hold for $\hookrightarrow\!\!\!\!\to_\beta$. However, Church Rosser (CR) , Subject Reduction (SR) and Strong Normalisation (SN) deserve special attention. CR and SN are shown to hold (without the need for SR in the case of SN). SR holds in $\lambda\underline{\omega}$ and $\lambda\_$, but fails in the remaining systems. This problem is solved in Section 6 by adding definitions.
- In Section 5 we add definitions to the Cube of [Barendregt 92] and show that all the properties of [Barendregt 92] (including SR) hold with definitions. CR is not touched with the addition of definitions, contrary to the account of [SP 93], where a reduction relation was introduced to capture definitions and hence CR had to be shown.
- In Section 6, we extend the Cube with both generalised reduction and definitions. We show that the Cube extended with definitions and generalised reduction, preserves all its important properties. We present in particular, the general proof of Strong Normalisation which applies to all earlier systems.
- In Section 7, we discuss the conservativity of the Cube with definitions, with respect to the Cube without definitions. We show that more terms are typable using definitions. However, when a judgment is derivable in a system of the Cube with definitions, the judgment itself where all the definitions are unfolded is derivable without definitions (Theorem 7.3). We also compare our system of definitions with that of [SP 93].

Figure 2 summarizes our results, showing that one can safely use the Cube with definitions only, or with both definitions and generalised reduction. When using generalised reduction without definitions, one must remain in the $\lambda\_$ and $\lambda\underline{\omega}$ as the other systems lose their SR.

2 The item notation

Let $\mathcal{I}$ translate terms in classical notation to terms in item notation such that:

$$\begin{array}{lcll} \mathcal{I}(A) & = & A & \text{if } A \text{ is a variable or a constant} \\ \mathcal{I}(\mathcal{O}_{x:A}.B) & = & (\mathcal{I}(A)\mathcal{O}_x)\mathcal{I}(B) & \text{if} \mathcal{O} = \lambda \text{ or } \Pi \\ \mathcal{I}(AB) & = & (\mathcal{I}(B)\delta)\mathcal{I}(A) & \end{array}$$

Notation 2.1 *Throughout the whole paper, we take $\mathcal{O}$ to range over $\{\lambda, \Pi\}$.*

The systems of the Cube are based on a set of *pseudo-expressions* $\mathcal{T}$ defined by:

$$\mathcal{T} = V \mid C \mid (\mathcal{T}\delta)\mathcal{T} \mid (\mathcal{T}\mathcal{O}_V)\mathcal{T}$$

where V and C are infinite collections of variables and constants respectively. We assume that $x, y, z, \ldots$ range over V and we take two special constants $*$ and $\Box$. These constants are called sorts and the meta-variables $S, S_1, S_2, \ldots$ are used to range over the set of sorts $\mathcal{S} = \{*, \Box\}$. We take $A, B, C, a, b \ldots$ to range over pseudo-expressions. Parentheses will be omitted when no confusion occurs. For convenience sake, we divide V in two disjoint sets V^* and $V^\Box$, the sets of object respectively constructor variables. We take $x^*, y^*, z^*, \ldots$ to range over V^* and $x^\Box, y^\Box, z^\Box, \ldots$ to range over $V^\Box$.

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 $(A\lambda_x)x \equiv (A\lambda_y)y$ where $\equiv$ is used to denote syntactical equality of terms. We assume moreover, the Barendregt variable convention which is formally stated as follows:

Convention 2.2 *(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 λ's have different variables as subscript. Hence, we will not have $(x\delta)((A\lambda_x)x\lambda_x)x$, but $(x\delta)((A\lambda_y)y\lambda_z)z$ instead.

Definition 2.3 *(Compatibility)*
Let $\omega \in \{\delta\} \cup \{\mathcal{O}_x \mid x \in V\}$. A relation $\rightarrow$ on terms is compatible iff the following holds:

$$\frac{A_1 \rightarrow A_2}{(A_1\omega)B \rightarrow (A_2\omega)B} \qquad \frac{B_1 \rightarrow B_2}{(A\omega)B_1 \rightarrow (A\omega)B_2}$$

Definition 2.4 *(β-reduction $\rightarrow_\beta$ for the Cube)*
In the Cube, β-reduction $\rightarrow_\beta$, is the least compatible relation generated out of:

$$(\beta) \qquad (C\delta)(B\lambda_x)A \rightarrow_\beta A[x := C]$$

We take $\twoheadrightarrow_\beta$ to be the reflexive transitive closure of $\rightarrow_\beta$ and we take $=_\beta$ to be conversion, i.e. the least equivalence relation generated by $\twoheadrightarrow_\beta$.

Note that β is not assumed for Π-expressions, i.e. $(C\delta)(B\Pi_x)A \not\rightarrow_\beta A[x := C]$ (see [KN 9y] for such an extension). Now, some needed machinery for item notation follows.

Definition 2.5 *((main) items, (main, $\delta\mathcal{O}$-)segments, end variable, weight)*

- *If x is a variable, and A is a pseudo-expression then $(A\lambda_x), (A\Pi_x)$ and $(A\delta)$ are items (called λ-item, Π-item and δ-item respectively). We use $s, s_1, s_i, \ldots$ to range over items.*
- *A concatenation of zero or more items is a* **segment**. *We use $\overline{s}, \overline{s}_1, \overline{s}_i, \ldots$ as meta-variables for segments. We write $\emptyset$ for the empty segment.*
- *Each pseudo-expression A is the concatenation of zero or more items and a variable or constant: $A \equiv s_1 s_2 \cdots s_n x$. These items $s_1, s_2, \ldots, s_n$ are called the* **main items** *of A, x is called the* **end variable** *of A, notation* $\mathtt{endvar}(A)$.
- *Analogously, a segment $\overline{s}$ is a concatenation of zero or more items: $\overline{s} \equiv s_1 s_2 \cdots s_n$; again, these items $s_1, s_2, \ldots, s_n$ (if any) are called the* **main items**, *this time of $\overline{s}$.*
- *A concatenation of adjacent main items (in A or $\overline{s}$), $s_m \cdots s_{m+k}$, is called a* **main segment** *(in A or $\overline{s}$).*
- *A* **$\delta\mathcal{O}$-segment** *is a δ-item immediately followed by an $\mathcal{O}$-item.*
- *The* **weight** *of a segment $\overline{s}$,* $\mathtt{weight}(\overline{s})$, *is the number of main items that compose the segment. Moreover, we define* $\mathtt{weight}(\overline{s}x) = \mathtt{weight}(\overline{s})$.

When one desires to start a β-reduction on the basis of a certain δ-item and a λ-item occurring in one segment (recall, no reductions are based on δ- and Π-items), the *matching* of the δ and the λ in question is the important thing, even when the δ- and λ-items are separated by other items. I.e., the relevant question is whether they *may* together become a $\delta\lambda$-segment after a number of β-steps. This depends solely on the structure of the intermediate segment. If such an intermediate segment has a certain form (to be called: well-balanced), then the δ-item and the λ-item match and β-reduction based on these two items may take place. Some well-balanced segments play another important role. They may act as a definition. For example, $(A\delta)(B\lambda_x)C$ may mean: we define x of type B to be A in C. Sometimes, definitions are interleaved as in $(A_1\delta)(B_1\delta)(B_2\lambda_x)(A_2\lambda_y)D$ where the definition "x becomes B_1" is used inside the definition "y becomes A_1". We will assume definitions not to contain Π-items in this paper. Extending this work to the case where for example $(A\delta)(B\Pi_x)$ is a definition has yet to be investigated. In what follows we define well-balanced segments.

Definition 2.6 *(well-balanced segments)*

- *The empty segment $\emptyset$ is a well-balanced segment.*
- *If $\overline{s}$ is well-balanced, then $(A\delta)\overline{s}(B\mathcal{O}_x)$ is well-balanced.*
- *The concatenation of well-balanced segments is a well-balanced segment.*

A well-balanced segment has the same structure as a matching composite of opening and closing brackets, each δ- (or $\mathcal{O}$-)item corresponding with an opening (resp. closing) bracket. In a definition, the first [matches the last] and no Π-items are allowed. A $\delta\mathcal{O}$-couple is a main δ-item and a main $\mathcal{O}$-item separated by a well-balanced segment. Such a couple is reducible in case $\mathcal{O} = \lambda$. The δ-item and $\mathcal{O}$-item of the $\delta\mathcal{O}$-couple are said to match and each of them is called a partner or a partnered item. The non-partnered items in a segment are called bachelor. The following definition summarizes all this:

Definition 2.7 *(match, $\delta\mathcal{O}$- (reducible) couple, partner, partnered item, bachelor item)* *Let $A \in \mathcal{T}$. Let $\overline{s} \equiv s_1 \cdots s_n$ be a segment occurring in A.*

- *We say that s_i and s_j* **match**, *when $1 \leq i < j \leq n$, s_i is a δ-item, s_j is a $\mathcal{O}$-item, and the sequence $s_{i+1}, \ldots, s_{j-1}$ forms a well-balanced segment.*
- *If s_i and s_j match, $s_i s_j$ is a* **$\delta\mathcal{O}$-couple**. *If $\mathcal{O} = \lambda$ then $s_i s_j$ is a* **reducible couple**.
- *When s_i and s_j match, we call both s_i and s_j the* **partners** *in the $\delta\mathcal{O}$-couple. We also say that s_i and s_j are* **partnered** *items.*
- *All non-partnered $\mathcal{O}$- (or δ-)items s_k in A, are called* **bachelor** *$\mathcal{O}$- (resp. δ-)items.*

Example 2.8 In $\overline{s} \equiv (a\lambda_x)(b\lambda_y)(c\delta)(d\lambda_z)(e\lambda_u)(f\delta)(g\delta)(h\delta)(i\lambda_v)(j\lambda_w)(k\delta)$:

- $(c\delta)$ matches with $(d\lambda_z)$, $(h\delta)$ matches with $(i\lambda_v)$ and $(g\delta)$ with $(j\lambda_w)$. The segments $(c\delta)(d\lambda_z)$ and $(h\delta)(i\lambda_v)$ are $\delta\lambda$-segments (and $\delta\lambda$-couples). There is another $\delta\lambda$-couple in $\overline{s}$, viz. the couple of $(g\delta)$ and $(j\lambda_w)$.
- $(c\delta)$, $(d\lambda_z)$, $(g\delta)$, $(h\delta)$, $(i\lambda_v)$ and $(j\lambda_w)$, are the partnered main items of $\overline{s}$. $(a\lambda_x)$, $(b\lambda_y)$, $(e\lambda_u)$, $(f\delta)$ and $(k\delta)$ are bachelor items.
- $(g\delta)(h\delta)(i\lambda_v)(j\lambda_w)$ is a well-balanced segment.

Definition 2.9 *(definitions, definition application)*

- *If $\overline{s}$ is well-balanced and does not contain main Π-items, then a segment $(A\delta)\overline{s}(B\lambda_x)$ occurring in a context is called a definition.*
- *Let $\overline{s}$ be a well-balanced segment, occurring in a context, which consists of definitions and $A \in \mathcal{T}$. We define the application of the definitions of $\overline{s}$ in A, $[A]_{\overline{s}}$ inductively as follows: $[A]_{\emptyset} \equiv A$, $[A]_{(B\delta)\overline{s_1}(C\lambda_x)} \equiv [A[x := B]]_{\overline{s_1}}$ and $[A]_{\overline{s_1}\,\overline{s_2}} \equiv [[A]_{\overline{s_2}}]_{\overline{s_1}}$. Note that substitution takes place from right to left and that when none of the binding variables of $\overline{s}$ are free in A, then $[A]_{\overline{s}} \equiv A$.*

Remark 2.10 The definitions of well-balanced segments and definitions together are equivalent to the following definition (which we use sometimes in proofs by induction):

1. $\emptyset$ is well-balanced.
2. If $\overline{s_1}, \overline{s_2}$ are well-balanced, then $(A\delta)\overline{s_1}(B\mathcal{O}_x)\overline{s_2}$ is well-balanced.
3. If $\overline{s}$ is well-balanced and does not contain main Π-items, then $(A\delta)\overline{s}(B\lambda_x)$ is a definition.

Remark 2.11 We maintain the same liberal attitude for definitions, as we did for generalised redexes. That is, not only $(A\delta)(B\lambda_x)$ may act as a definition in a context, but also $(A\delta)\overline{s}(B\lambda_x)$ for any well-balanced segment $\overline{s}$ without main Π-items.

Note that we speak of definitions when such an $(A\delta)\overline{s}(B\lambda_x)$ occurs in a context; otherwise, when $(A\delta)\overline{s}(B\lambda_x)$ occurs in a term, we speak of a β-redex.

3 The ordinary typing relation and its properties

We now introduce some general notions concerning typing rules which are the same as the usual ones when we do not allow definitions in the context (as is the case in the λ-cube of [Barendregt 92]). When definitions are present however, the notions are more general.

Definition 3.1 *(declarations, pseudocontexts, $\subseteq'$)*

1. *A declaration is a λ-item. In a declaration $d \equiv (A\lambda_x)$, we define* $\mathtt{subj}(d)$, $\mathtt{pred}(d)$ *and* $\underline{d}$ *to be x, A and $\emptyset$ respectively.*

2. *For a definition $d \equiv (B\delta)\overline{s}(A\lambda_x)$ we define* $\mathtt{subj}(d)$, $\mathtt{pred}(d)$, $\underline{d}$ *and* $\mathtt{def}(d)$ *to be x, A, $\overline{s}$ and B respectively.*

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

4. *A pseudocontext is a concatenation of declarations and definitions such that if $(A\lambda_x)$ and $(B\lambda_y)$ are two different main items of the pseudocontext, then $x \not\equiv y$. We use $\Gamma, \Delta, \Gamma', \Gamma_1, \Gamma_2, \ldots$ to range over pseudocontexts.*

5. *For Γ a pseudocontext we define*

   $dom(\Gamma) = \{x \in V \mid (A\lambda_x)$ *is a main λ-item in Γ for some $A\}$,*

   Γ-$\mathtt{decl} = \{s \mid s$ *is a bachelor main λ-item of $\Gamma\}$,*

   Γ-$\mathtt{def} = \{\overline{s} \mid \overline{s} \equiv (A\delta)\overline{s}_1(B\lambda_x)$ *is a main segment of Γ where $\overline{s}_1$ is well-balanced $\}$,*

   Note that $dom(\Gamma) = \{\mathtt{subj}(d) \mid d \in \Gamma\text{-}\mathtt{decl} \cup \Gamma\text{-}\mathtt{def}\}$.

6. *Define $\subseteq'$ between pseudocontexts as the least reflexive transitive relation satisfying:*

   - $\Gamma\Delta \subseteq' \Gamma(C\lambda_x)\Delta$ *if no λ-item in Δ matches a δ-item in Γ*
   - $\Gamma\underline{d}\Delta \subseteq' \Gamma d\Delta$ *if d is a definition*
   - $\Gamma\overline{s}(A\lambda_x)\Delta \subseteq' \Gamma(D\delta)\overline{s}(A\lambda_x)\Delta$ *if $(A\lambda_x)$ is bachelor in $\Gamma\overline{s}(A\lambda_x)\Delta$, $\overline{s}$ is well-balanced*

Example 3.2 If $\Gamma \equiv (a\lambda_x)(b\lambda_y)(c\delta)(d\lambda_z)(e\lambda_u)(f\delta)(g\delta)(i\lambda_v)(j\lambda_w)$ then Γ-$\mathtt{decl} = \{(a\lambda_x), (b\lambda_y), (e\lambda_u)\}$ and Γ-$\mathtt{def} = \{(c\delta)(d\lambda_z), (f\delta)(g\delta)(i\lambda_v)(j\lambda_w), (g\delta)(i\lambda_v)\}$. Furthermore $\Gamma \subseteq' (*\lambda_r)(a\lambda_x)(b\lambda_y)(h\delta)(c\delta)(d\lambda_z)(k\lambda_{r'})(l\delta)(e\lambda_u)(f\delta)(g\delta)(i\lambda_v)(j\lambda_w)$.

Note that $\Gamma \subseteq' \Gamma' \not\Rightarrow \Gamma$-$\mathtt{decl} \subseteq \Gamma'$-$\mathtt{decl}$, but $\Gamma \subseteq' \Gamma' \Rightarrow \Gamma$-$\mathtt{def} \subseteq \Gamma'$-$\mathtt{def}$.

Definition 3.3 *(statements, judgements, $\prec$)*

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 Γ 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 Γ be a pseudocontext and $d \in \Gamma$-$\mathtt{def} \cup \Gamma$-$\mathtt{decl}$, Γ invites d, notation $\Gamma \prec d$, iff*

   - Γd *is a pseudocontext*
   - $\Gamma \vdash \mathtt{pred}(d) : S$ *for some sort S and* $\mathtt{subj}(d) \in V^S$.

- *if d is a definition then* $\Gamma \vdash \mathtt{def}(d) : \mathtt{pred}(d)$

Definition 3.4 *(Definitional β-equality) For all legal contexts Γ we define the binary relation $\Gamma \vdash \cdot =_{\mathtt{def}} \cdot$ to be the equivalence relation generated by*

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

Remark 3.5 If no definitions are present in Γ then $\Gamma \vdash A =_{\mathtt{def}} B$ is the same as $A =_\beta B$.

Definition 3.6 *Let Γ be a pseudocontext and A be a pseudo-expression.*

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 $\Gamma \vdash \mathtt{subj}(d) : \mathtt{pred}(d)$.*
   - *If d is a definition: $\Gamma \vdash d$ iff $\Gamma \vdash \mathtt{subj}(d) : \mathtt{pred}(d) \wedge \Gamma \vdash \mathtt{def}(d) : \mathtt{pred}(d) \wedge \Gamma \vdash \underline{d} \wedge \Gamma \vdash \mathtt{subj}(d) =_{\mathtt{def}} \mathtt{def}(d)$.*
   - *$\Gamma \vdash d_1 \cdots d_n$ iff $\Gamma \vdash d_i$ for all $1 \leq i \leq n$.*
2. *Γ is called legal if $\exists P, Q \in \mathcal{T}$ such that $\Gamma \vdash P : Q$.*
3. *$A \in \mathcal{T}$ is called a Γ-term if $\exists B \in \mathcal{T}[\Gamma \vdash A : B$ or $\Gamma \vdash B : A]$.*
   We take Γ-terms $= \{A \in \mathcal{T} \mid \exists B \in \mathcal{T}[\Gamma \vdash A : B \vee \Gamma \vdash B : A]\}$.
4. *We take Γ-kinds $= \{A \mid \Gamma \vdash A : \Box\}$ and Γ-types $= \{A \in \mathcal{T} \mid \Gamma \vdash A : *\}$.*
5. *$A \in \mathcal{T}$ is called a Γ-element if $\exists B \in \mathcal{T} \exists S \in \mathcal{S}[\Gamma \vdash A : B$ and $\Gamma \vdash B : S]$. We have two categories of elements: constructors and objects. We take Γ-constructors $= \{A \in \mathcal{T} \mid \exists B \in \mathcal{T}[\Gamma \vdash A : B : \Box]\}$ and Γ-objects $= \{A \in \mathcal{T} \mid \exists B \in \mathcal{T}[\Gamma \vdash A : B : *]\}$.*
6. *$A \in \mathcal{T}$ is called legal if $\exists \Gamma[A \in \Gamma$-terms$]$. Moreover, A is an X, if $\exists \Gamma[A \in \Gamma$-$Xs]$ for $X \in \{$type, term, kind, object, constructor$\}$.*

In the Cube as presented in [Barendregt 92], the only declarations allowed are of the form $(A\lambda_x)$. Hence there are no definitions. Therefore, $\Gamma \prec d$ is of the form $\Gamma \prec (A\lambda_x)$ and means that $\Gamma \vdash A : S$ for some S and that x is fresh in Γ, A. Moreover, for any $d \equiv (A\lambda_x)$, remember that $\underline{d} \equiv \emptyset$, $\mathtt{subj}(d) \equiv x$ and $\mathtt{pred}(d) \equiv A$. **Hence, in this section, d is a meta-variable for declarations only and $=_{\mathtt{def}}$ is the same as $=_\beta$ (which is independent of $\vdash$).**

3.1 The typing relation

Definition 3.7 *(Axioms and rules of the Cube: d is a declaration, $=_{\mathtt{def}}$ is $=_\beta$)*

(axiom) $\quad <> \vdash * : \Box$

(start rule) $\quad \dfrac{\Gamma \prec d}{\Gamma d \vdash \mathtt{subj}(d) : \mathtt{pred}(d)}$

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

(application rule) $$\frac{\Gamma \vdash F : (A\Pi_x)B \qquad \Gamma \vdash a : A}{\Gamma \vdash (a\delta)F : B[x := a]}$$

(abstraction rule) $$\frac{\Gamma(A\lambda_x) \vdash b : B \qquad \Gamma \vdash (A\Pi_x)B : S}{\Gamma \vdash (A\lambda_x)b : (A\Pi_x)B}$$

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

(formation rule) $$\frac{\Gamma \vdash A : S_1 \qquad \Gamma(A\lambda_x) \vdash B : S_2}{\Gamma \vdash (A\Pi_x)B : S_2} \ \textit{if } (S_1, S_2) \textit{ is a rule}$$

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

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

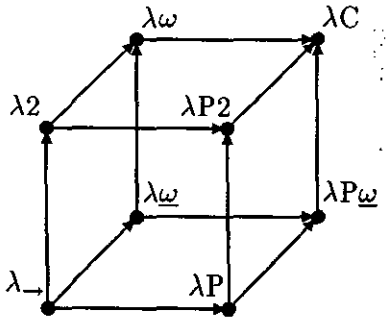

Figure 3: The Cube

Here are examples of typable terms in some systems of the Cube that we use further on.

Example 3.8

1. $\vdash_{\lambda 2} (*\Pi_\alpha)(\alpha\Pi_y)\alpha : *$ as we have the rule $(\Box,*)$, but $\not\vdash_{\mathcal{L}} (*\Pi_\alpha)(\alpha\Pi_y)\alpha : \tau$ for any τ where $\mathcal{L} \in \{\lambda_{\rightarrow}, \lambda\underline{\omega}, \lambda P, \lambda P\underline{\omega}\}$.
2. $(*\lambda_\sigma)(\sigma\lambda_t)((\sigma\Pi_q) * \lambda_Q)((t\delta)Q\lambda_N) \vdash_{\lambda P} (N\delta)(t\delta)(\sigma\lambda_x)((x\delta)Q\lambda_y)(y\delta)((x\delta)Q\lambda_Z)Z : (t\delta)Q$ but this derivation could not be obtained in $\lambda_{\rightarrow}$, $\lambda\underline{\omega}$ or $\lambda 2$ as we need the $(*,\Box)$ rule in order to derive that $(\sigma\Pi_q)* : \Box$ and hence that $((\sigma\Pi_q) * \lambda_Q)$ is allowed in the context.
3. If $\mathcal{L} \in \{\lambda_{\rightarrow}, \lambda\underline{\omega}\}$, then $(*\lambda_\beta)(\beta\lambda_{y'}) \not\vdash_{\mathcal{L}} (y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)x : \beta$ because the term

of part 1 of this example is not typable in $\mathcal{L}$ (note that with definitions, the last 9 steps above are replaced by a single one in Example 5.2). Here is how this judgement if derivable in $\lambda 2$.

$\vdash * : \Box$	(axiom)
$(*\lambda_\beta) \vdash_{\lambda 2} \beta : * : \Box$	(start resp. weakening rule)
$(*\lambda_\beta)(\beta\lambda_{y'}) \vdash_{\lambda 2} y' : \beta : * : \Box$	(start resp. weakening rule)
$(*\lambda_\beta)(\beta\lambda_{y'})(*\lambda_\alpha) \vdash_{\lambda 2} \alpha : *$	(start)
$(*\lambda_\beta)(\beta\lambda_{y'})(*\lambda_\alpha)(\alpha\lambda_y) \vdash_{\lambda 2} y : \alpha : *$	(start resp. weakening rule)
$(*\lambda_\beta)(\beta\lambda_{y'})(*\lambda_\alpha)(\alpha\lambda_y)(\alpha\lambda_x) \vdash_{\lambda 2} x : \alpha : *$	(start resp. weakening rule)
$(*\lambda_\beta)(\beta\lambda_{y'})(*\lambda_\alpha)(\alpha\lambda_y) \vdash_{\lambda 2} (\alpha\Pi_x)\alpha : *$	formation rule $(*,*)$)
$(*\lambda_\beta)(\beta\lambda_{y'})(*\lambda_\alpha)(\alpha\lambda_y) \vdash_{\lambda 2} (\alpha\lambda_x)x : (\alpha\Pi_x)\alpha : *$	(abstraction)
$(*\lambda_\beta)(\beta\lambda_{y'})(*\lambda_\alpha)(\alpha\lambda_y) \vdash_{\lambda 2} (y\delta)(\alpha\lambda_x)x : \alpha$	(application)
$(*\lambda_\beta)(\beta\lambda_{y'})(*\lambda_\alpha) \vdash_{\lambda 2} (\alpha\Pi_y)\alpha : *$	formation rule $(*,*)$)
$(*\lambda_\beta)(\beta\lambda_{y'})(*\lambda_\alpha) \vdash_{\lambda 2} (\alpha\lambda_y)(y\delta)(\alpha\lambda_x)x : (\alpha\Pi_y)\alpha : *$	(abstraction rule)
$(*\lambda_\beta)(\beta\lambda_{y'}) \vdash_{\lambda 2} (*\Pi_\alpha)(\alpha\Pi_y)\alpha : *$	formation rule $(\Box,*)$)
$(*\lambda_\beta)(\beta\lambda_{y'}) \vdash_{\lambda 2} (*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)x : (*\Pi_\alpha)(\alpha\Pi_y)\alpha$	(abstraction rule)
$(*\lambda_\beta)(\beta\lambda_{y'}) \vdash_{\lambda 2} (\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)x : (\beta\Pi_y)\beta$	(application rule)
$(*\lambda_\beta)(\beta\lambda_{y'}) \vdash_{\lambda 2} (y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)x : \beta$	(application rule)

3.2 Properties of the ordinary typing relation

Here we list the properties of the Cube (see [Barendregt 92]). These properties will be established for the Cube extended with generalised reduction and definition mechanisms.

Theorem 3.9 *(The Church Rosser Theorem for $\twoheadrightarrow_\beta$)*
If $A \twoheadrightarrow_\beta B$ and $A \twoheadrightarrow_\beta C$ then there exists D such that $B \twoheadrightarrow_\beta D$ and $C \twoheadrightarrow_\beta D$ □

Lemma 3.10 *(Free variable lemma for $\vdash$)*
Let Γ be a legal context such that $\Gamma \vdash B : C$. Then the following holds:

1. *If d and d' are two different elements of Γ-*`decl`*, then* `subj`$(d) \not\equiv$ `subj`(d').
2. *$FV(B), FV(C) \subseteq dom(\Gamma)$.*
3. *For s_1 a main item of Γ, $FV(s_1) \subseteq \{$`subj`$(d) \mid d \in \Gamma$-`decl`$, d$ is to the left of s_1 in $\Gamma\}$.*

Proof: *All by induction on the derivation of $\Gamma \vdash B : C$.* □

Lemma 3.11 *(Start Lemma for $\vdash$)*
Let Γ be a legal context. Then $\Gamma \vdash * : \Box$ and $\forall d \in' \Gamma[\Gamma \vdash d]$.
Proof: *As Γ is legal, then $\exists A, B \in \mathcal{T}$ such that $\Gamma \vdash A : B$. Now use induction on the derivation $\Gamma \vdash A : B$.* □

Note that this and the following lemmas state the same thing as the corresponding lemmas from [Barendregt 92] as Γ consists of declarations only.

Lemma 3.12 *(Transitivity Lemma for $\vdash$)*
Let Γ and Δ be legal contexts. Then: $[\Gamma \vdash \Delta \wedge \Delta \vdash A : B] \Rightarrow \Gamma \vdash A : B$.
Proof: *Induction on the derivation rules.* □

Lemma 3.13 *(Substitution Lemma for $\vdash$)*
Assume $\Gamma(A\lambda_x)\Delta \vdash B : C$ *and* $\Gamma \vdash D : A$ *then* $\Gamma(\Delta[x := D]) \vdash B[x := D] : C[x := D]$.
Proof: *By induction on the derivation rules.* □

Lemma 3.14 *(Thinning Lemma for $\vdash$)*
Let Γ *and* Δ *be legal contexts such that* $\Gamma \subseteq' \Delta$. *Then* $\Gamma \vdash A : B \Rightarrow \Delta \vdash A : B$
Proof: *By induction on the length of the derivation of* $\Gamma \vdash A : B$. □

Lemma 3.15 *(Generation Lemma for $\vdash$)*

1. $\Gamma \vdash x : C \Rightarrow \exists S_1, S_2 \in \mathcal{S} \exists B =_\beta C[\Gamma \vdash B : S_1 \wedge (B\lambda_x) \in' \Gamma \wedge \Gamma \vdash C : S_2]$.
2. $\Gamma \vdash (A\Pi_x)B : C \Rightarrow \exists (S_1, S_2 \in \mathcal{S})[\Gamma \vdash A : S_1 \wedge \Gamma(A\lambda_x) \vdash B : S_2 \wedge (S_1, S_2)$ *is a rule* $\wedge C =_\beta S_2 \wedge [C \not\equiv S_2 \Rightarrow \exists S[\Gamma \vdash C : S]]]$
3. $\Gamma \vdash (A\lambda_x)b : C \Rightarrow \exists (S, B)[\Gamma \vdash (A\Pi_x)B : S \wedge \Gamma(A\lambda_x) \vdash b : B \wedge C =_\beta (A\Pi_x)B \wedge C \not\equiv (A\Pi_x)B \Rightarrow \exists S \in \mathcal{S}[\Gamma \vdash C : S]]$.
4. $\Gamma \vdash (a\delta)F : C \Rightarrow \exists A, B, x[\Gamma \vdash F : (A\Pi_x)B \wedge \Gamma \vdash a : A \wedge C =_\beta B[x := a] \wedge (B[x := a] \not\equiv C \Rightarrow \exists S \in \mathcal{S}[\Gamma \vdash C : S])]$.

Proof: *By induction on the derivation rules, using thinning lemma.* □

Corollary 3.16 *(Generation Corollary for $\vdash$)*

1. $\Gamma \vdash A : B \Rightarrow \exists S[B \equiv S$ *or* $\Gamma \vdash B : S]$
2. $\Gamma \vdash A : (B_1\Pi_x)B_2 \Rightarrow \exists S[\Gamma \vdash (B_1\Pi_x)B_2 : S]$
3. *If* A *is a* Γ*-term, then* A *is* $\Box$, *a* Γ*-kind or a* Γ*-element.*
4. *If* A *is legal and* B *is a subexpression of* A *then* B *is legal.* □

Theorem 3.17 *(Subject Reduction for $\vdash$ and $\twoheadrightarrow_\beta$)*
$\Gamma \vdash A : B \wedge A \twoheadrightarrow_\beta A' \Rightarrow \Gamma \vdash A' : B$
Proof: $\Gamma \vdash A : B \wedge A \twoheadrightarrow_\beta A' \Rightarrow \Gamma \vdash A' : B$ *and* $\Gamma \vdash A : B \wedge \Gamma \twoheadrightarrow_\beta \Gamma' \Rightarrow \Gamma' \vdash A : B$, *where* $\Gamma \twoheadrightarrow_\beta \Gamma'$ *means* $\Gamma \equiv \Gamma_1(A\lambda_x)\Gamma_2$, $\Gamma' \equiv \Gamma_1(A'\lambda_x)\Gamma_2$ *and* $A \twoheadrightarrow_\beta A'$, *are proved simultaneously by induction on the derivation rules.* □

Corollary 3.18 *(SR Corollary for $\vdash$ and $\twoheadrightarrow_\beta$)*

1. *If* $\Gamma \vdash A : B$ *and* $B \twoheadrightarrow_\beta B'$ *then* $\Gamma \vdash A : B'$.
2. *If* A *is a* Γ*-term and* $A \twoheadrightarrow_\beta A'$ *then* A' *is a* Γ*-term.* □

Lemma 3.19 *(Unicity of Types for $\vdash$ and $\twoheadrightarrow_\beta$)*

1. $\Gamma \vdash A : B_1 \wedge \Gamma \vdash A : B_2 \Rightarrow B_1 =_\beta B_2$
2. $\Gamma \vdash A : B \wedge \Gamma \vdash A' : B' \wedge A =_\beta A' \Rightarrow B =_\beta B'$
3. $\Gamma \vdash B : S, B =_\beta B', \Gamma \vdash A' : B'$ *then* $\Gamma \vdash B' : S$.

Proof: *1. by induction on the structure of* A, *2. by Church Rosser, Subject Reduction and 1, and 3. by Corollary 3.16, Subject Reduction and 1.* □

Theorem 3.20 *(Strong Normalisation with respect to $\vdash$ and $\twoheadrightarrow_\beta$)*
For all $\vdash$*-legal terms* M, M *is strongly normalising with respect to* $\twoheadrightarrow_\beta$.
Proof: *see section 6.* □

4 Generalising reduction in the Cube

In this section we extend the classical notions of redexes and β-reduction of the Cube and show that all the properties of Section 3.2 except SR are preserved. We show moreover that for $\lambda\underline{\omega}$ and $\lambda{\to}$, SR holds yet for the remaining systems it fails.

4.1 The generalised reduction

We allow *$\delta\lambda$-couples* to have the same "reduction rights" as *$\delta\lambda$-segments* as follows:

Definition 4.1 *(General β-reduction $\hookrightarrow_\beta$ for the Cube)*
General one-step β-reduction $\hookrightarrow_\beta$, is the least compatible relation generated out of:

$$(\textit{general } \beta) \qquad (B\delta)\overline{s}(C\lambda_x)A \hookrightarrow_\beta \overline{s}(A[x := B]) \qquad \textit{if } \overline{s} \textit{ is well-balanced}$$

General $\hookrightarrow\!\!\!\!\rightarrow_\beta$ is the reflexive and transitive closure of $\hookrightarrow_\beta$ and $\approx_\beta$ is the least equivalence relation generated by $\hookrightarrow\!\!\!\!\rightarrow_\beta$.

Example 4.2 Take Example 1.2. As $(c\delta)(P\lambda_x)(b\delta)(Q\lambda_y)$ is a well-balanced segment, then:

$$\begin{array}{ll} A \equiv (d\delta)(c\delta)(P\lambda_x)(b\delta)(Q\lambda_y)(R\lambda_z)(a\delta)z & \hookrightarrow_\beta \\ (c\delta)(P\lambda_x)(b\delta)(Q\lambda_y)(((a\delta)z)[z := d]) & \equiv \\ (c\delta)(P\lambda_x)(b\delta)(Q\lambda_y)(a\delta)d & \end{array}$$

$(d\delta)(R\lambda_z)$ also has a corresponding ("generalised") redex in the traditional notation, which will appear after two one-step β-reductions, leading to $(\lambda_{z:R}.za)d$. With $\hookrightarrow_\beta$, we could reduce $((\lambda_{x:P}.(\lambda_{y:Q}.\lambda_{z:R}.za)b)c)d$ to $(\lambda_{x:P}.(\lambda_{y:Q}.da)b)c$. This is difficult to carry out in the classical λ-calculus. We strongly believe that it is the item notation which enables us to extend reduction smoothly beyond $\twoheadrightarrow_\beta$. Moreover, $\hookrightarrow_\beta$ extends $\to_\beta$.

Lemma 4.3 *If $A \to_\beta B$ in the sense of Definition 2.4, then $A \hookrightarrow_\beta B$ in the sense of Definition 4.1. Moreover, if $A \hookrightarrow_\beta B$ comes from contracting a $\delta\lambda$-segment then $A \to_\beta B$.*
Proof: *Obvious as a $\delta\lambda$-segment is a definition.* □

Lemma 4.4 *If $A \hookrightarrow_\beta B$ then $A =_\beta B$.*
Proof: *It suffices to consider the case $A \equiv \overline{s_1}(C\delta)\overline{s}(D\lambda_x)E$ where the contracted redex is based on $(C\delta)(D\lambda_x)$, $B \equiv \overline{s_1}\,\overline{s}(E[x := C])$, and $\overline{s}$ is well-balanced (hence $\mathtt{weight}(\overline{s})$ is even). We prove the lemma by induction on $\mathtt{weight}(\overline{s})$. Case $\mathtt{weight}(\overline{s}) = 0$ then obvious as $\hookrightarrow_\beta$ coincides with $\to_\beta$ in this case. Assume the property holds when $\mathtt{weight}(\overline{s}) = 2n$. Take $\overline{s}$ such that $\mathtt{weight}(\overline{s}) = 2n + 2$. Now, $\overline{s} \equiv (C'\delta)\overline{s'}(D'\lambda_y)\overline{s''}$ where $\overline{s'}$, $\overline{s''}$ are well-balanced. Assume $x \not\equiv y$ (if necessary, use renaming).*

- *From $\overline{s}(E[x := C]) \hookrightarrow_\beta \overline{s'}(\overline{s''}(E[x := C])[y := C'])$, IH and compatibility, $B =_\beta \overline{s_1}\,\overline{s'}(\overline{s''}(E[x := C])[y := C']) \equiv \overline{s_1}\,\overline{s'}(\overline{s''}[y := C'])(E[x := C][y := C']) \equiv B''$.*
- *Moreover, $A \equiv \overline{s_1}(C\delta)(C'\delta)\overline{s'}(D'\lambda_y)\overline{s''}(D\lambda_x)E \hookrightarrow_\beta \overline{s_1}(C\delta)\overline{s'}(\overline{s''}(D\lambda_x)E[y := C']) \equiv^{BC} \overline{s_1}(C\delta)\overline{s'}(\overline{s''}[y := C'])(D[y := C']\lambda_x)(E[y := C']) \equiv B'$. So by IH $A =_\beta B'$.*
- *$B' \hookrightarrow_\beta \overline{s_1}\overline{s'}(\overline{s''}[y := C'])(E[y := C'][x := C])$, $x, y \notin FV(C) \cup FV(C')$ (by BC). Hence, by IH and substitution $B' =_\beta \overline{s_1}\overline{s'}(\overline{s''}[y := C'])(E[x := C][y := C']) \equiv B''$.*

Therefore, $A =_\beta B', B' =_\beta B''$ and $B =_\beta B''$, hence $A =_\beta B$. □

The following shows that conversion does not change the typing relation of Section 3.1.

Corollary 4.5
If $A \hookrightarrow\!\!\!\!\rightarrow_\beta B$ then $A =_\beta B$. Moreover, $A \approx_\beta B$ iff $A =_\beta B$. □

4.2 Properties of ordinary typing with generalised reduction

Because $=_\beta$ and $\approx_\beta$ are equivalent, the only lemmas/theorems of Section 3.2 affected by our extension of reductions are those which have $\twoheadrightarrow_\beta$ in their heading. These are CR (Theorem 3.9), SR (Theorem 3.17) and its Corollary 3.18, Unicity of Types (Lemma 3.19) and SN (Theorem 3.20). In this section, we show that CR and SN hold for the Cube with $\hookrightarrow\!\!\!\!\rightarrow_\beta$ and that SR holds for $\lambda\underline{\omega}$ and $\lambda{\rightarrow}$ but fails for the other six systems. Unicity of typing depends on SR and on the fact that $=_\beta$ is the same as $\hookrightarrow\!\!\!\!\rightarrow_\beta$. Hence, we ignore it here as once we prove SR, its proof will be exactly that of Lemma 3.19.

Theorem 4.6 *(The general Church Rosser theorem for $\hookrightarrow\!\!\!\!\rightarrow_\beta$)*
If $A \hookrightarrow\!\!\!\!\rightarrow_\beta B$ and $A \hookrightarrow\!\!\!\!\rightarrow_\beta C$, then there exists D such that $B \hookrightarrow\!\!\!\!\rightarrow_\beta D$ and $C \hookrightarrow\!\!\!\!\rightarrow_\beta D$.
Proof: *As $A \hookrightarrow\!\!\!\!\rightarrow_\beta B$ and $A \hookrightarrow\!\!\!\!\rightarrow_\beta C$ then by Corollary 4.5, $A =_\beta B$ and $A =_\beta C$. Hence, $B =_\beta C$ and by CR for $\twoheadrightarrow_\beta$, there exists D such that $B \twoheadrightarrow_\beta D$ and $C \twoheadrightarrow_\beta D$. But, $A \twoheadrightarrow_\beta B$ implies $A \hookrightarrow\!\!\!\!\rightarrow_\beta B$. Hence CR holds for $\hookrightarrow\!\!\!\!\rightarrow_\beta$.* □

Theorem 4.7 *(Strong Normalisation with respect to $\vdash$ and $\hookrightarrow\!\!\!\!\rightarrow_\beta$)*
For all $\vdash$-legal terms M, M is strongly normalising with respect to $\hookrightarrow\!\!\!\!\rightarrow_\beta$.
Proof: *This is a special case of the proof of Theorem 6.27.* □

In the following, $\mathcal{L}$ ranges over $\lambda\underline{\omega}$ and $\lambda{\rightarrow}$. Here we show that SR holds for $\mathcal{L}$.

Lemma 4.8 *If $\Gamma \vdash_{\mathcal{L}} A : \Box$ then $A \in \{*, (*\Pi_x)*, (*\Pi_x)(*\Pi_y)*, ((*\Pi_x) * \Pi_y)*, \ldots\}$.*
Proof: *By induction on the derivation rules.* □

Lemma 4.9 *If B is a legal $\mathcal{L}$-term, B' is a $\mathcal{L}$-kind and $B =_\beta B'$ then B is a kind.*
Proof: *First show by induction on the derivations: If $*$ is a subterm of A and A is legal then A is a kind or $*$ is type-information in A (as in $(*\lambda_x)y$). Now, as B' is a kind, B' is in normal form, hence $B \twoheadrightarrow_\beta B'$ and by the former result B must be a kind too.* □

Lemma 4.10 *If $\Gamma \vdash_{\mathcal{L}} (A\Pi_x)B : S$, then $\Gamma \vdash_{\mathcal{L}} A : S$, $\Gamma(A\lambda_x) \vdash_{\mathcal{L}} B : S$ and $x \notin FV(B)$.*
Proof: *Show by induction on the derivation of $\Gamma \vdash_{\mathcal{L}} A : B$ that if B a kind, then for all $(C\lambda_{x^*}) \in \Gamma$-`decl`, $x^* \notin FV(A)$. We only treat two cases:*

- *application rule: $\Gamma \vdash_{\mathcal{L}} (a\delta)F : B[x := a]$ out of $\Gamma \vdash_{\mathcal{L}} F : (A\Pi_x)B$ and $\Gamma \vdash_{\mathcal{L}} a : A$. Suppose $B[x := a]$ is a kind and $(C\lambda_y) \in' \Gamma$, $\Gamma \vdash_{\mathcal{L}} C : *$. If $x \notin FV(B)$ then B is a kind, so A and $(A\Pi_x)B$ are kinds too, hence $y \notin FV(a), FV(F)$ by the IH.*

  If $x \in FV(B)$ then a is a kind (as $B[x := a]$ is a kind) and hence $A \equiv \Box$ which is impossible as $\Gamma \vdash_{\mathcal{L}} F : (A\Pi_x)B$.

- *conversion rule: $\Gamma \vdash_{\mathcal{L}} A : B'$ out of $\Gamma \vdash_{\mathcal{L}} A : B$, $\Gamma \vdash_{\mathcal{L}} B' : S$, $B =_\beta B'$. Suppose B' is a kind, then by lemma 4.9: B is a kind, hence by induction hypothesis we are done.* □

Lemma 4.11

1. $\Gamma \vdash (A\delta)B : C \Rightarrow \Gamma \vdash C : S$ *for some sort* S.
2. *If* $\Gamma \vdash_{\mathcal{L}} A : S_1, \Gamma \vdash_{\mathcal{L}} B : S_2$ *and* $A =_\beta B$ *then* $S_1 \equiv S_2$.

Proof:

1. *Generation gives* $\Gamma \vdash A : D$, $\Gamma \vdash B : (D\Pi_x)E$, $E[x := A] =_\beta C$ *and* $E[x := A] \not\equiv C \Rightarrow \Gamma \vdash C : S$. *So suppose* $E[x := A] \equiv C$, *then* $\Gamma \vdash B : (D\Pi_x)E$ *implies by Corollary 3.16 that* $\Gamma \vdash (D\Pi_x)E : S$. *Hence, by generation* $\Gamma \vdash D : S_1$, $\Gamma(D\lambda_x) \vdash E : S_2$. *Now, use* $\Gamma \vdash A : D$ *and substitution to get* $\Gamma \vdash E[x := A] : S_2$.
2. *Note that* $S_1 \equiv \Box$ *or* $S_2 \equiv \Box$, *hence by Lemma 4.9,* $S_1 \equiv S_2$. □

The crucial step in the proof of Subject Reduction in $\lambda\underline{\omega}$ and $\lambda_{\rightarrow}$ is proved in the following:

Lemma 4.12 *(Shuffle Lemma for* $\lambda\underline{\omega}$ *and* $\lambda_{\rightarrow}$*)*
$\Gamma \vdash_{\mathcal{L}} \overline{s}_1(A\delta)\overline{s}_2 B : C \iff \Gamma \vdash_{\mathcal{L}} \overline{s}_1\overline{s}_2(A\delta)B : C$ *where* $\overline{s}_2$ *is well-balanced and the binding variables in* $\overline{s}_2$ *are not free in* A.

Proof: *By induction on* $\mathtt{weight}(\overline{s}_2)$. *Case* $\mathtt{weight}(\overline{s}_2) = 0$ *then nothing to prove. Case* $\mathtt{weight}(\overline{s}_2) = 2$, *say* $\overline{s}_2 \equiv (D\delta)(E\lambda_x)$, *use induction on* $\mathtt{weight}(\overline{s}_1)$. *Suppose first,* $\mathtt{weight}(\overline{s}_1) = 0$.

$\Rightarrow$) *suppose* $\Gamma \vdash_{\mathcal{L}} (A\delta)(D\delta)(E\lambda_x)B : C$. *Using generation three times, we obtain:*

$$\Gamma \vdash_{\mathcal{L}} A : F \tag{1}$$
$$\Gamma \vdash_{\mathcal{L}} (D\delta)(E\lambda_x)B : (F\Pi_y)G \tag{2}$$
$$G \equiv G[y := A] =_\beta C \quad (\textit{Lemma } 4.10, \textit{Corollary } 3.16) \tag{3}$$
$$\Gamma \vdash_{\mathcal{L}} D : H \tag{4}$$
$$\Gamma \vdash_{\mathcal{L}} (E\lambda_x)B : (H\Pi_z)I$$
$$I \equiv I[z := D] =_\beta (F\Pi_y)G \quad (\textit{Lemma } 4.10, \textit{Corollary } 3.16) \tag{5}$$
$$\Gamma \vdash_{\mathcal{L}} (E\Pi_x)J : S_1$$
$$\Gamma(E\lambda_x) \vdash_{\mathcal{L}} B : J \tag{6}$$
$$(H\Pi_z)I =_\beta (E\Pi_x)J \tag{7}$$

Out of (7) and Lemma 4.10 we see that $x \equiv z$, $H =_\beta E$, $I =_\beta J$, $y \notin FV(G), x \notin FV(I) \cup FV(J)$, $\Gamma \vdash_{\mathcal{L}} F, G, H, I, E : S_1$ *(8)*

and out of (7) and (5): $J =_\beta (F\Pi_y)G$. *Hence (9)*

$$\Gamma(E\lambda_x) \vdash_{\mathcal{L}} B : (F\Pi_y)G \quad \textit{(conversion, (6), (9), (8) implies by the generation and thinning lemmas: } \Gamma(E\lambda_x) \vdash_{\mathcal{L}} (F\Pi_y)G : S_1\textit{)} \tag{10}$$
$$\Gamma(E\lambda_x) \vdash_{\mathcal{L}} A : F \quad \textit{(thinning lemma, (1))} \tag{11}$$
$$\Gamma(E\lambda_x) \vdash_{\mathcal{L}} (A\delta)B : G \quad \textit{((10), (11), application, } G[y := A] \equiv G\textit{)} \tag{12}$$
$$\Gamma \vdash_{\mathcal{L}} (H\Pi_x)G, (E\Pi_x)G : S_1 \quad \textit{(formation, thinning, } \Gamma \vdash_{\mathcal{L}} H, G, E : S_1\textit{)} \tag{13}$$
$$\Gamma \vdash_{\mathcal{L}} (E\lambda_x)(A\delta)B : (H\Pi_x)G \quad \textit{((12), (13), abstraction, conversion, } (8) \Rightarrow (E\Pi_x)G =_\beta (H\Pi_x)G\textit{)} \tag{14}$$

$$\Gamma \vdash_{\mathcal{L}} (D\delta)(E\lambda_x)(A\delta)B : G \quad \textit{((14), application, (4), } G[x := D] \equiv G\textit{)} \tag{15}$$

$$\Gamma \vdash_{\mathcal{L}} C : S \quad \textit{(Lemma 4.11, hypothesis)} \tag{16}$$

$$\Gamma \vdash_{\mathcal{L}} (D\delta)(E\lambda_x)(A\delta)B : C \quad \textit{(conversion, (15), (16), (3))} \tag{17}$$

$\Leftarrow$) *Suppose* $\Gamma \vdash_{\mathcal{L}} (D\delta)(E\lambda_x)(A\delta)B : C$

Then $\Gamma \vdash_{\mathcal{L}} C : S_1$ *(Lemma 4.11)* (18)

and by generation three times we get:

$$\Gamma \vdash_{\mathcal{L}} D : F \tag{19}$$

$$\Gamma \vdash_{\mathcal{L}} (E\lambda_x)(A\delta)B : (F\Pi_y)G$$

$$G \equiv G[y := D] =_\beta C \quad (\textit{Lemma } 4.10, \textit{Corollary } 3.16) \tag{20}$$

$$\Gamma \vdash_{\mathcal{L}} (E\Pi_x)H : S_2 \tag{21}$$

$$\Gamma(E\lambda_x) \vdash_{\mathcal{L}} (A\delta)B : H$$

$$(E\Pi_x)H =_\beta (F\Pi_y)G \tag{22}$$

$$\Gamma(E\lambda_x) \vdash_{\mathcal{L}} A : I \tag{23}$$

$$\Gamma(E\lambda_x) \vdash_{\mathcal{L}} B : (I\Pi_z)J \tag{24}$$

$$J \equiv J[z := A] =_\beta H \quad (\textit{Lemma } 4.10, \textit{Corollary } 3.16) \tag{25}$$

Now (24) and Corollary 3.16 imply that for some S_3, $\Gamma(E\lambda_x) \vdash_{\mathcal{L}} (I\Pi_z)J : S_3$. *Hence, by Lemma 4.10,* $z \notin FV(J), \Gamma(E\lambda_x) \vdash_{\mathcal{L}} J : S_3$. *Also, by Lemma 4.10, we get out of (21) that* $\Gamma \vdash_{\mathcal{L}} E : S_2, \Gamma(E\lambda_x) \vdash_{\mathcal{L}} H : S_2$ *and* $x \notin FV(H)$. *Now,* $J =_\beta H$ *from (25), hence* $x \notin FV(J)$. *Moreover, by Lemma 4.11, we see* $S_2 \equiv S_3$. *Hence,*

$$\Gamma \vdash_{\mathcal{L}} (E\Pi_x)(I\Pi_z)J : S_2 \quad \textit{formation} \tag{26}$$

$$\Gamma \vdash_{\mathcal{L}} (E\lambda_x)B : (E\Pi_x)(I\Pi_z)J \quad \textit{((26), (24), abstraction)} \tag{27}$$

$$\Gamma \vdash_{\mathcal{L}} (D\delta)(E\lambda_x)B : (I\Pi_z)J \quad \textit{(application, (27), } x \notin FV(I,J)\textit{,}$$

$$\Gamma \vdash_{\mathcal{L}} D : E \textit{ because (22) implies } E =_\beta F \tag{28}$$

and we use conversion, (19), $\Gamma \vdash_{\mathcal{L}} E : S_2$*)*

$$\Gamma \vdash_{\mathcal{L}} (A\delta)(D\delta)(E\lambda_x)B : J \quad \textit{(out of } \Gamma(E\lambda_x) \vdash_{\mathcal{L}} A : I \textit{ and } \Gamma \vdash D : E \tag{29}$$

we find by substitution $(x \notin FV(A,I))$*,*

$\Gamma \vdash_{\mathcal{L}} A : I$*. Now, use application)*

$$\Gamma \vdash_{\mathcal{L}} (A\delta)(D\delta)(E\lambda_x)B : C \quad \textit{((29), (conversion; } C =_\beta J$$

follows from (25),

(22) and (20))

Now suppose $\mathtt{weight}(\overline{s}_1) = n+1$. *Using the generation lemma we obtain* $\Gamma' \vdash_{\mathcal{L}} \overline{s}'_1(A\delta)\overline{s}_2 B : C'$, *where* $\mathtt{weight}(\overline{s}'_1) = n$, *hence the induction hypothesis says* $\Gamma' \vdash_{\mathcal{L}} \overline{s}'_1\overline{s}_2(A\delta)B : C'$ *and by applying the appropriate derivation rule we obtain* $\Gamma \vdash_{\mathcal{L}} \overline{s}_1\overline{s}_2(A\delta)B : C$.
Case $\mathtt{weight}(\overline{s}_2) = 2(n+1), n \geq 1$, *then* $\overline{s}_2 \equiv (D\delta)\overline{s}_3(E\lambda_x)\overline{s}_4$ *for some* C, D, x *and well-balanced segments* $\overline{s}_3, \overline{s}_4$. *Then,* $\mathtt{weight}(\overline{s}_3), \mathtt{weight}(\overline{s}_4) \leq 2n$ *and we see:*

$$\Gamma \vdash_{\mathcal{L}} \overline{s}_1(A\delta)(D\delta)\overline{s}_3(E\lambda_x)\overline{s}_4 B : C \overset{I.H.}{\Longleftrightarrow} \Gamma \vdash_{\mathcal{L}} \overline{s}_1(A\delta)\overline{s}_3(D\delta)(E\lambda_x)\overline{s}_4 B : C \overset{I.H.}{\Longleftrightarrow}$$

$$\Gamma \vdash_{\mathcal{L}} \overline{s}_1\overline{s}_3(A\delta)(D\delta)(E\lambda_x)\overline{s}_4 B : C \overset{I.H.}{\Longleftrightarrow} \Gamma \vdash_{\mathcal{L}} \overline{s}_1\overline{s}_3(D\delta)(E\lambda_x)(A\delta)\overline{s}_4 B : C \overset{I.H.}{\Longleftrightarrow}$$

$$\Gamma \vdash_{\mathcal{L}} \overline{s}_1(D\delta)\overline{s}_3(E\lambda_x)(A\delta)\overline{s}_4 B : C \overset{I.H.}{\Longleftrightarrow} \Gamma \vdash_{\mathcal{L}} \overline{s}_1(D\delta)\overline{s}_3(E\lambda_x)\overline{s}_4(A\delta)B : C \qquad \square$$

Theorem 4.13 (Generalised Subject Reduction for $\lambda\underline{\omega}$ and $\lambda_{\rightarrow}$ for $\vdash$ and $\hookrightarrow\!\!\!\!\rightarrow_\beta$)
If $\Gamma \vdash_{\mathcal{L}} A : B$ and $A \hookrightarrow_\beta A'$ then $\Gamma \vdash_{\mathcal{L}} A' : B$.

Proof: *We prove by simultaneous induction on the generation of $\Gamma \vdash_{\mathcal{L}} A : B$ that*

$$\Gamma \vdash_{\mathcal{L}} A : B \wedge A \hookrightarrow_\beta A' \;\Rightarrow\; \Gamma \vdash_{\mathcal{L}} A' : B \qquad (i)$$
$$\Gamma \vdash_{\mathcal{L}} A : B \wedge \Gamma \hookrightarrow_\beta \Gamma' \;\Rightarrow\; \Gamma' \vdash_{\mathcal{L}} A : B \qquad (ii)$$

where $\Gamma \hookrightarrow_\beta \Gamma'$ means $\Gamma \equiv \Gamma_1(A\lambda_x)\Gamma_2, \Gamma' \equiv \Gamma_1(A'\lambda_x)\Gamma_2$ and $A \hookrightarrow_\beta A'$ for some $\Gamma_1, \Gamma_2, A, A', x$. The cases in which the last rule applied is axiom, start, weakening or conversion are easy (for start: use conversion). We treat the three other cases.
Formation: $\Gamma \vdash_{\mathcal{L}} (A_1\Pi_x)B_1 : S_1$ is a direct consequence of $\Gamma \vdash_{\mathcal{L}} A_1 : S_1$ and $\Gamma(A_1\lambda_x) \vdash_{\mathcal{L}} B_1 : S_1$, then (i) comes from IH(i) and IH(ii); (ii) comes from IH(ii). Abstraction: similar to formation. Application: $\Gamma \vdash_{\mathcal{L}} (a\delta)F : B_1[x := a]$ is a direct consequence of $\Gamma \vdash_{\mathcal{L}} F : (A_1\Pi_x)B_1$ and $\Gamma \vdash_{\mathcal{L}} a : A_1$. Now (ii) comes from IH(ii). We consider various cases:

- *Subcase 1: $(a\delta)F \hookrightarrow_\beta (a\delta)F'$ because $F \hookrightarrow_\beta F'$. Then (i) follows from IH(i).*
- *Subcase 2: $(a\delta)F \hookrightarrow_\beta (a'\delta)F$ because $a \hookrightarrow_\beta a'$. From IH$(i)$ and application, $\Gamma \vdash (a'\delta)F : B_1[x := a']$. Also, from Corollary 3.16, for some S_1: $\Gamma \vdash_{\mathcal{L}} (A_1\Pi_x)B_1 : S_1$ and hence by generation: $\Gamma(A\lambda_x) \vdash_{\mathcal{L}} B_1 : S_1$ and thus by substitution $\Gamma \vdash_{\mathcal{L}} B_1[x := a] : S_1$. Now conversion gives $\Gamma \vdash_{\mathcal{L}} (a'\delta)F : B_1[x := a]$ which proves (i).*
- *Subcase 3: $F \equiv \overline{s}(A'\lambda_y)F'$, $\overline{s}$ well-balanced and $(a\delta)F \hookrightarrow_\beta \overline{s}F'[y := a]$. Now, by Lemma 4.12, $\Gamma \vdash_{\mathcal{L}} \overline{s}(a\delta)(A'\lambda_y)F' : B_1[x := a]$ and $\overline{s}(a\delta)(A'\lambda_y)F' \rightarrow_\beta \overline{s}F'[y := a]$ so by SR for $\rightarrow\!\!\!\!\rightarrow_\beta$, $\Gamma \vdash_{\mathcal{L}} \overline{s}F'[y := a] : B_1[x := a]$ which proves (i).* □

SR however is not valid for the other systems of the Cube as the following examples show:

Example 4.14 *(SR does not hold in $\lambda 2$ using $\hookrightarrow\!\!\!\!\rightarrow_\beta$)*
$(*\lambda_\beta)(\beta\lambda_{y'}) \vdash_{\lambda 2} (\forall_{\mathcal{L}} \text{ for } \mathcal{L} \in \{\lambda_{\rightarrow}, \lambda\underline{\omega}\})\ (y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)x : \beta$ (see Example 3.8).
Moreover, $(y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)x \hookrightarrow_\beta (\beta\delta)(*\lambda_\alpha)(y'\delta)(\alpha\lambda_x)x$.
Yet, $(*\lambda_\beta)(\beta\lambda_{y'}) \not\vdash_{\lambda 2} (\beta\delta)(*\lambda_\alpha)(y'\delta)(\alpha\lambda_x)x : \beta$.
Even, $(*\lambda_\beta)(\beta\lambda_{y'}) \not\vdash_{\lambda 2} (\beta\delta)(*\lambda_\alpha)(y'\delta)(\alpha\lambda_x)x : \tau$ for any τ.
This is because $(\alpha\lambda_x)x : (\alpha\Pi_x)\alpha$ and $y : \beta$ yet α and β are unrelated and hence we fail in firing the application rule to find the type of $(y'\delta)(\alpha\lambda_x)x$. Looking closer however, one finds that $(\beta\delta)(*\lambda_\alpha)$ is defining α to be β, yet no such information can be used to combine $(\alpha\Pi_x)\alpha$ with β. We will redefine the rules of the Cube so that such information can be taken into account. Finally note that failure of SR in $\lambda 2$, means its failure in $\lambda P2, \lambda\omega$ and λC.

Example 4.15 *(SR does not hold in λP using $\hookrightarrow\!\!\!\!\rightarrow_\beta$)*
$(*\lambda_\sigma)(\sigma\lambda_t)((\sigma\Pi_q)*\lambda_Q)((t\delta)Q\lambda_N) \vdash_{\lambda P} (N\delta)(t\delta)(\sigma\lambda_x)((x\delta)Q\lambda_y)(y\delta)((x\delta)Q\lambda_Z)Z : (t\delta)Q$. Note here that this cannot be derived in $\lambda_{\rightarrow}$, $\lambda 2$ or $\lambda\underline{\omega}$ (see Example 3.8).
And $(N\delta)(t\delta)(\sigma\lambda_x)((x\delta)Q\lambda_y)(y\delta)((x\delta)Q\lambda_Z)Z \hookrightarrow_\beta (t\delta)(\sigma\lambda_x)(N\delta)((x\delta)Q\lambda_Z)Z$
Now, $N : (t\delta)Q, t : \sigma, y : (x\delta)Q, x : \sigma, (t\delta)Q \neq (x\delta)Q$.
$(*\lambda_\sigma)(\sigma\lambda_t)((\sigma\Pi_q) * \lambda_Q)((t\delta)Q\lambda_N) \not\vdash_{\lambda P} (t\delta)(\sigma\lambda_x)(N\delta)((x\delta)Q\lambda_Z)Z : \tau$ for any τ.
Here again the reason of failure is similar to the above example. At one stage, we need to match $(x\delta)Q$ with $(t\delta)Q$ but this is not possible even though we do have the definition segment: $(t\delta)(\sigma\lambda_x)$ which defines x to be t. All this calls for the need to use these definitions. Finally note that failure of SR in λP, means its failure in $\lambda P2, \lambda P\underline{\omega}$ and λC.

5 Extending the Cube with definition mechanisms

As a first step in the direction of including extended reduction in the systems of the Cube, we now investigate adding definitions to the Cube. We shall extend the derivation rules so that we can use definitions in the context. The rules remain unchanged except for the addition of one rule, the (*def rule*), and that the use of $\Gamma \vdash B =_{\mathtt{def}} B'$ in the conversion rule really has an effect now, rather than simply postulating $B =_\beta B'$.

5.1 The definition mechanisms and extended typing

Definition 5.1 *(Axioms and rules of the Cube extended with definitions: d ranges over declarations and definitions)*

(axiom) $\quad <> \vdash^e * : \Box$

(start rule) $\quad \dfrac{\Gamma \prec d}{\Gamma d \vdash^e \mathtt{subj}(d) : \mathtt{pred}(d)}$

(weakening rule) $\quad \dfrac{\Gamma \prec d \qquad \Gamma \vdash^e D : E}{\Gamma d \vdash^e D : E}$

(application rule) $\quad \dfrac{\Gamma \vdash^e F : (A\Pi_x)B \qquad \Gamma \vdash^e a : A}{\Gamma \vdash^e (a\delta)F : B[x := a]}$

(abstraction rule) $\quad \dfrac{\Gamma(A\lambda_x) \vdash^e b : B \qquad \Gamma \vdash^e (A\Pi_x)B : S}{\Gamma \vdash^e (A\lambda_x)b : (A\Pi_x)B}$

(def rule) $\quad \dfrac{\Gamma d \vdash^e C : D}{\Gamma \vdash^e dC : [D]_d}$ *if d is a definition*

(conversion rule) $\quad \dfrac{\Gamma \vdash^e A : B \qquad \Gamma \vdash^e B' : S \qquad \Gamma \vdash^e B =_{\mathtt{def}} B'}{\Gamma \vdash^e A : B'}$

(formation rule) $\quad \dfrac{\Gamma \vdash^e A : S_1 \qquad \Gamma(A\lambda_x) \vdash^e B : S_2}{\Gamma \vdash^e (A\Pi_x)B : S_2}$ *If* (S_1, S_2) *is a rule*

Note that in the abstraction rule, it follows that $(A\lambda_x)$ is bachelor in $\Gamma(A\lambda_x)$. The reason is that we can show that if Γ is legal then Γ contains no bachelor main δ-items. Hence as $\Gamma \vdash^e (A\Pi_x)B : S$, Γ has no bachelor δ-items and so $(A\lambda_x)$ cannot be matched in Γ.

By $\Gamma\underline{d} \vdash^e \mathtt{def}(d) : \mathtt{pred}(d)$ in the *(start rule)* and *(weakening rule)*, abbreviating $\Box$ (as in $(\Box\delta)(A\lambda_x)$) is not allowed. Also by $\Gamma\underline{d} \vdash^e \mathtt{pred}(d) : S$, abbreviating kinds is not allowed. One might argue that this last condition could be omitted but it doesn't seem urgent to do so. The (*def rule*) says that if $C : D$ can be deduced from a concatenation of definitions d, then dC will be of type D where all the sub-definitions in d have been unfolded in D. We do not get type dD in order to avoid things like $d\Box$. Note that the *(def rule)* does global substitution in the predicate of all the occurrences of subjects in d. The reason is that d no longer remains in the context. In the conversion rule however, substitution is local as Γ keeps all its information (see Definition 3.4). The following examples show how this works:

Example 5.2 Here is how the term in Example 3.8 and its $\hookrightarrow_\beta$-contractum is typed in $\lambda 2$. (Note how quicker we can type terms once we have definitions. Note also that the derivation

given in Example 3.8 is also valid here, yet it is more clear and efficient to use the definitional segments $(y\delta)(\alpha\lambda_x)$ and $(y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)$. The present derivation is even valid in $\lambda\!\!\rightarrow$, because we don't need $(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)x$ to have a type due to the (*def rule*).)

$\vdash^e_{\lambda 2} * : \Box$	(axiom)
$(*\lambda_\beta) \vdash^e_{\lambda 2} \beta : * : \Box$	(start resp. weakening)
$(*\lambda_\beta)(\beta\lambda_{y'}) \vdash^e_{\lambda 2} y' : \beta : * : \Box$	(start resp. weakening)
$(*\lambda_\beta)(\beta\lambda_{y'})(\beta\delta)(*\lambda_\alpha) \vdash^e_{\lambda 2} y' : \beta : * : \Box, \alpha : *$	(start resp. weakening)
$(*\lambda_\beta)(\beta\lambda_{y'})(\beta\delta)(*\lambda_\alpha) \vdash^e_{\lambda 2} \alpha =_{\mathtt{def}} \beta$	(definition of $=_{\mathtt{def}}$)
$(*\lambda_\beta)(\beta\lambda_{y'})(\beta\delta)(*\lambda_\alpha) \vdash^e_{\lambda 2} y' : \alpha : *$	(conversion)
$(*\lambda_\beta)(\beta\lambda_{y'})(y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y) \vdash^e_{\lambda 2} y : \alpha : *$	(start resp. weakening)
$(*\lambda_\beta)(\beta\lambda_{y'})(y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x) \vdash^e_{\lambda 2} x : \alpha$	(start resp. weakening)
$[\alpha]_{(y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)} \equiv \alpha[x := y][y := y'][\alpha := \beta] \equiv \beta$	
$(*\lambda_\beta)(\beta\lambda_{y'}) \vdash^e_{\lambda 2} (y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)x : \beta$	(def rule)

Also $(*\lambda_\beta)(\beta\lambda_{y'}) \vdash^e_{\lambda 2} (\beta\delta)(*\lambda_\alpha)(y'\delta)(\alpha\lambda_x)x : \beta$ as follows (needed derivation steps, including $(*\lambda_\beta)(\beta\lambda_{y'})(\beta\delta)(*\lambda_\alpha) \vdash^e_{\lambda_2} y' : \alpha$ by *(conversion)* , are left to the reader):

$(*\lambda_\beta)(\beta\lambda_{y'})(\beta\delta)(*\lambda_\alpha)(y'\delta)(\alpha\lambda_x) \vdash^e_{\lambda 2} x : \alpha$ so by (*def rule*):
$(*\lambda_\beta)(\beta\lambda_{y'}) \vdash^e_{\lambda 2} (\beta\delta)(*\lambda_\alpha)(y'\delta)(\alpha\lambda_x)x : \alpha[x := y'][\alpha := \beta] \equiv \beta$

Example 5.3 Also the term of Example 4.14 can be easily and quickly typed in λP (note that this term cannot be typed in $\lambda\!\!\rightarrow$ as the term Q can't):

$(*\lambda_\sigma)(\sigma\lambda_t)((\sigma\Pi_q) * \lambda_Q)((t\delta)Q\lambda_N)(N\delta)(t\delta)(\sigma\lambda_x)((x\delta)Q\lambda_y)(y\delta)((x\delta)Q\lambda_Z) \vdash^e_{\lambda P} Z : (x\delta)Q$
$(*\lambda_\sigma)(\sigma\lambda_t)((\sigma\Pi_q) * \lambda_Q)((t\delta)Q\lambda_N) \vdash^e_{\lambda_P} (N\delta)(t\delta)(\sigma\lambda_x)((x\delta)Q\lambda_y)(y\delta)((x\delta)Q\lambda_Z)Z : (t\delta)Q$

Its $\hookrightarrow_\beta$-contractum gets the same type as follows:

$(*\lambda_\sigma)(\sigma\lambda_t)((\sigma\Pi_q) * \lambda_Q)((t\delta)Q\lambda_N)(t\delta)(\sigma\lambda_x)(N\delta)((x\delta)Q\lambda_Z) \vdash^e_{\lambda P} Z : (x\delta)Q$
$(*\lambda_\sigma)(\sigma\lambda_t)((\sigma\Pi_q) * \lambda_Q)((t\delta)Q\lambda_N) \vdash^e_{\lambda_P} (t\delta)(\sigma\lambda_x)(N\delta)((x\delta)Q\lambda_Z)Z : (t\delta)Q$

Remark 5.4 We need $\Gamma \vdash^e A =_{\mathtt{def}} B$ instead of $A =_\beta B$ in the conversion rule because we want from $(*\lambda_A)(A\delta)(*\lambda_x) \vdash^e A : *$ and y is fresh to derive not only $(*\lambda_A)(A\delta)(*\lambda_x)(A\lambda_y) \vdash^e y : A$ but also $(*\lambda_A)(A\delta)(*\lambda_x)(A\lambda_y) \vdash^e y : x$. This is not possible if conversion is left with $B =_\beta B'$: how can we ever derive $(*\lambda_A)(A\delta)(*\lambda_x)(A\lambda_y) \vdash^e y : x$ as $x \neq_\beta A$? If we change to the conversion rule using $=_{\mathtt{def}}$, then we are fine:
$(*\lambda_A)(A\delta)(*\lambda_x)(A\lambda_y) \vdash^e y : A$
$(*\lambda_A)(A\delta)(*\lambda_x)(A\lambda_y) \vdash^e x : *$
$(*\lambda_A)(A\delta)(*\lambda_x)(A\lambda_y) \vdash^e x =_{\mathtt{def}} A$ and so with conversion,
$(*\lambda_A)(A\delta)(*\lambda_x)(A\lambda_y) \vdash^e y : x$

5.2 Properties of the Cube with definitions

If we look at Section 3.2 and because we have changed $\vdash$ to $\vdash^e$ but left $\twoheadrightarrow_\beta$ unchanged, we see that all the lemmas and theorems which had $\vdash$ in their heading get affected. In this section, we will list these lemmas and theorems for $\vdash^e$ and give their proofs.

Lemma 5.5 *(Free variable lemma for $\vdash^e$)*
Let Γ be a legal context such that $\Gamma \vdash^e B : C$. Then the following holds:

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

2. *$FV(B), FV(C) \subseteq dom(\Gamma)$.*

3. *For s_1 a main item of Γ, $FV(s_1) \subseteq \{\mathtt{subj}(d) \mid d \in \Gamma$-$\mathtt{decl} \cup \Gamma$-$\mathtt{def}, d$ is to the left of s_1 in $\Gamma\}$.*

Proof: *All by induction on the derivation of $\Gamma \vdash^e B : C$.* □

Lemma 5.6 *(Start Lemma for $\vdash^e$)*
Let Γ be a legal context. Then $\Gamma \vdash^e * : \Box$ and $\forall d \in' \Gamma[\Gamma \vdash^e d]$.
Proof: *Γ legal $\Rightarrow \exists B, C[\Gamma \vdash^e B : C]$; now use induction on the derivation $\Gamma \vdash^e B : C$.* □

Lemma 5.7 *(Transitivity Lemma for $\vdash^e$)*
Let Γ and Δ be legal contexts. Then: $[\Gamma \vdash^e \Delta \wedge \Delta \vdash^e A : B] \Rightarrow \Gamma \vdash^e A : B$.
Proof: *Induction on the derivation $\Delta \vdash^e A : B$.* □

Lemma 5.8 *(Definition-shuffling for $\vdash^e$) Let d be a definition.*

1. *If $\Gamma d \Delta \vdash^e C =_{\mathtt{def}} D$ then $\Gamma \underline{d}(\mathtt{def}(d)\delta)(\mathtt{pred}(d)\lambda_{\mathtt{subj}(d)})\Delta \vdash^e C =_{\mathtt{def}} D$.*

2. *If $\Gamma d \Delta \vdash^e C : D$ then $\Gamma \underline{d}(\mathtt{def}(d)\delta)(\mathtt{pred}(d)\lambda_{\mathtt{subj}(d)})\Delta \vdash^e C : D$.*

Proof: *1. is by induction on the generation of $\Gamma(A\delta)\overline{s}(B\lambda_x)\Delta \vdash^e C =_{\mathtt{def}} D$. 2. is by induction on the proof of $\Gamma(A\delta)\overline{s}(B\lambda_x)\Delta \vdash^e C : D$ using 1. for conversion.* □

Lemma 5.9 *(Thinning for $\vdash^e$)*

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

2. *If Γ and Δ are legal contexts such that $\Gamma \subseteq' \Delta$ and if $\Gamma \vdash^e A : B$, then $\Delta \vdash^e A : B$.*

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

- *If $\Gamma\Delta \vdash^e A : B$, $\Gamma \vdash^e C : S$, x is fresh, and no λ-item in Δ is bound by a δ-item in Γ, then also $\Gamma(C\lambda_x)\Delta \vdash^e A : B$. We show this by induction on the derivation $\Gamma\Delta \vdash^e A : B$ using 1. for conversion.*

- *If $\Gamma\overline{s}\Delta \vdash^e A : B$, $\Gamma\overline{s} \vdash^e C : D : S$, $[C]_{\overline{s}} \equiv C$, x is fresh, $\overline{s}$ is well-balanced, then also $\Gamma(C\delta)\overline{s}(D\lambda_x)\Delta \vdash^e A : B$. We show this by induction on the derivation $\Gamma\overline{s}\Delta \vdash^e A : B$. In the case of (start) where $\Gamma(A\delta)\overline{s}(B\lambda_x) \vdash^e x : A$ comes from $\Gamma\overline{s} \vdash^e A : B : S$, $[A]_{\overline{s}} \equiv A$, x fresh, then $[A]_{(C\delta)\overline{s}(D\lambda_x)} \equiv A$ because x fresh and $\Gamma(C\delta)\overline{s}(D\lambda_x) \vdash^e A : B : S$ by IH.*

- *If $\Gamma\overline{s}(A\lambda_x)\Delta \vdash^e B : C, (A\lambda_x)$ bachelor, $\overline{s}$ well-balanced, $\Gamma\overline{s} \vdash^e D : A, [D]_{\overline{s}} \equiv D$, then $\Gamma(D\delta)\overline{s}(A\lambda_x)\Delta \vdash^e B : C$ is shown by induction on the derivation $\Gamma\overline{s}(A\lambda_x)\Delta \vdash^e B : C$ (for conversion, use 1.).* □

Lemma 5.10 *(Substitution lemma for $\vdash^e$) Let d be a definition.*

1. *If $\Gamma d \Delta \vdash^e A =_{\mathtt{def}} B$, A and B are $\Gamma d\Delta$-legal terms, then $\Gamma[\Delta]_d \vdash^e [A]_d =_{\mathtt{def}} [B]_d$*

2. *If B is a Γd-legal term, then $\Gamma d \vdash^e B =_{\mathtt{def}} [B]_d$*

3. *If $\Gamma(A\delta)(B\lambda_x)\Delta \vdash^e C : D$ then $\Gamma\Delta[x := A] \vdash^e C[x := A] : D[x := A]$*

4. *If* $\Gamma(B\lambda_x)\Delta \vdash^e C : D$, $\Gamma \vdash^e A : B$, $(B\lambda_x)$ *bachelor in* Γ, *then* $\Gamma\Delta[x := A] \vdash^e C[x := A] : D[x := A]$

5. *If* $\Gamma d\Delta \vdash^e C : D$, *then* $\Gamma[\Delta]_d \vdash^e [C]_d : [D]_d$

Proof:

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

   Case $\Gamma d\Delta \vdash^e d_1 C =_{\mathtt{def}} \underline{d_1}(C[\mathtt{subj}(d_1) := \mathtt{def}(d_1)])$.

   Then $[d_1 C]_d \equiv ([\mathtt{def}(d_1)]_d\delta)[\underline{d_1}]_d([\mathtt{pred}(d_1)]_d\lambda_{\mathtt{subj}(d_1)})[C]_d$

   ($d_1 C$ *is* $\Gamma d\Delta$*-legal* $\Rightarrow \mathtt{subj}(d_1) \notin dom(d)$*)*

   and $[\underline{d1}(C[\mathtt{subj}(d_1) := \mathtt{def}(d_1)])]_d \equiv [\underline{d_1}]_d([C]_d[\mathtt{subj}(d_1) := [\mathtt{def}(d_1)]_d])$,

   hence $\Gamma[\Delta]_d \vdash^e [d_1 C]_d =_{\mathtt{def}} [\underline{d_1}(C[\mathtt{subj}(d_1) := \mathtt{def}(d_1)])]_d$

2. *Induction on the structure of* B.

   Case $B \equiv x \in dom(d)$*: use* $(=_{\mathtt{def}}\ \ def)$.

   Case $B \equiv x \notin dom(d)$*: use* $(=_{\mathtt{def}}\ \ refl)$.

   Case $B \equiv (C\delta)D$*: use* $(=_{\mathtt{def}}\ \ comp1)$.

   Case $B \equiv (C\mathcal{O}_x)D$ $(\mathcal{O} \in \{\lambda, \Pi\})$*: use* $(=_{\mathtt{def}}\ \ comp2)$.

3. *Induction on the derivation rules, using 1., 2. and thinning, 4. Idem and 5. use 3.* □

Lemma 5.11 *(Generation Lemma for* $\vdash^e$*)*

1. *If* $\Gamma \vdash^e x : A$ *then for some* B: $(B\lambda_x) \in' \Gamma$, $\Gamma \vdash^e B : S$, $\Gamma \vdash^e A =_{\mathtt{def}} B$ *and* $\Gamma \vdash^e A : S'$ *for some sort* S'.

2. *If* $\Gamma \vdash^e (A\lambda_x)B : C$ *then for some* D *and sort* S: $\Gamma(A\lambda_x) \vdash^e B : D$, $\Gamma \vdash^e (A\Pi_x)D : S$, $\Gamma \vdash^e (A\Pi_x)D =_{\mathtt{def}} C$ *and if* $(A\Pi_x)D \not\equiv C$ *then* $\Gamma \vdash^e C : S'$ *for some sort* S'.

3. *If* $\Gamma \vdash^e (A\Pi_x)B : C$ *then for some sorts* S_1, S_2: $\Gamma \vdash^e A : S_1$, $\Gamma \vdash^e B : S_2$, $(S_1, S_2) \in \mathcal{R}$, $\Gamma \vdash^e C =_{\mathtt{def}} S_2$ *and if* $S_2 \not\equiv C$ *then* $\Gamma \vdash^e C : S$ *for some sort* S.

4. *If* $\Gamma \vdash^e (A\delta)B : C$, $(A\delta)$ *bachelor in* B *, then for some* D, E, x: $\Gamma \vdash^e A : D$, $\Gamma \vdash^e B : (D\Pi_x)E$, $\Gamma \vdash^e E[x := A] =_{\mathtt{def}} C$ *and if* $E[x := A] \not\equiv C$ *then* $\Gamma \vdash^e C : S$ *for some* S.

5. *If* $\Gamma \vdash^e \overline{s}A : B$, *then* $\Gamma\overline{s} \vdash^e A : B$

Proof: *1., 2., 3. and 4. follow by a tedious but straightforward induction on the derivations (use the thinning lemma). As to 5., we use induction on* $\mathtt{weight}(\overline{s})$. *Case* $\mathtt{weight}(\overline{s}) = 0$*: nothing to prove. If we have proven the hypothesis for all segments* $\overline{s}$ *that obey* $\mathtt{weight}(\overline{s}) \leq 2n$ *and* $\mathtt{weight}(\overline{s}) = 2n + 2$, $\overline{s} \equiv \overline{s_1 s_2}$ *(neither* $\overline{s_1} \equiv \emptyset$ *nor* $\overline{s_2} \equiv \emptyset$*) then by the IH:* $\Gamma\overline{s_1} \vdash^e \overline{s_2}A : B$*, again applying the induction hypothesis gives* $\Gamma\overline{s_1 s_2} \vdash^e A : B$. *If we have proven the hypothesis for all segments* $\overline{s}$ *for which* $\mathtt{weight}(\overline{s}) \leq 2n$ *and* $\mathtt{weight}(\overline{s}) = 2n + 2, \overline{s} \equiv (D\delta)\overline{s_1}(E\lambda_x)$ *where* $\mathtt{weight}(\overline{s_1}) = 2n$ *then an easy induction to the derivation rules shows that one of the following two cases is applicable:*

- $\Gamma\overline{s} \vdash^e A : B'$, $\Gamma \vdash^e [B']_{\overline{s}} =_{\mathtt{def}} B$ *and if* $[B']_{\overline{s}} \not\equiv B$ *then* $\Gamma \vdash^e B : S$ *for some sort* S.

- $\Gamma \vdash^e D : F$, $\Gamma \vdash^e \overline{s_1}(E\lambda_x)A : (F\Pi_y)G$, $\Gamma \vdash^e B =_{\mathtt{def}} G[y := D]$ *and if* $G[y := D] \not\equiv B$ *then* $\Gamma \vdash^e B : S$ *for some sort* S.

In the first case note that $FV(B) \cap dom(\overline{s}) = \emptyset$ *and by thinning* $\Gamma\overline{s} \vdash^e [B']_{\overline{s}} =_{\mathtt{def}} B$, *by substitution* $\Gamma\overline{s} \vdash^e [B']_{\overline{s}} =_{\mathtt{def}} B'$. *So* $\Gamma\overline{s} \vdash^e B' =_{\mathtt{def}} B$ *and by conversion* $\Gamma\overline{s} \vdash^e A : B$. *In the second case we know by the induction hypothesis that* $\Gamma\overline{s_1} \vdash^e (E\lambda_x)A : (F\Pi_y)G$,

Now 2. tells us $\Gamma\overline{s_1}(E\lambda_x) \vdash^e A : L$, $\Gamma\overline{s_1} \vdash^e (E\Pi_x)L =_{\mathtt{def}} (F\Pi_y)G$ *and if* $(E\Pi_x)L \not\equiv (F\Pi_y)G$ *then* $\Gamma\overline{s_1} \vdash^e (F\Pi_y)G : S_1$ *for some* S_1.

This means that $x \equiv y$, $\Gamma\overline{s_1} \vdash^e E =_{\mathtt{def}} F$, $\Gamma\overline{s_1} \vdash^e L =_{\mathtt{def}} G$. *Out of* $\Gamma\overline{s_1} \vdash^e (E\Pi_x)L : S$ *we get by 3. that* $\Gamma\overline{s_1} \vdash^e E : S_2$ *for some sort* S_2, *thinning gives* $\Gamma\overline{s_1} \vdash^e D : F$ *so by conversion and thinning* $\Gamma\overline{s} \vdash^e A : L$.

Out of $\Gamma \vdash^e B =_{\mathtt{def}} G[x := D]$ *we get (thinning and substitution)* $\Gamma\overline{s} \vdash^e B =_{\mathtt{def}} G$, *out of* $\Gamma\overline{s_1} \vdash^e L =_{\mathtt{def}} G$ *we get* $\Gamma\overline{s} \vdash^e L =_{\mathtt{def}} G$, *hence* $\Gamma\overline{s} \vdash^e B =_{\mathtt{def}} L$.

Now if $G[y := D] \not\equiv B$ *then* $\Gamma \vdash^e B : S$ *for some sort* S, *and if* $G[y := D] \equiv B$ *then we get out of* $\Gamma\overline{s_1} \vdash^e (E\lambda_x)A : (F\Pi_y)G$ *that* $\Gamma\overline{s_1} \vdash^e G : S'$ *for some sort* S', *by thinning and substitution we get that* $\Gamma\overline{s} \vdash^e G[y := D] : S'$. *In any case, we get* $\Gamma\overline{s} \vdash^e B : S$ *for some sort* S *and by conversion we may conlude* $\Gamma\overline{s} \vdash^e A : B$. □

Theorem 5.12 *(Subject Reduction for* $\vdash^e$ *and* $\twoheadrightarrow_\beta$*)*
$\Gamma \vdash^e A : B$ *and* $A \twoheadrightarrow A'$ *then* $\Gamma \vdash^e A' : B$.

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

Theorem 5.13 *(Strong Normalisation for the Cube with respect to* $\vdash^e$ *and* $\twoheadrightarrow_\beta$*)*
For all $\vdash^e$*-legal terms* M, M *is strongly normalising with respect to* $\twoheadrightarrow_\beta$.

Proof: *This is a special case of the proof of Theorem 6.27.* □

6 The Cube with definitions and generalised reduction

Now we extend the type system of section 5 by changing the reduction $\twoheadrightarrow_\beta$ into $\hookrightarrow\!\!\!\!\twoheadrightarrow_\beta$. As was the case in section 4 the derivation rules stay the same as those with classical β-reduction, hence almost all lemmas that have been proved for the system in section 5 are still valid. The only properties that have to be investigated are Church-Rosser, Subject Reduction and Strong Normalisation. We will show now that all these properties too are still valid.

Theorem 6.1 *(The general Church Rosser theorem for* $\hookrightarrow\!\!\!\!\twoheadrightarrow_\beta$*)*
If $A \hookrightarrow\!\!\!\!\twoheadrightarrow_\beta B$ *and* $A \hookrightarrow\!\!\!\!\twoheadrightarrow_\beta C$, *then there exists* D *such that* $B \hookrightarrow\!\!\!\!\twoheadrightarrow_\beta D$ *and* $C \hookrightarrow\!\!\!\!\twoheadrightarrow_\beta D$.

Proof: *see Theorem 4.6.* □

Theorem 6.2 *(Subject Reduction for* $\vdash^e$ *and* $\hookrightarrow\!\!\!\!\twoheadrightarrow_\beta$*)*
If $\Gamma \vdash^e A : B$ *and* $A \hookrightarrow\!\!\!\!\twoheadrightarrow_\beta A'$ *then* $\Gamma \vdash^e A' : B$.

Proof: *We only need to consider* $A \hookrightarrow_\beta A'$. *Suppose* $\Gamma \vdash^e dC : D$. *Then by generation,* $\Gamma d \vdash^e C : D$. *Hence by definition-shuffling (5.8, say* $A \equiv \mathtt{def}(d)$, $B \equiv \mathtt{pred}(d)$ *and* $x \equiv \mathtt{subj}(d)$*),* $\Gamma\underline{d}(A\delta)(B\lambda_x) \vdash^e C : D$. *Hence by substitution* $\Gamma\underline{d} \vdash^e C[x := A] : D[x := A]$, *and by* (def rule) $\Gamma \vdash^e \underline{d}(C[x := A]) : [D[x := A]]_{\underline{d}}$, *which is* $\Gamma \vdash^e \underline{d}(C[x := A]) : [D]_d$. *Now by the variable convention* $[D]_d \equiv D$ *so we are done. The compatibility cases are easy.* □

Now we present the proof of SN for the Cube extended with definitions and $\hookrightarrow\!\!\!\!\twoheadrightarrow_\beta$.

6.1 Strong Normalisation

In [BKN 9x], we used the technique of [Barendregt 92] to show Strong Normalisation for $\lambda_{\rightarrow}$ with extended reduction. However, here we use the proof of [Geuvers 94] due to its flexibility and the possibility of its generalisation to systems beyond the Cube, which we may be investigating in the future. Here is the terminology that will be needed. Let $\twoheadrightarrow$ be a reduction relation which contains $\twoheadrightarrow_\beta$, is Church Rosser and for which the least equivalence relation closed under it, denoted $=_{\rightarrow}$ is the same as $=_\beta$. Let $\vdash$ be a typing relation which ranges over $\vdash$ of Section 3 and over $\vdash^e$ of Section 5.

Definition 6.3 *Define a map* $\# : \mathcal{T} \longrightarrow \{0,1,2,3\}$ *by* $\#(\Box) = 3$, $\#(*) = 2$, $\#(x^{\Box}) = 1$, $\#(x^*) = 0$, $\#(A) = \#(\mathtt{endvar}(A))$. *For* $A \in \mathcal{T}$, $\#(A)$ *is called the* **degree** *of* A. *Call a statement* $A : B$ *OK iff* $\#(A) + 1 = \#(B)$, *call a definition* d *OK iff* $\#(\mathtt{def}(d)) = \#(\mathtt{subj}(d)) = \#(\mathtt{pred}(d)) - 1$, *and call a judgement* $\Gamma \vdash A : B$ *OK iff* $A : B$ *is OK, all definitions* $d \in \Gamma$-$\mathtt{def}$ *are OK and for all items* $(C\mathcal{O}_x) \in \Gamma, A, B$ $(\mathcal{O} \in \{\lambda, \Pi\})$: $x : C$ *is OK.*

We shall use $\#$ to prove that the classes of kinds, constructors and objects are mutually exclusive. First we collect some basic facts about $\Box$ and $*$ in the type systems:

Lemma 6.4

1. *If* $\Gamma \vdash A : B$ *then* $A \not\equiv \Box$.
2. *If* Γ *is a legal context, then* $\Box$ *does not occur in* Γ.
3. *If* A *is a legal term, then* $A \equiv \Box$ *or* $\Box$ *does not occur in* A.
4. *Suppose* $\Gamma \vdash A : B$, *then* $\mathtt{endvar}(A) \equiv * \iff B \equiv \Box$.
5. *If* $(A\delta)$ *is an item in a legal context then* $\mathtt{endvar}(A) \not\equiv *$.
6. *If* $(A\delta)$ *is an item in a legal term then* $\mathtt{endvar}(A) \not\equiv *$.

Proof:

1. *induction on the derivation rules.*
2. *simultaneous induction with 3. on the derivation rules using 1.*
4. *induction on the derivation rules; for* $\Rightarrow$ *use 1. and 3. We treat the case in which* $\Gamma \vdash A : B'$ *is a consequence of* $\Gamma \vdash A : B$, $\Gamma \vdash B' : S$ *and* $\Gamma \vdash B = B'$. *From the induction hypothesis it follows that* $B \equiv \Box$. *Then substituting and reducing introduce no* $\Box$ *in* B' *as by 1.* $\Box \notin \Gamma$, *so* $\Box \in B'$. *But then by 3.:* $B' \equiv \Box$.
5. *induction on the derivation rules; use 4. and 2.*
6. *induction on the derivation rules; use 5., 4. and 3.* $\Box$

Now we can prove that whenever $\Gamma \vdash A : B$ then $\#(A) + 1 = \#(B)$.

Lemma 6.5 *For all contexts* Γ *and terms* A, B: *if* $\Gamma \vdash A : B$ *then* $\Gamma \vdash A : B$ *is OK.*
Proof: *We use induction on the derivation rules, we treat three cases.*

- $\Gamma \vdash (a\delta)F : B[x := a]$ *as a consequence of* $\Gamma \vdash F : (A\Pi_x)B$, $\Gamma \vdash a : A$, *then by the induction hypothesis* $\#(x) = \#(A) - 1 = \#(a)$ *and it can easily be seen that* $\#(x) = \#(a) \Rightarrow \#(B[x := a]) = \#(B)$.

- $\Gamma \vdash dC : [D]_d$ *out of* $\Gamma d \vdash C : D$, *then by the induction hypothesis: for all subdefinitions* d' *of* d, $\#(\mathtt{def}(d')) = \#(\mathtt{subj}(d'))$ *so by repeatedly applying* $\#(x) = \#(a) \Rightarrow \#(B[x := a]) = \#(B)$ *we get* $\#([D]_d) = \#(D)$.

- $\Gamma \vdash A : B'$ *out of* $\Gamma \vdash A : B$, $\Gamma \vdash B' : S'$, $\Gamma \vdash B = B'$, *then by the generation corollary 3.16* $B \equiv \Box$ *or* $\Gamma \vdash B : S$ *for some sort* S.

  If $B \equiv \Box$ *then* $\Gamma \vdash B = B'$ *implies* $B' \equiv \Box$ *as in the proof of lemma 6.4.*

  If $B \not\equiv \Box$ *then* $S \not\equiv \Box \wedge B' \not\equiv \Box$ *implies* $S \equiv S'$; *suppose now* $S \equiv \Box$, *then* $\Gamma \vdash B : \Box$ *so by lemma 6.4* $\mathtt{endvar}(B) \equiv *$ *so again by lemma 6.4 also* $\mathtt{endvar}(B') \equiv *$, *hence* $\#(B') = \#(B) = 2$. *If* $S' \equiv \Box$ *then similar* $\#(B) = \#(B') = 2$. □

Corollary 6.6 *If* Γ *is a legal context, then*

1. Γ-*kinds* $\cap$ Γ-*constructors* $= \emptyset$,

   Γ-*kinds* $\cap$ Γ-*objects* $= \emptyset$,

   Γ-*constructors* $\cap$ Γ-*objects* $= \emptyset$,

   $\Box \notin \Gamma$-*kinds* $\cup$ Γ-*constructors* $\cup$ Γ-*objects*.

2. *If* $(A\Pi_x)B$ *is a* Γ-*term then* A *and* B *are both a* Γ-*kind or a* Γ-*type.*

3. *If* $(A\lambda_x)B$ *is a* Γ-*term then* A *is a* Γ-*kind or a* Γ-*type and* B *is a* Γ-*constructor or a* Γ-*object.*

4. *If* $(A\delta)B$ *is a* Γ-*term then* A *and* B *are both a* Γ-*constructor or a* Γ-*object.*

Proof: *1. is a direct consequence of lemma 6.5. 2., 3. and 4. are an easy corollary of the relevant Generation Lemma and Generation Corollary.* □

Lemma 6.7 *(Soundness of* $\twoheadrightarrow$*) If* $A, B \in \mathcal{T}$ *are legal terms such that* $A =_{\twoheadrightarrow} B$ *then there is a path of one-step reductions and expansions via legal terms between* A *and* B.

Proof: *By Church-Rosser there exists a term* C *such that* $A \twoheadrightarrow_\beta C$ *and* $B \twoheadrightarrow_\beta C$. *By Subject Reduction for ordinary* β*-reduction all terms on the path* $A \cdots C \cdots B$ *are legal.* □

Definition 6.8

- *Define the set of untyped* λ*-terms by* $\Lambda = V \mid C \mid (\Lambda\delta)\Lambda \mid (\lambda_V)\Lambda$

- *We say that a term* $M \in \Lambda$ *is* **strongly normalising** *with respect to* $\twoheadrightarrow$ *iff every* $\twoheadrightarrow$*-reduction path starting at* M, *terminates. Note that a priori it isn't clear whether* M *is strongly normalising with respect to* $\twoheadrightarrow_\beta$ *implies that* M *is strongly normalising with respect to* $\twoheadrightarrow_\beta$ *and that the reverse is trivial.*

- *We define* $SN_{\twoheadrightarrow} = \{M \in \Lambda : M$ *is strongly normalising with respect to* $\twoheadrightarrow\}$.

- *For* $A, B \subseteq \Lambda$ *we define* $A \longrightarrow B = \{M \in \Lambda \mid \forall N \in A[(N\delta)M \in B]\}$.

Definition 6.9 *Define the* **key redex** *of a term M as follows:*

1. *$(A\delta)(B\lambda_x)C$ has key redex $(A\delta)(B\lambda_x)C$.*
2. *If M has key redex N, then $(P\delta)M$ has key redex N.*

Define $\mathbf{red_k}(M)$ to be the term obtained from M by contracting its key redex. Note that not all terms have a key redex and that if a term has a key redex then it is unique.

Definition 6.10

- *Define the set of base terms $\mathcal{B}_\to \subseteq \Lambda$ by $V \subseteq \mathcal{B}_\to$, and if $M \in \mathcal{B}_\to, N \in SN_\to$ then also $(N\delta)M \in \mathcal{B}_\to$.*
- *Call $X \subseteq \Lambda$ **saturated**$_\to$ iff: $X \subseteq SN_\to$, $\mathcal{B}_\to \subseteq X$ and for all $M \in \Lambda$: if $M \in SN_\to$ and $red_k(M) \in X$ then also $M \in X$.*
- *Define $SAT_\to = \{X \subseteq \Lambda : X \text{ is saturated}_\to\}$*

Lemma 6.11

1. *$SN_\to \in SAT_\to$.*
2. *$\forall X \in SAT_\to : X \neq \emptyset$.*
3. *If $N \in SN_\to, M \in X \in SAT_\to$ and $x \notin FV(M)$ then $(N\delta)(\lambda_x)M \in X$. (Note here that [Geuvers 94] takes $(N\delta)(M\delta)(\lambda_y)(\lambda_x)y$ instead of $(N\delta)(\lambda_x)M$. The first however, will not fit our purposes as is explained in Remark 6.25)*
4. *$A, B \in SAT_\to \Rightarrow A \longrightarrow B \in SAT_\to$.*
5. *If I is a set and $X_i \in SAT_\to$ for all $i \in I$, then $\bigcap_{i\in I} X_i \in SAT_\to$.*

Proof: *1. $SN_\to \subseteq SN_\to$, $\mathcal{B}_\to \subseteq SN_\to$. Furthermore, if $M \in SN_\to$ and $\mathbf{red_k}(M) \in SN_\to$ then also $M \in SN_\to$ as $\twoheadrightarrow_\beta \subseteq \twoheadrightarrow$. 2. and 3. by the definition of saturated sets. 5. is easy. Here we treat 4. Suppose $A, B \in SAT_\to$.*

- *As $v \in A$ for all $v \in V$, we see: $t \in A \longrightarrow B \Rightarrow (v\delta)t \in B \Rightarrow (v\delta)t \in SN_\to \Rightarrow t \in SN_\to$. So $A \longrightarrow B \subseteq SN_\to$.*
- *If $M \in \mathcal{B}, N \in A$ then $N \in SN$ so $MN \in \mathcal{B} \subseteq B$, so $\mathcal{B} \subseteq A \longrightarrow B$.*
- *If $M \in SN_\to$, $\mathbf{red_k}(M) \in A \longrightarrow B$ then for all $N \in A$: $(N\delta)\mathbf{red_k}(M) \in B$ hence $(N\delta)M \in B$, hence also $M \in A \longrightarrow B$.* □

We define three maps, first $\mathcal{V}$ of Γ-kinds to the function space of $SAT_\to$, then $[\![\ ]\!]_\xi$ of Γ-terms\Γ-objects to elements of the function space of $SAT_\to$, and third $(\!(\ )\!)_\rho$ of Γ-terms to Λ, such that when certain conditions are met we have:
$\Gamma \vdash A : B : \Box \Rightarrow [\![A]\!]_\xi \in \mathcal{V}(B), [\![B]\!]_\xi \in SAT_\to$ and $\Gamma \vdash A : B \Rightarrow (\!(A)\!)_\rho \in [\![B]\!]_\xi$.

Remark 6.12 It can easily be seen using lemma 6.5 that all kinds look like $*$ with some Π-applications and definitions in front of it, and have no bachelor λ- or δ-items.

Definition 6.13 *Define for all kinds A the set-theoretical interpretation of A as follows:*

- $\mathcal{V}(*) = SAT_{\to}$,
- $\mathcal{V}((A\Pi_{x^\square})B) = \mathcal{V}(A) \to \mathcal{V}(B)$, *the function space of* $\mathcal{V}(A)$ *to* $\mathcal{V}(B)$
- $\mathcal{V}((A\Pi_{x^*})B) = \mathcal{V}(B)$
- $\mathcal{V}(dA) = \mathcal{V}(A)$ *if d a definition*

Now define $\mathcal{U} = \bigcup\{\mathcal{V}(A) \mid A \text{ is a } \vdash\text{-kind}\}$.

Lemma 6.14

1. *If A is a legal kind, B a legal constructor and C is a legal object, then* $\mathcal{V}(A) = \mathcal{V}(A[x^\square := B])$ *and* $\mathcal{V}(A) = \mathcal{V}(A[x^* := C])$.
2. *If dA is a legal kind (remember Remark 6.12), d is a definition, then* $\mathcal{V}([A]_d) = \mathcal{V}(A)$.

Proof: *1. is by induction on the structure of A (note: A has no bachelor δ- or λ-items), 2. is by 1. (note: all definienda in a definition are either constructors or objects).* □

Definition 6.15 *Let Γ be a $\vdash$-legal context.*

- *A* **Γ-constructor valuation,** *notation $\xi \models^\square \Gamma$, is a map $\xi : V^\square \longrightarrow \mathcal{U}$ such that for all $(A\lambda_x) \in' \Gamma$ with A a Γ-kind (i.e. $x \in V^\square$): $\xi(x) \in \mathcal{U}(A)$.*
- *If ξ is a constructor valuation, then $[\![\ ]\!]_\xi : \Gamma\text{-terms}\backslash\Gamma\text{-objects} \to \mathcal{U}$ is defined inductively as follows:*

$$
\begin{array}{lcl}
[\![\square]\!]_\xi & := & SN_{\to} \\
[\![*]\!]_\xi & := & SN_{\to} \\
[\![x^\square]\!]_\xi & := & \xi(x^\square) \\
[\![(A\delta)B]\!]_\xi & := & \begin{cases} [\![B]\!]_\xi[\![A]\!]_\xi & \text{if } A \in \Gamma\text{-constructors} \\ [\![B]\!]_\xi & \text{if } A \in \Gamma\text{-objects} \end{cases} \\
[\![(A\lambda_x)B]\!]_\xi & := & \begin{cases} \lambda f \in \mathcal{V}(A).[\![B]\!]_{\xi(x:=f)} & \text{if } A \in \Gamma\text{-kinds} \\ [\![B]\!]_\xi & \text{if } A \in \Gamma\text{-types} \end{cases} \\
[\![(A\Pi_x)B]\!]_\xi & := & \begin{cases} [\![A]\!]_\xi \to \bigcap_{f \in \mathcal{V}(A)} [\![B]\!]_{\xi(x:=f)} & \text{if } A \in \Gamma\text{-kinds}, x \in V^\square \\ [\![A]\!]_\xi \to [\![B]\!]_\xi & \text{if } A \in \Gamma\text{-types}, x \in V^* \end{cases}
\end{array}
$$

where $\xi(x := N)$ is the valuation that assigns $\xi(y)$ to $y \not\equiv x$ and N to x. Furthermore, with $[\![A]\!]_\xi[\![B]\!]_\xi$ we mean application of the function $[\![A]\!]_\xi$ onto its argument $[\![B]\!]_\xi$ and by λ we mean function-abstraction.

Before we verify that $[\![\ ]\!]_\xi$ is well defined, here are some helpful facts about $[\![\ ]\!]_\xi$.

Lemma 6.16 *Let $A, A' \in \Gamma\text{-terms}\backslash\Gamma\text{-objects}, B \in \Gamma\text{-constructors}, C \in \Gamma\text{-objects}, x^\square$ a constructor variable and x^* an object variable. Then*

1. $[\![A[x^{\Box} := B]]\!]_\xi = [\![A]\!]_{\xi(x^{\Box}:=[\![B]\!]_\xi)}$

2. $[\![A[x^* := C]]\!]_\xi = [\![A]\!]_\xi$

3. $A =_\beta A' \Rightarrow [\![A]\!]_\xi = [\![A']\!]_\xi$

Proof: *1. and 2. are by induction on the structure of A. 3. is by induction on the length of the path from A to A' through the legal terms (use Lemma 6.7).* □

Remark 6.17 Note that $=_\beta$ and $=_{\rightarrow}$ are the same equality relations.

Lemma 6.18 *(Soundness of $[\![\ ]\!]_\xi$) If $\Gamma \vdash A : B : \Box$ then for all ξ such that $\xi \models^{\Box} \Gamma$, we have: $[\![A]\!]_\xi$ and $[\![B]\!]_\xi$ are well-defined and $[\![A]\!]_\xi \in \mathcal{V}(B)$, $[\![B]\!]_\xi \in SAT_{\rightarrow}$.*

Proof: *By induction on the derivation rules. We treat two cases:*

- *$\Gamma \vdash (a\delta)F : B[x := A]$ as a consequence of $\Gamma \vdash F : (A\Pi_x)B$ and $\Gamma \vdash a : A$. It is not difficult to see that $[\![B[x := A]]\!]_\xi \in SAT_{\rightarrow}$ if $[\![B[x := A]]\!]_\xi$ is a kind, because by Lemma 6.5, then also B is a kind. Furthermore, by the induction hypothesis $[\![F]\!]_\xi \in \mathcal{V}((A\Pi_x)B)$ and if A is a kind then also $[\![a]\!]_\xi \in \mathcal{V}(A)$. If A is not a kind, then $[\![(a\delta)F]\!]_\xi = [\![F]\!]_\xi \in \mathcal{V}((A\Pi_x)B) = \mathcal{V}(B)$. If A is a kind, then $[\![F]\!]_\xi \in \mathcal{V}((A\Pi_x)B) = \mathcal{V}(A) \to \mathcal{V}(B)$ and hence $[\![(a\delta)F]\!]_\xi = [\![F]\!]_\xi[\![a]\!]_\xi \in \mathcal{V}(B) \overset{\text{Lemma 6.14}}{=} \mathcal{V}(B[x := a])$.*

- *$\Gamma \vdash dC : [D]_d$ as a consequence of $\Gamma d \vdash C : D$. Then by the induction hypothesis $[\![C]\!]_\xi \in \mathcal{V}(D)$ for all $\xi \models^{\Box} \Gamma d$ and if D is a kind, then $[\![D]\!]_\xi \in SAT_{\rightarrow}$. Now let $\xi \models^{\Box} \Gamma$, then $[\![dC]\!]_\xi \overset{\text{Lemma 6.16.3}}{=} [\![[C]_d]\!]_\xi = [\![C]\!]_{\xi'}$ where $\xi'(x^{\Box}) = \xi(x^{\Box})$ if $x^{\Box}$ is not the subject of a subdefinition in d, and $\xi'(x^{\Box}) = [\![def(d')]\!]_\xi$ if $x^{\Box}$ is the subject of a d' a subdefinition of d. But $\xi' \models^{\Box} \Gamma d$, so $[\![C]\!]_{\xi'} \in \mathcal{V}(D) \overset{\text{Lemma 6.14}}{=} \mathcal{V}([D]_d)$.* □

Definition 6.19 *If $\xi \models^{\Box} \Gamma$, then we call ξ cute with respect to Γ if for all $d \in \Gamma$-$\mathtt{def}$ such that $\mathtt{subj}(d) \in V^{\Box}$, $\xi(\mathtt{subj}(d)) = [\![\mathtt{def}(d)]\!]_\xi$.*

Lemma 6.20

1. *If $\xi \models^{\Box} \Gamma$ and A is Γ-legal, then $[\![A]\!]_\xi$ depends only on the values of ξ on the free constructor variables of A.*

2. *If $\xi \models^{\Box} \Gamma$ then there is a cute ξ' such that $\xi' \models^{\Box} \Gamma$ and $\xi' = \xi$ on the non-definitional constructor variables of $dom(\Gamma)$.*

3. *If $\xi \models^{\Box} \Gamma$ and ξ is cute with respect to Γ then $\Gamma \vdash A =_{\mathtt{def}} B \Longrightarrow [\![A]\!]_\xi = [\![B]\!]_\xi$.*

Proof: *1. is easy, 2. is a consequence of 1. and 3. is proved by induction on the generation of $=_{\mathtt{def}}$ using Lemma 6.16.*

Definition 6.21

- *Let $\xi \models^{\Box} \Gamma$ such that ξ is cute with respect to Γ. An object valuation of Γ with respect to ξ, notation $\rho, \xi \models \Gamma$, is a map $\rho : V \to \Lambda$ such that for all $(A\lambda_x) \in' \Gamma$: $\rho(x) \in [\![A]\!]_\xi$ (regardless of whether $A \in \Gamma$-kinds or $A \in \Gamma$-types).*

- *For $\rho, \xi \models \Gamma$ (note: this implies ξ is cute, define $(\![\ ]\!)_\rho : \Gamma$-terms $\longrightarrow \Lambda$ as follows:*

$$\begin{array}{rcll}
(\![x]\!)_\rho & := & \rho(x) & \\
(\![*]\!)_\rho & := & * & \\
(\![\Box]\!)_\rho & := & \Box & \\
(\![(N\delta)M]\!)_\rho & := & ((\![N]\!)_\rho\delta)(\![M]\!)_\rho & \\
(\![(A\lambda_x)B]\!)_\rho & := & ((\![A]\!)_\rho\delta)(\lambda_y)(\lambda_x)(\![B]\!)_{\rho(x:=x)} & (\textit{where } y \notin FV(B)) \\
(\![(A\Pi_x)B]\!)_\rho & := & ((\lambda_y)(\![B]\!)_{\rho(x:=y)}\delta)((\![A]\!)_\rho\delta)x & (\textit{where } y \notin FV(B))
\end{array}$$

  Note that we need BC *to ensure that no unwanted bindings occur in the case $(A\Pi_x)B$. The use of the endvariable x in this case is not essential, we also could reserve one special variable w that should not be used otherwise and define $(\![(A\Pi_x)B]\!)_\rho$ to be the term $((\lambda_x)(\![B]\!)_{\rho(x:=x)}\delta)((\![A]\!)_\rho\delta)w$.*

- *We define another map $\lceil\ \rceil : \Gamma$-terms $\longrightarrow \Lambda$ by*

$$\begin{array}{rcll}
\lceil x\rceil & := & x & \\
\lceil *\rceil & := & * & \\
\lceil \Box\rceil & := & \Box & \\
\lceil (N\delta)M\rceil & := & (\lceil N\rceil\delta)\lceil M\rceil & \\
\lceil (A\lambda_x)B\rceil & := & (\lceil A\rceil\delta)(\lambda_y)(\lambda_x)\lceil B\rceil & (\textit{where } y \notin FV(B)) \\
\lceil (A\Pi_x)B\rceil & := & ((\lambda_y)\lceil B(x:=y)\rceil\delta)(\lceil A\rceil\delta)x & (\textit{where } y \notin FV(B))
\end{array}$$

Definition 6.22 *Let Γ be a context, $A, B \in \Gamma$-terms. Γ satisfies that A is of type B with respect to $\vdash$ and $\twoheadrightarrow$, notation $\Gamma \models^{\vdash}_{\twoheadrightarrow} A : B$, iff $\forall \xi, \rho[\rho, \xi \models \Gamma \Rightarrow (\![A]\!)_\rho \in [\![B]\!]_\xi]$.*

Lemma 6.23

1. *If $\Gamma(A\delta)\underline{d}(B\lambda_x)\Delta$ is a legal context and $\rho, \xi \models \Gamma(A\delta)\underline{d}(B\lambda_x)\Delta$ then $(\![A]\!)_\rho \in [\![B]\!]_\xi$ and $(\![B]\!)_\rho \in SAT_{\twoheadrightarrow}$.*
2. *$\Gamma d \models A : B \Longrightarrow \Gamma \models dA : [B]_d$*

Proof: *1. is by induction on the derivation rules. 2. is by induction on* weight*(d). If $d \equiv \emptyset$ then nothing to prove, suppose now $d \equiv (C\delta)\overline{s}_1(D\lambda_x)\overline{s}_2$. Then by the induction hypothesis $\Gamma(C\delta)\overline{s}_1(D\lambda_x) \models \overline{s}_2 A : [B]_{\overline{s}_2}$.*

- *Suppose $x \in V^*$. Let $\rho, \xi \models \Gamma\overline{s}_1$. Then for all $E \in [\![D]\!]_\xi$ we have $\rho(x := E), \xi \models \Gamma(C\delta)\overline{s}_1(D\lambda_x)$. Hence $(\![\overline{s}_2 A]\!)_{\rho(x:=E)} \in [\![[B]_{\overline{s}_2}]\!]_\xi$, hence $(\lambda_x)(\![\overline{s}_2 A]\!)_{\rho(x:=x)} \in [\![D]\!]_\xi \rightarrow [\![[B]_{\overline{s}_2}]\!]_\xi$ and also $((\![D]\!)_\rho\delta)(\lambda_y)(\lambda_x)(\![\overline{s}_2 A]\!)_{\rho(x:=x)} \in [\![D]\!]_\xi \rightarrow [\![[B]_{\overline{s}_2}]\!]_\xi$ (by 1. $(\![D]\!)_\rho \in SAT_{\twoheadrightarrow}$, use Lemma 6.11). This means $\Gamma\overline{s}_1 \models (D\lambda_x)\overline{s}_2 A : (D\Pi_x)[B]_{\overline{s}_2}$, so by the induction hypothesis $\Gamma \models \overline{s}_1(D\lambda_x)\overline{s}_2 A : ([D]_{\overline{s}_1}\Pi_x)[B]_{\overline{s}_1\overline{s}_2}$. If $\rho, \xi \models \Gamma$ then by 1. $(\![C]\!)_\rho \in [\![D]\!]_\xi$ and $(\![\overline{s}_1(D\lambda_x)\overline{s}_2 A]\!)_\rho \in [\![[D]_{\overline{s}_1}]\!]_\xi \rightarrow [\![[B]_{\overline{s}_1\overline{s}_2}]\!]_\xi$, hence $((\![C]\!)_\rho\delta)(\![\overline{s}1(D\lambda_x)\overline{s}_2 A]\!)_\rho \in [\![[B]_{\overline{s}_1\overline{s}_2}]\!]_\xi = [\![[B]_{(C\delta)\overline{s}_1(D\lambda_x)\overline{s}_2}]\!]_\xi$, so $\Gamma \models (C\delta)\overline{s}_1(D\lambda_x)\overline{s}_2 A : [B]_{(C\delta)\overline{s}_1(D\lambda_x)\overline{s}_2}$.*
- *Suppose $x \in V^\Box$. Let $\rho, \xi \models \Gamma\overline{s}_1$. Then $\rho(x := E), \xi(x := f) \models \Gamma(C\delta)\overline{s}_1(D\lambda_x)$ for all $f \in \mathcal{U}(D)$ and $E \in [\![D]\!]_\xi$, so $(\![\overline{s}_2 A]\!)_{\rho(x:=E)} \in [\![[B]_{\overline{s}_2}]\!]_\xi\xi(x := f)$, hence $(\lambda_x)(\![\overline{s}_2 A]\!)_{\rho(x:=x)} \in [\![D]\!]_\xi \rightarrow \bigcap_{f\in\mathcal{U}(D)}[\![[B]_{\overline{s}_2}]\!]_{\xi(x:=f)}$. But then also $((\![D]\!)_\rho\delta)(\lambda_y)(\lambda_x)(\![\overline{s}_2 A]\!)_{\rho(x:=x)} \in [\![D]\!]_\xi \rightarrow$*

$\bigcap_{f\in\mathcal{U}(D)}[\![[B]_{\overline{s}_2}]\!]_{\xi(x:=f)}$ *(use 1. and Lemma 6.11). Hence we see:* $\Gamma\overline{s}_1 \models (D\lambda_x)\overline{s}_2A : (D\Pi_x)[B]_{\overline{s}_2}$, *so by IH* $\Gamma \models \overline{s}_1(D\lambda_x)\overline{s}_2A : ([D]_{\overline{s}_1}\Pi_x)[B]_{\overline{s}_1\overline{s}_2}$.

Now let $\rho,\xi \models \Gamma$. *Then* $(\!(\overline{s}_1(D\lambda_x)\overline{s}_2A)\!)_\rho \in [\![([D]_{\overline{s}_1}\Pi_x)[B]_{\overline{s}_1\overline{s}_2}]\!]_\xi$ *and* $(\!(C)\!)_\rho \in [\![[D]_{\overline{s}_1}]\!]_\xi$ *by 1., so* $((\!(C)\!)_\rho\rho\delta)((\!(\overline{s}_1(D\lambda_x)\overline{s}_2A)\!)_\rho\rho) \in \bigcap_{f\in\mathcal{U}(D)}[\![[B]_{\overline{s}_1\overline{s}_2}]\!]_{\xi(x:=f)}$.

This means $(\!((C\delta)\overline{s}_1(D\lambda_x)\overline{s}_2A)\!)_\rho\rho \in [\![[B]_{\overline{s}_1\overline{s}_2}]\!]_{\xi(x:=[\![C]\!]_\xi} = [\![[B]_{\overline{s}_1\overline{s}_2}[x := C]]\!]_\xi \overset{BC}{=} [\![[B]_{(C\delta)\overline{s}_1(D\lambda_x)\overline{s}_2}]\!]_\xi$, *hence* $\Gamma \models (C\delta)\overline{s}_1(D\lambda_x)\overline{s}_2A : [B]_{(C\delta)\overline{s}_1(D\lambda_x)\overline{s}_2}$. □

Lemma 6.24 *(*$(\!(\ )\!)_\rho$ *versus* $\lceil\ \rceil$*)*

1. $\forall M \in \Gamma$*-terms,* $\forall\rho$: $(\!(M)\!)_\rho \equiv \lceil M\rceil[\vec{x} := \rho(\vec{x})]$ *where* $\vec{x}$ *are the free variables of* M.
2. *If* $\overline{s}$ *is a well-balanced segment then* $\lceil\overline{s}A\rceil \equiv \lceil\overline{s}\rceil\lceil A\rceil$ *and* $\lceil\overline{s}\rceil$ *is also well-balanced. Moreover,* $FV(\lceil A\rceil) = FV(A)$.
3. *For all* $M \in \Gamma$*-terms:* $\lceil M\rceil$ *is strongly normalising* $\Rightarrow$ M *is strongly normalising.*

Proof: *The first statement is easy to verify. The second statement is also easy. The third statement can be proved as follows: we prove by induction on the structure of* M, *that whenever* $M \twoheadrightarrow N$, *then* $\lceil M\rceil \twoheadrightarrow \lceil N\rceil$. *We show the only non-trivial case (note that when* $\twoheadrightarrow$ *is* $\twoheadrightarrow_\beta$, *then* $\overline{s} \equiv \emptyset$*). If* $M \equiv (A\delta)\overline{s}(B\lambda_x)C \twoheadrightarrow \overline{s}(C[x := A]) \equiv N$, *then*

$\lceil M\rceil \equiv (\lceil A\rceil\delta)\lceil\overline{s}(B\lambda_x)C\rceil \equiv (\lceil A\rceil\delta)\lceil\overline{s}\rceil(\lceil B\rceil\delta)(\lambda_y)(\lambda_x)\lceil C\rceil$
$\twoheadrightarrow (\lceil A\rceil\delta)\lceil\overline{s}\rceil(\lambda_x)\lceil C\rceil$ *(note that* $y \notin FV((\lambda_x)\lceil C\rceil)$*)*
$\twoheadrightarrow \lceil\overline{s}\rceil\lceil C\rceil[x := \lceil A\rceil] \equiv \lceil\overline{s}\rceil\lceil C[x := A]\rceil \equiv \lceil\overline{s}(C[x := A])\rceil \equiv \lceil N\rceil$. □

Remark 6.25 With this Lemma, it becomes clear why we depart from [Geuvers 94] by using $\lceil(A\lambda_x)B\rceil$ to be $(\lceil A\rceil\delta)(\lambda_y)(\lambda_x)\lceil B\rceil$ instead of $(\lceil A\rceil\delta)((\lambda_x)\lceil B\rceil\delta)(\lambda_u)(\lambda_v)u$.

Consider for example $P \equiv (A\delta)(B\delta)(C\lambda_x)(D\lambda_y)E$ and $Q \equiv (B\delta)(C\lambda_x)E[y := A]$. It is obvious that $P \hookrightarrow_\beta Q$ and that $\lceil P\rceil \equiv (\lceil A\rceil\delta)(\lceil B\rceil\delta)(\lceil C\rceil\delta)(\lambda_p)(\lambda_x)(\lceil D\rceil)\delta)(\lambda_q)(\lambda_y)\lceil E\rceil \hookrightarrow\!\!\!\!\rightarrow_\beta \lceil Q\rceil \equiv (\lceil B\rceil\delta)(\lceil C\rceil\delta)(\lambda_p)(\lambda_x)\lceil E\rceil[y := \lceil A\rceil]$. Yet, if we use the translation of [Geuvers 94], then we get $\lceil P\rceil \equiv (\lceil A\rceil\delta)(\lceil B\rceil\delta)(\lceil C\rceil\delta)((\lambda_x)\lceil(D\lambda_y)E\rceil\delta)(\lambda_u)(\lambda_v)u$
$\not\twoheadrightarrow_\beta \lceil Q\rceil \equiv (\lceil B\rceil\delta)(\lceil C\rceil\delta)((\lambda_x)\lceil E\rceil[y := \lceil A\rceil]\delta)(\lambda_s)(\lambda_t)s$.

Lemma 6.26 $\Gamma \vdash A : B \Rightarrow \Gamma \models A : B$

Proof: *Use induction on the structure of* A *to prove that if* $\rho,\xi \models \Gamma$ *then* $(\!(A)\!)_\rho \in [\![B]\!]_\xi$*:*

- $A \equiv x$. *Then by generation for some* B': $\Gamma \vdash B' =_{\mathtt{def}} B$ *and* $(B'\lambda_x) \in \Gamma\text{-}\mathtt{decl} \cup \Gamma\text{-}\mathtt{def}$, *so by* $\rho,\xi \models \Gamma, (B'\lambda_x) \in \Gamma\text{-}\mathtt{decl} \cup \Gamma\text{-}\mathtt{def}$, *and Lemma 6.16, we get* $(\!(A)\!)_\rho = \rho(x) \in [\![B']\!]_\xi = [\![B]\!]_\xi$.
- $A \equiv (P\lambda_x)Q$, *with* $P \in \Gamma$*-kinds.*

  Then by the generation lemma for some R, $\Gamma(P\lambda_x) \vdash Q : R$ *with* $\Gamma \vdash (P\Pi_x)R =_{\mathtt{def}} B$, $\Gamma \vdash P : \Box$. *By IH we find that* $(\!(Q)\!)_{\rho(x:=p)} \in [\![R]\!]_{\xi(x:=f)}$ *for all* $p \in [\![P]\!]_\xi, f \in \mathcal{V}(P)$, *so* $(\!(Q)\!)_{\rho(x:=p)} \in \bigcap_{f\in\mathcal{V}(P)}[\![R]\!]_{\xi(x:=f)}$. *By IH also* $(\!(P)\!)_\rho \in [\![\Box]\!]_\xi = SN_\rightarrow$ *so by Lemma 6.11* $(\!(A)\!)_\rho = (\!((P\lambda_x)Q)\!)_\rho = ((\!(P)\!)_\rho\delta)(\lambda_y)(\lambda_x)(\!(Q)\!)_{\rho(x:=x)} \in [\![P]\!]_\xi \to \bigcap_{f\in\mathcal{V}(P)}[\![R]\!]_{\xi(x:=f)} = [\![B]\!]_\xi$.
- $A \equiv (P\lambda_x)Q$ *with* $P \in \Gamma$*-types. Then similar to the previous case.*

- *If $\vdash$ is ordinary typing and $A \equiv (P\delta)Q$ with $P \in \Gamma$-objects. Then $\Gamma \vdash Q : (R\Pi_x)T$, $\Gamma \vdash P : R$ for some R, T with $\Gamma \vdash T[x := P] =_{\mathtt{def}} B$ (again generation lemma). Now by IH and lemma 6.11 we see that $(\!|Q|\!)_\rho \in [\![R]\!]_\xi \longrightarrow [\![T]\!]_\xi$ and $(\!|P|\!)_\rho \in [\![R]\!]_\xi$,*

  so $(\!|A|\!)_\rho = (\!|(P\delta)Q|\!)_\rho = ((\!|P|\!)_\rho\delta)(\!|Q|\!)_\rho \in [\![T]\!]_\xi = [\![T[x := P]]\!]_\xi = [\![B]\!]_\xi$.

- *$A \equiv dP$ where d is a definition. Then by the Generation Lemma $\Gamma d \vdash^e P : B$. By the induction hypothesis we then know that $\Gamma d \models P : B$, hence by Lemma 6.23.2 we get that $\Gamma \models dP : [B]_d$, but $[B]_d \equiv B$, so $\Gamma \models dP : B$.*

- *$A \equiv (P\delta)Q$ with $P \in \Gamma$-constructors where $(P\delta)$ is bachelor in $(P\delta)Q$ then also similar.*

- *$A \equiv (P\Pi_x)Q$. Then by generation $\Gamma \vdash P : S_1$, $\Gamma(P\lambda_x) \vdash Q : S_2$, $S_2 =_\beta B$.*

  If $P \in \Gamma$-kinds, then IH says $(\!|P|\!)_\rho \in [\![\Box]\!]_\xi$, $(\!|Q|\!)_{\rho(x:=p)} \in [\![S_2]\!]_{\xi(x:=f)}$ for all $p \in [\![P]\!]_\xi, f \in \mathcal{V}(P)$, hence $[\![P]\!]_\xi \in SN_\to$, $(\lambda_x)(\!|Q|\!)_{\rho(x:=x)} \in SN$.

  But this means $(\!|A|\!)_\rho = ((\lambda_x)(\!|Q|\!)_{\rho(x:=x)}\delta)((\!|P|\!)_\rho\delta)x \in SN = [\![S_2]\!]_\xi = [\![B]\!]_\xi$.

  If $P \in \Gamma$-types, then similar. □

Theorem 6.27 *(Strong Normalisation for the Cube with respect to $\vdash^e$ and $\hookrightarrow\!\!\!\!\to_\beta$)*
For all $\vdash^e$-legal terms M, M is strongly normalising with respect to $\hookrightarrow\!\!\!\!\to_\beta$.

Proof: *Let M be a $\vdash^e$-legal term. Then either $M \equiv \Box$ or for some context Γ and term N, $\Gamma \vdash^e M : N$. In the first case, clearly M is strongly normalising. In the second case, define canonical elements $c^A \in \mathcal{V}(A)$ for all $A \in \Gamma$-kinds as follows:*

$$\begin{array}{lcll} c^* & := & SN_{\to_\beta} & \\ c^{(A\Pi_x)B} & := & \lambda f \in \mathcal{V}(A).c^B & \text{if } A \in \Gamma\text{-kinds}, x \in V^\Box \\ c^{(A\Pi_x)B} & := & c^B & \text{if } A \in \Gamma\text{-types}, x \in V^* \end{array}$$

Take ξ such that $\xi(x) = c^A$ whenever $(A\lambda_x) \in' \Gamma$ and $\xi(\mathtt{subj}(d)) = [\![\mathtt{def}(d)]\!]_\xi$ whenever $d \in' \Gamma$-def and take ρ such that $\rho(\mathtt{subj}(d)) = (\!|\mathtt{def}(d)|\!)_\rho$ for all subdefinitions d of Γ and $\rho(x) = x$ otherwise. Then $\rho, \xi \models \Gamma$, hence $(\!|M|\!)_\rho \in [\![N]\!]_\xi$, where $(\!|M|\!)_\rho = \lceil M \rceil$ as mentioned in lemma 6.24. Hence $\lceil M \rceil \in [\![N]\!]_\xi \subseteq SN_{\to_\beta}$. By lemma 6.24 now also $M \in SN_{\to_\beta}$. □

This Theorem proves also SN for the other Cubes in this paper (the Cube extended with nothing, definitions or $\hookrightarrow\!\!\!\!\to_\beta$) as the legal terms of those Cubes are also legal in the Cube of this section, and SN with respect to $\hookrightarrow\!\!\!\!\to_\beta$ implies SN with respect to $\to_\beta$.

7 Comparing the type system with definitions to other type systems

In this section we will compare the type systems generated by $\vdash^e$ with the one generated by $\vdash$, from two different points of view. The first is the conservativity, where we show that in a certain sense, definitions are harmless. That is, even though we can type more terms using $\vdash^e$ than using $\vdash$, whenever a judgement is derivable in a theory $\mathcal{L}$ using definitions and $\vdash^e$, it is also derivable in the theory $\mathcal{L}$ without definitions, using only $\vdash$ and where all the definitions are unfolded. The second viewpoint is about the effectiveness of derivations. More work has to be done yet but it is certain that there is a gain in using definitions.

7.1 Conservativity

As we saw in example 5.2, in the type systems with definitions there are more legal terms. Therefore, it has to be investigated to what extent the set of legal terms has changed. Note first that all derivable judgements in a type system of the λ-cube are derivable in the same type system extended with definitions as we only extended, not changed, the derivation rules. A second remark concerns the bypassing of the *formation rule* by using the *weakening* and *definition rule* instead: In $\lambda 2$ without definitions we can derive the following by using the formation rules $(*,*)$ and $(\Box,*)$ (take $\Gamma \equiv (*\lambda_\beta)(\beta\lambda_y)$):

$\Gamma \vdash_{\lambda 2} y : \beta : * : \Box$	
$\Gamma(*\lambda_\alpha) \vdash_{\lambda 2} \alpha : *$	(start)
$\Gamma(*\lambda_\alpha)(\alpha\lambda_x) \vdash_{\lambda 2} x : \alpha : *$	(start resp weakening)
$\Gamma(*\lambda_\alpha) \vdash_{\lambda 2} (\alpha\Pi_x)\alpha : *$	(formation rule $(*,*)$)
$\Gamma(*\lambda_\alpha) \vdash_{\lambda 2} (\alpha\lambda_x)x : (\alpha\Pi_x)\alpha$	(abstraction)
$\Gamma \vdash_{\lambda 2} (*\Pi_\alpha)(\alpha\Pi_x)\alpha : *$	(fromation rule $(\Box,*)$)
$\Gamma \vdash_{\lambda 2} (*\lambda_\alpha)(\alpha\lambda_x)x : (*\Pi_\alpha)(\alpha\Pi_x)\alpha$	(abstraction)
$\Gamma \vdash_{\lambda 2} (\beta\delta)(*\lambda_\alpha)(\alpha\lambda_x)x : (\beta\Pi_x)\beta$	(application, we already knew $\Gamma \vdash_{\lambda 2} \beta : *$)
$\Gamma \vdash_{\lambda 2} (y\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_x)x : \beta$	(application, we already knew $\Gamma \vdash_{\lambda 2} y : \beta$)

It is not possible to derive this judgement in $\lambda\!\!\rightarrow$ as the *formation rule* $(\Box,*)$ is needed. Now we observe that the term $(y\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_x)x$ can be seen as x with two definitions added, and using this observation we can derive the judgement in a type system with definition without having to use the formation rules $(*,*)$ and $(\Box,*)$:

$\Gamma \vdash^e_{\lambda\!\!\rightarrow} y : \beta : * : \Box$	
$\Gamma(\beta\delta)(*\lambda_\alpha) \vdash^e_{\lambda\!\!\rightarrow} y : \beta, \alpha : *$	(weakening resp. start)
$\Gamma(\beta\delta)(*\lambda_\alpha) \vdash^e_{\lambda\!\!\rightarrow} \alpha =_{\mathtt{def}} \beta$	(use the definition in the context)
$\Gamma(\beta\delta)(*\lambda_\alpha) \vdash^e_{\lambda\!\!\rightarrow} y : \alpha$	(conversion)
$\Gamma(y\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_x) \vdash^e_{\lambda\!\!\rightarrow} x : \alpha$	(start)
$\Gamma \vdash^e_{\lambda\!\!\rightarrow} (y\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_x)x : \alpha[x := y][\alpha := \beta] \equiv \beta$	(definition rule)

This example shows that in $\lambda\!\!\rightarrow_{\mathtt{def}}$ we have more legal judgements than in $\lambda\!\!\rightarrow$. Now we take a look at the judgement $\Gamma \vdash (\beta\delta)(*\lambda_\alpha)(M\lambda_x)x : (M\Pi_x)M$ where $M \equiv (y\delta)(\beta\lambda_z)(\beta\delta)(*\lambda_\gamma)\gamma$ and $\Gamma \equiv (*\lambda_\beta)(\beta\lambda_y)$. This judgement can be derived in λC using the formation rules $(\Box,\Box)$, $(\Box,*)$, $(*,\Box)$ and $(*,*)$ in the following way:

$\Gamma \vdash_{\lambda C} \beta : * : \Box$	
$\Gamma(*\lambda_\alpha) \vdash_{\lambda C} \beta : * : \Box$	(weakening)
$\Gamma(*\lambda_\alpha)(\beta\lambda_z) \vdash_{\lambda C} z : \beta : * : \Box$	(start resp. weakening)
$\Gamma(*\lambda_\alpha)(\beta\lambda_z)(*\lambda_\gamma) \vdash_{\lambda C} \gamma : * : \Box$	(start resp. weakening)
$\Gamma(*\lambda_\alpha)(\beta\lambda_z) \vdash_{\lambda C} (*\Pi_\gamma)* : \Box$	(formation rule $(\Box,\Box)$)
$\Gamma(*\lambda_\alpha)(\beta\lambda_z) \vdash_{\lambda C} (*\lambda_\gamma)\gamma : (*\Pi_\gamma)*$	(abstraction)
$\Gamma(*\lambda_\alpha)(\beta\lambda_z) \vdash_{\lambda C} (\beta\delta)(*\lambda_\gamma)\gamma : *$	(application)
$\Gamma(*\lambda_\alpha) \vdash_{\lambda C} (\beta\Pi_z)* : \Box$	(formation rule $(*,\Box)$)
$\Gamma(*\lambda_\alpha) \vdash_{\lambda C} (\beta\lambda_z)(\beta\delta)(*\lambda_\gamma)\gamma : (\beta\Pi_z)*$	(abstraction)

$\Gamma(*\lambda_\alpha) \vdash_{\lambda C} M : *$ (application, $M \equiv (y\delta)(\beta\lambda_z)(\beta\delta)(*\lambda_\gamma)\gamma$)
$\Gamma(*\lambda_\alpha)(M\lambda_x) \vdash_{\lambda C} x : M : *$ (start resp. weakening)
$\Gamma(*\lambda_\alpha) \vdash_{\lambda C} (M\Pi_x)M : *$ (formation rule $(*,*)$)
$\Gamma(*\lambda_\alpha) \vdash_{\lambda C} (M\lambda_x)x : (M\Pi_x)M$ (abstraction)
$\Gamma \vdash_{\lambda C} (*\Pi_\alpha)(M\Pi_x)M : *$ (formation rule $(\Box,*)$)
$\Gamma \vdash_{\lambda C} (*\lambda_\alpha)(M\lambda_x)x : (*\Pi_\alpha)(M\Pi_x)M$ (abstraction)
$\Gamma \vdash_{\lambda C} (\beta\delta)(*\lambda_\alpha)(M\lambda_x)x : (M\Pi_x)M$ (application)

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

$\Gamma \vdash^e_{\lambda_\to} \beta : * : \Box$
$\Gamma(\beta\delta)(*\lambda_\alpha) \vdash^e_{\lambda_\to} \beta : * : \Box$ (weakening)
$\Gamma(\beta\delta)(*\lambda_\alpha)(y\delta)(\beta\lambda_z) \vdash^e_{\lambda_\to} \beta : * : \Box$ (weakening)
$\Gamma(\beta\delta)(*\lambda_\alpha)(y\delta)(\beta\lambda_z)(\beta\delta)(*\lambda_\gamma) \vdash^e_{\lambda_\to} \gamma : *$ (weakening)
$\Gamma(\beta\delta)(*\lambda_\alpha) \vdash^e_{\lambda_\to} (y\delta)(\beta\lambda_z)(\beta\delta)(*\lambda_\gamma)\gamma : *[\gamma := \beta][z := y]$ i.e. $M : *$ (definition rule)
$\Gamma(\beta\delta)(*\lambda_\alpha)(M\lambda_x) \vdash^e_{\lambda_\to} x : M : *$ (start resp. weakening)
$\Gamma(\beta\delta)(*\lambda_\alpha) \vdash^e_{\lambda_\to} (M\Pi_x)M : *$ (formation rule $(*,*)$)
$\Gamma(\beta\delta)(*\lambda_\alpha) \vdash^e_{\lambda_\to} (M\lambda_x)x : (M\Pi_x)M$ (abstraction)
$\Gamma \vdash^e_{\lambda_\to} (\beta\delta)(*\lambda_\alpha)(M\lambda_x)x : (M\Pi_x)M[\alpha := \beta] \equiv (M\Pi_x)M$ (definition rule)

This example shows that in every system of the λ-cube (except λC), adding definitions gives more derivable judgements. As was shown in Eample 5.2, $(*\lambda_\beta)(\beta\lambda_{y'}) \vdash^e_{\lambda 2} (\beta\delta)(*\lambda_\alpha)(y'\delta)(\alpha\lambda_x)x : \beta$ is derivable in $\lambda_{2\mathbf{def}}$ and hence is also derivable in $\lambda C_{\mathbf{def}}$, but this judgement cannot be derived in λC as y is of type β and not of type α. At first sight this might cause the reader to suspect type systems with definitions of having too much derivable judgements. However, we have a conservativity result stating that a judgement that can be derived in $\mathcal{L}_{\mathbf{def}}$ can be derived in $\mathcal{L}$ when all definitions in the whole judgement have been unfolded.

Definition 7.1 *For $\Gamma \vdash^e A : B$ a judgement we define the unfolding of $\Gamma \vdash^e A : B$, $[\Gamma \vdash^e A : B]^u$ to be the judgement obtained from $\Gamma \vdash^e A : B$ in the following way:*

- *first, mark all visible $\delta\lambda$-couples in Γ, A and B,*
- *second, contract in Γ, A and B all these marked $\delta\lambda$-couples.*

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

Example 7.2 $[(*\lambda_\beta)(\beta\lambda_y)(y\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_x)(\alpha\lambda_z) \vdash^e ((\alpha\lambda_u)u\delta)((\alpha\Pi_u)\beta\lambda_v)(x\delta)v : \alpha]^u$ is $(*\lambda_\beta)(\beta\lambda_y)((\alpha\lambda_z)[x := y][\alpha := \beta]) \vdash^e (((x\delta)v)[v := (\alpha\lambda_u)u])[x := y][\alpha := \beta] : \alpha[x := y][\alpha := \beta]$, which is $(*\lambda_\beta)(\beta\lambda_y)(\beta\lambda_z) \vdash^e (y\delta)(\beta\lambda_u)u : \beta$. Note that the resulting context contains only λ-items and that the resulting subject and predicate need not be in normal form.

Theorem 7.3 *Let $\mathcal{L}$ be one of the systems of the Cube, Γ a context with definitions and A, B pseudoterms. If $\Gamma \vdash^e_{\mathcal{L}} A : B$ then $\Gamma' \vdash_{\mathcal{L}} A' : B'$, where $\Gamma' \vdash_{\mathcal{L}} A' : B'$ is $[\Gamma \vdash^e_{\mathcal{L}} A : B]^u$.*

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

- *The last rule applied is the* start rule. *Then* $\Gamma d \vdash^e_{\mathcal{L}} \mathtt{subj}(d) : \mathtt{pred}(d)$ *as a consequence of* $\Gamma \prec d$. *Now if* $d \equiv (A\lambda_x)$ *then by IH* $\Gamma' \vdash_{\mathcal{L}} A' : S$ *(S a sort, x fresh) so by the* start rule $\Gamma'(A'\lambda_x) \vdash_{\mathcal{L}} x : A'$. *On the other hand, if d is a definition, say* $d \equiv (A\delta)\underline{d}(B\lambda_x)$, *then by IH* $(\Gamma\underline{d})' \vdash_{\mathcal{L}} A' : B' : S$ *(S a sort), which is* $\Gamma' \vdash_{\mathcal{L}} A' : B' : S$ *as d will be fully unfolded, and the unfolding of* $\Gamma d \vdash^e_{\mathcal{L}} \mathtt{subj}(d) : \mathtt{pred}(d)$ *is* $\Gamma' \vdash_{\mathcal{L}} \mathtt{def}(d)' : \mathtt{pred}(d)'$ *which is* $\Gamma' \vdash_{\mathcal{L}} A' : B'$ *so we are done.*

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

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

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

- *The last rule applied is the* definition rule. *Then* $\Gamma \vdash^e_{\mathcal{L}} dc : [D]_d$ *as a consequence of* $\Gamma d \vdash C : D$. *By IH,* $\Gamma' \vdash_{\mathcal{L}} [C']_d : [D']_d$ *which is the unfolding of* $\Gamma \vdash^e_{\mathcal{L}} dc : [D]_d$.

Remark 7.4 It is not sufficient in theorem 7.3 to unfold all the definitions in the context only, because a redex in the subject may have been used to change the type when it was still in the context, this is illustrated by $(*\lambda_\beta)(\beta\lambda_y) \vdash^e_{\lambda\to} (\beta\delta)(*\lambda_\alpha)(y\delta)(\alpha\lambda_x)x : \beta$ which cannot be derived using $\vdash_{\lambda\to}$. However, this judgement where all the definitions are unfolded in context, subject and predicate, is derivable using $\vdash$. That is, $(*\lambda_\beta)(\beta\lambda_y) \vdash_{\lambda\to} y : \beta$.

7.2 Shorter derivations

As we already noted, derivations using the definition mechanism seem to need considerably less derivation steps to derive a judgement that can also be derived without definitions. As to the type-checking of terms, we do not think that type-checking in the extended systems will be more difficult than in the λ-cube of Barendregt, nor do we think it will become less.

7.3 Comparison with the systems of the Barendregt cube

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

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

Furthermore, in the terms, there are more visible redexes and all these redexes are subject to contraction.

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

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

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

7.4 Comparison with the type systems of Poll and Severi

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

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

   In our approach, we treat definitions like β-redexes, hence the syntax of pseudoterms remains the same. We only need to change the syntax of contexts and extend the notion of β-equality in a natural way to be able to use the definitions in the context.

2. [SP 93] have a rule that takes a definition out of the context and puts it in front of the term and type. In our extended system however, we only put the definition in front of the term and unfold it in the type. By the conversion rule, now also the type with the definition in front of it instead of the unfolded type can be derived (due to the generation corollary).

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

References

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

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

[BKKS 87] Barendregt, H.P., Kennaway, J.R., Klop, J.W., and Sleep M.R., Needed reduction and spine strategies for the λ-calculus, *Information and Computation 75 (3),* 1191-231, 1987.

[BKN 9x] Bloo, R., Kamareddine, F., Nederpelt, R., Beyond β-reduction in Church's $\lambda_{\rightarrow}$, Computing Science Note 94/20, Eindhoven University of Technology, Department of Mathematics and Computing Science, 1994.

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

[Gardner 94] Gardner, P., Discovering Needed Reductions Using Type Theory, to appear in TACS, 1994.

[Geuvers 94] Geuvers, H., A short and flexible proof of Strong Normalisation for the Calculus of Constructions, notes of a talk given at the BRA types workshop, Bastad, Sweden, 1994.

[KN 93] Kamareddine, F., and Nederpelt, R.P., On stepwise explicit substitution, *International Journal of Foundations of Computer Science 4 (3)*, 197-240, 1993.

[KN 9z] Kamareddine, F., and Nederpelt, R.P., *The beauty of the λ-calculus*, in preparation.

[KN 9y] Kamareddine, F., and Nederpelt, R.P., Canonical Typing and Π–conversion in the Barendregt Cube, to appear in *the Journal of Functional Programming.*

[KN94b] Kamareddine, F., and Nederpelt, R.P., Refining reduction in the λ-calculus, Computing Science Note 94/18, Eindhoven University of Technology, Department of Mathematics and Computing Science, 1994.

[Launchbury 93] Launchbury, J., A natural semantics of lazy evaluation, *ACM POPL 93*, 144-154, 1993.

[Lévy 80] Lévy, J.-J. Optimal reductions in the lambda calculus, in *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, J. Seldin and R. Hindley eds, Academic Press, 1980.

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

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

[NK 94] Nederpelt, R.P., and Kamareddine, F., A unified approach to type theory through a refined λ-calculus, in *λ-calculus and domain theory*, Proceedings of the 1992 conference on *Mathematical Foundations of Programming Semantics*, ed. M. Mislove et. al., 1994, Theoretical Computer Science, Springer.

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

Computing Science Reports

Department of Mathematics and Computing Science
Eindhoven University of Technology

In this series appeared:

91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.
91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.

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	Knowledge Base Systems, a Formal Model, p. 21.
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 types, p. 26.
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.

91/35	F.S. de Boer J.W. Klop C. Palamidessi	Asynchronous communication in process algebra, p. 20.
92/01	J. Coenen J. Zwiers W.-P. de Roever	A note on compositional refinement, p. 27.
92/02	J. Coenen J. Hooman	A compositional semantics for fault tolerant real-time systems, p. 18.
92/03	J.C.M. Baeten J.A. Bergstra	Real space process algebra, p. 42.
92/04	J.P.H.W.v.d.Eijnde	Program derivation in acyclic graphs and related problems, p. 90.
92/05	J.P.H.W.v.d.Eijnde	Conservative fixpoint functions on a graph, p. 25.
92/06	J.C.M. Baeten J.A. Bergstra	Discrete time process algebra, p.45.
92/07	R.P. Nederpelt	The fine-structure of lambda calculus, p. 110.
92/08	R.P. Nederpelt F. Kamareddine	On stepwise explicit substitution, p. 30.
92/09	R.C. Backhouse	Calculating the Warshall/Floyd path algorithm, p. 14.
92/10	P.M.P. Rambags	Composition and decomposition in a CPN model, p. 55.
92/11	R.C. Backhouse J.S.C.P.v.d.Woude	Demonic operators and monotype factors, p. 29.
92/12	F. Kamareddine	Set theory and nominalisation, Part I, p.26.
92/13	F. Kamareddine	Set theory and nominalisation, Part II, p.22.
92/14	J.C.M. Baeten	The total order assumption, p. 10.
92/15	F. Kamareddine	A system at the cross-roads of functional and logic programming, p.36.
92/16	R.R. Seljée	Integrity checking in deductive databases; an exposition, p.32.
92/17	W.M.P. van der Aalst	Interval timed coloured Petri nets and their analysis, p. 20.
92/18	R.Nederpelt F. Kamareddine	A unified approach to Type Theory through a refined lambda-calculus, p. 30.
92/19	J.C.M.Baeten J.A.Bergstra S.A.Smolka	Axiomatizing Probabilistic Processes: ACP with Generative Probabilities, p. 36.
92/20	F.Kamareddine	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 Fω, 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	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach

		Part V: Specification Language, p. 89.
93/14	J.C.M. Baeten J.A. Bergstra	On Sequential Composition, Action Prefixes and Process Prefix, p. 21.
93/15	J.C.M. Baeten J.A. Bergstra R.N. Bol	A Real-Time Process Logic, p. 31.
93/16	H. Schepers J. Hooman	A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27
93/17	D. Alstein P. van der Stok	Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.
93/18	C. Verhoef	A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.
93/19	G-J. Houben	The Design of an Online Help Facility for ExSpect, p.21.
93/20	F.S. de Boer	A Process Algebra of Concurrent Constraint Programming, p. 15.
93/21	M. Codish D. Dams G. Filé M. Bruynooghe	Freeness Analysis for Logic Programs - And Correctness?, p. 24.
93/22	E. Poll	A Typechecker for Bijective Pure Type Systems, p. 28.
93/23	E. de Kogel	Relational Algebra and Equational Proofs, p. 23.
93/24	E. Poll and Paula Severi	Pure Type Systems with Definitions, p. 38.
93/25	H. Schepers and R. Gerth	A Compositional Proof Theory for Fault Tolerant Real-Time Distributed Systems, p. 31.
93/26	W.M.P. van der Aalst	Multi-dimensional Petri nets, p. 25.
93/27	T. Kloks and D. Kratsch	Finding all minimal separators of a graph, p. 11.
93/28	F. Kamareddine and R. Nederpelt	A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.
93/29	R. Post and P. De Bra	GOLD, a Graph Oriented Language for Databases, p. 42.
93/30	J. Deogun T. Kloks D. Kratsch H. Müller	On Vertex Ranking for Permutation and Other Graphs, p. 11.
93/31	W. Körver	Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.
93/32	H. ten Eikelder and H. van Geldrop	On the Correctness of some Algorithms to generate Finite Automata for Regular Expressions, p. 17.

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.
93/35	W. Ferrer and P. Severi	Abstract Reduction and Topology, p. 28.
93/36	J.C.M. Baeten and J.A. Bergstra	Non Interleaving Process Algebra, p. 17.
93/37	J. Brunekreef J-P. Katoen R. Koymans S. Mauw	Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.
93/38	C. Verhoef	A general conservative extension theorem in process algebra, p. 17.
93/39	W.P.M. Nuijten E.H.L. Aarts D.A.A. van Erp Taalman Kip K.M. van Hee	Job Shop Scheduling by Constraint Satisfaction, p. 22.
93/40	P.D.V. van der Stok M.M.M.P.J. Claessen D. Alstein	A Hierarchical Membership Protocol for Synchronous Distributed Systems, p. 43.
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 taxomomy of finite automata construction algorithms, p. 87.
93/44	B.W. Watson	A taxonomy of finite automata minimization algorithms, p. 23.
93/45	E.J. Luit J.M.M. Martin	A precise clock synchronization protocol,p.
93/46	T. Kloks D. Kratsch J. Spinrad	Treewidth and Patwidth of Cocomparability graphs of Bounded Dimension, p. 14.
93/47	W. v.d. Aalst P. De Bra G.J. Houben Y. Kornatzky	Browsing Semantics in the "Tower" Model, p. 19.
93/48	R. Gerth	Verifying Sequentially 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 K.M. van Hee	Application of Marcov Decision Processe 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	Logic Programming and Negation: A Survey, p. 62.
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 Schema with State- & Activitycharts, p. 30.
94/12	T. Kloks D. Kratsch H. Müller	Dominoes, p. 14.
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 $\lambda\rightarrow$, p. 22.
94/21	B.W. Watson	An introduction to the Fire engine: A C++ toolkit for Finite automata and Regular Expressions.
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 R. Gerth	Abstract Interpretation of Reactive Systems: Abstractions Preserving $\forall$CTL*, $\exists$CTL* and CTL*, p. 28.
94/25	T. Kloks	$K_{1,3}$-free and W_4-free graphs, p. 10.
94/26	R.R. Hoogerwoord	On the foundations of functional programming: a programmer's point of view, p. 54.
94/27	S. Mauw and H. Mulder	Regularity of BPA-Systems is Decidable, p. 14.
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. Ştefanescu	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.
94/32	J.J. Vereijken	Fischer's Protocol in Timed Process Algebra, p. 38.
94/33	T. Laan	A formalization of the Ramified Type Theory, p.40.