Eindhoven University of Technology

Department of Mathematics and Computing Science

Canonical typing and Π-conversion

by

F. Kamareddine and R.P. Nederpelt

94/02

Computing Science Note 94/02
Eindhoven, January 1994

Copies can be ordered from:
Mrs. M. Philips
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

Canonical typing and Π-conversion [*]

Fairouz Kamareddine [†]
Department of Computing Science
17 Lilybank Gardens
University of Glasgow
Glasgow G12 8QQ, Scotland
email: `fairouz@dcs.glasgow.ac.uk`
and

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

December 27, 1993

[*]First of all, we are very grateful to our colleague Bert van Benthem Jutting who has read draft versions of the manuscript, and who has made very useful suggestions. Furthermore, we are grateful for the discussions with Henk Barendregt, Inge Bethke, Tijn Borghuis, Herman Geuvers and Erik Poll, and for the helpful remarks received from them.

[†]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 the summer of 1993. Furthermore, Kamareddine is grateful to the Department of Mathematics and Computer Science, University of Amsterdam, and in particular to Jan Bergstra and Inge Bethke for their hospitality during the preparation of this article.

Abstract

In usual type theory, if a function f is of type $\sigma \rightarrow \sigma'$ and an argument a is of type σ, then the type of fa is immediately given to be σ' and no mention is made of the fact that what has happened is a form of β-conversion. A similar observation holds for the generalized Cartesian product types, $\Pi_{x:\sigma}.\tau$. In fact, many versions of type theory assume that β holds of both types and terms, yet only a few attempt to study the theory where terms and types are really treated equally and where β-conversion is used for both.

A unified treatment however, of types and terms is becoming indispensible especially in the approaches which try to generalise many systems under a unique one. For example, [Barendregt 91] provides the Barendregt cube and the Pure Type Systems (PTSs) which are a generalisation of many type theories. Yet even such a generalisation does not use β-conversion for both types and terms. This is unattractive, in a calculus where types have the same syntax as terms (such as the calculi of the cube or the PTSs). For example, in those systems, even though compatibility holds for the typing of abstraction, it does not hold for the typing of application. That is, even though

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

holds, the following does not hold:

$$M : N \Rightarrow MP : NP.$$

Based on this observation, we present a λ-calculus in which the conversion rules apply to types as well as terms. Abstraction and application, moreover, range over both types and terms. We extend the calculus with a canonical type operator τ in order to associate types to terms. The type of fa will then be Fa, where F is the type of f and the statement $\Gamma \vdash t : \sigma$ from usual type theory is split in two statements in our system: $\Gamma \vdash t$ and $\tau(\Gamma, t) = \sigma$. Such a splitting enables us to discuss the two questions of the typability of a term and of what is its type separately. Again we believe that this splitting is important and should be usually considered.

As a demonstration of what we can do with our calculus, we interpret Church's $\lambda_{\rightarrow}$ in our calculus. This enables us to view our approach as an attempt to extend $\lambda_{\rightarrow}$ with a unified treatment for type and term substitution and conversion and at splitting $\Gamma \vdash t : \sigma$ in the two statements: $\Gamma \vdash t$ and $\tau(\Gamma, t) =_\beta \sigma$. Such an approach should eventually be used to deal with the Barendregt cube and the Pure Type Systems.

Keywords: λ-*Calculus, Type Theory, Church Rosser Theorem, Types as Terms,* $\lambda_{\rightarrow}$.

Contents

1 Introduction

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

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

From then till the present day, research on the area has grown and one can find various reformulations of type theories. A taxonomy of type systems has recently been given by Barendregt ([Barendregt 92]). A version of Church's simple theory of types can be found in this taxonomy under the name $\lambda_{\rightarrow}$ or $\lambda_{\rightarrow}$Church. This $\lambda_{\rightarrow}$ has, apart from *type variables*, so-called *arrow-types* of the form $\sigma \rightarrow \sigma'$, for each pair of types σ and σ'. In higher type theories, arrow-types are replaced by dependent products $\Pi_{y:\sigma}.\sigma'$, where the type σ' may contain y as a free variable, and thus may *depend on y*. This means that abstraction can be over types, similarly to the usual abstraction over terms: $\lambda_{y:\sigma}.t$.

But, once we allow abstraction over types, it would be nice to discuss the conversion rules which govern these types. We propose conversion rules which act similarly to those for terms. For example, if t, t' are terms, σ, σ' are types, and if λ and Π are used for abstraction over terms and types respectively, then not only $(\lambda_{y:\sigma}.t)t' \rightarrow_\beta t[y := t']$, but also $(\Pi_{y:\sigma}.\sigma')t' \rightarrow_\beta \sigma'[y := t']$.

This strategy of permitting Π-*application* $(\Pi_{y:\sigma}.\sigma')t'$ in term construction and using an extended version of β-reduction for such a Π-application, however, is not commonly used. Yet, it is desirable. Especially now in the new tradition which attempts to unify and generalise the type systems. See for example the Barendregt cube in [Barendregt 92] and the fine structure of the λ-calculus in [KN 9y].

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

$$\text{If } f : \Pi_{y:\sigma}.\sigma' \text{ and } a : \sigma, \text{ then } fa : \sigma'[y := a] \tag{1}$$

$$\text{If } f : \Pi_{y:\sigma}.\sigma' \text{ and } a : \sigma, \text{ then } fa : (\Pi_{y:\sigma}.\sigma')a \text{ (which } \beta\text{-converts to } fa : \sigma'[y := a]). \tag{2}$$

In fact, it is more elegant and uniform to use the second notation instead of the first one. The formulation of the theories and the proofs becomes easier. Furthermore, with the second notation, one maintains a *compatibility property* for the typing of *all* applications:

$M : N \Rightarrow MP : NP.$

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

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

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

$A : *, b : A, a : A$	$\vdash$	$a : A$	(start)
$A : *, b : A$	$\vdash$	$(\lambda_{a:A}.a) : (\Pi_{a:A}.A)$	(abstraction)
$A : *, b : A$	$\vdash$	$(\lambda_{a:A}.a)b : (\Pi_{a:A}.A)b$	(application)
$A : *, b : A$	$\vdash$	$(\lambda_{a:A}.a)b : A$	(conversion)

It is our belief that it is simpler to treat terms and types in a unified manner. Moreover, such a unified treatment provides a step towards the generalisation of type systems. In fact, such a generalisation is an important topic of research at the present time. For example, Barendregt's taxonomy of type systems in [Barendregt 92] and our generalised system in [NK 94] which accommodates all the systems of the Barendregt cube are attempts at combining all the important results and structures of type systems in a compact and elegant way. As a step towards this compact and elegant way, we believe that conversion should apply to both types and terms. Hence, this paper aims at extending the conversion rules of terms to types.

We start in Section 2 by presenting the calculus $\lambda_{\to\tau}$, being a form of $\lambda_{\to}$ in which terms and types can be treated alike and where types contain abstraction and application rather than being simple as in $\lambda_{\to}$. The Church Rosser theorem is shown in Section 3 to hold for the calculus. In Section 4, we present the technical machinery relevant for contexts and variables. Important notions such as context ordering and the companion term of a context-and-expression pair are introduced and discussed for binding variables in terms and contexts. For substitution purposes, contexts must be restricted to the well-behaved ones but it is shown that this restriction is only cosmetic, in that for any (Γ, M), we can find an α-variant (Γ', M') where Γ' is well-behaved. In Section 5, we introduce the typing operator τ. This operator will find the types of terms within contexts. τ will satisfy most of the desired properties of typing operators, such as weakening, and substitution.

Typing a term however, is not the only important notion. We need to study the type of the term too and to study the well-typedness of the term. In fact, we think that a more elegant notion of typing can be obtained if we split the judgement $\Gamma \vdash t : \sigma$ in two judgements: $\Gamma \vdash t$ and $\tau(\Gamma, t) =_\beta \sigma$ which say that t is well typed and has for type σ. So instead of concentrating on the whole formula $\Gamma \vdash t : \sigma$ at once, we engage ourselves first in showing the well-typedness of t and then in looking for its type. In fact, we believe that this separation is important especially when we move away from the simpler type theories such as $\lambda_{\to}$ to a more involved ones such as those of the systems of the Barendregt cube where types and terms are inter-mingled. In such a case, not only we need to discuss $\Gamma \vdash t$ but also that the type of t is well-typed (or consistent). We use the notion *consistent* instead of *well-typed* in order to cover for both cases when $\Gamma \vdash t$ and $\Gamma \vdash \tau(\Gamma, t)$. For this reason, we introduce in section 6 the notion of consistency of an expression with respect to a context. All terms which are consistent with respect to a context, are typable (via τ) in the context and their types are also consistent. I.e. if $\Gamma \vdash t$ then $\tau(\Gamma, t)$ is defined and $\Gamma \vdash \tau(\Gamma, t)$. Hence, we define $\Gamma \vdash M$ for M being a type as well as a term. Furthermore, all the information about binding and freeness relevant to t and to typing it in context Γ, is present in Γ and in t. So the expression

$\Gamma \vdash t$ can be treated as a term on its right. We believe that separating $\Gamma \vdash t : \sigma$ into $\Gamma \vdash t$ and $\tau(\Gamma, t) =_\beta \sigma$ deserves attention. Moreover, consistency $\vdash$ has all the desirable properties of type theory. For example, Basis Lemma, Generation Lemma, Correctness of Subexpressions, Weakening, Substitution, Context Reduction, Subject Reduction, Unicity and Correctness of Types all hold for consistent expressions.

Hence the calculus presented unifies the treatment of types and terms, while preserving all the important properties, from Church Rosser to subject reduction and type unicity/correctness. To give the reader a feel for the elegance of the approach, we interpret in Section 7 Church's $\lambda_\rightarrow$ in our calculus. The main result is that $\Gamma \vdash_{\lambda_\rightarrow} t : \sigma$ iff $\Gamma \vdash \mathcal{I}(t)$ and $\Gamma \vdash \mathcal{I}(\sigma)$ in $\lambda_{\rightarrow\tau}$, for $\mathcal{I}$ being the interpretation function from $\lambda_\rightarrow$ to $\lambda_{\rightarrow\tau}$. Moreover, $\tau(\Gamma, \mathcal{I}(t)) =_\beta \mathcal{I}(\sigma)$ in $\lambda_{\rightarrow\tau}$. Furthermore, if $\Gamma' \vdash t'$ in $\lambda_{\rightarrow\tau}$ then there are Γ, t and σ in $\lambda_\rightarrow$ such that $\Gamma' \twoheadrightarrow_\beta \mathcal{I}(\Gamma)$, $t' \twoheadrightarrow_\beta \mathcal{I}(t)$, $\tau(\Gamma', t') =_\beta \mathcal{I}(\sigma)$ and $\Gamma \vdash_{\lambda_\rightarrow} t : \sigma$. We believe that our calculus can be used to provide similar conditions for other type systems and it would be interesting to extend these results for the Pure Type Systems. Hence $\lambda_{\rightarrow\tau}$ can be looked at as a system which discusses and generalises conditions of typing in the known type systems.

2 $\lambda_{\rightarrow\tau}$

We assume two kinds of expressions: types and terms. We assume moreover, an infinite set $\mathcal{V}$ of type variables and an infinite set V of term variables. We let $\mathcal{T}$ be the set of types and T be the set of terms and assume two abstraction operators Π and λ. The Π abstracts over types and the λ over terms. Both $\mathcal{T}$ and T are defined as follows:

$\mathcal{V} = \alpha \mid \mathcal{V}'$
$V = x \mid V'$
$\mathcal{T} = \mathcal{V} \mid (\Pi_{V:\mathcal{T}}.\mathcal{T}) \mid (\mathcal{T}T)$
$T = V \mid (\lambda_{V:\mathcal{T}}.T) \mid (TT)$

Note that this definition allows that types are applied to terms.

Examples of types are: α'', $(\Pi_{x:\alpha}.(\alpha'x'))$ and $((\Pi_{x:\alpha}.\alpha')x')$.

Examples of terms are: x''', $(\lambda_{x:\alpha}.(x''x))$ and $((\lambda_{x:\alpha}.x'')x)$.

We often omit brackets conforming to the usual conventions. We use the meta-variable π to range over $\{\lambda, \Pi\}$. We let $\gamma, \gamma_1, \ldots$ range over $\mathcal{V}$ and $y, z, \ldots$ range over V. Also, $\omega, \omega', \ldots$ range over $\mathcal{V} \cup V$ and we assume that $\mathcal{V} \cap V = \emptyset$. We use $\sigma, \sigma', \sigma'', \ldots, \sigma_1, \sigma_2, \ldots$ to range over $\mathcal{T}$ (the types), $t, t', t'', \ldots, t_1, t_2, \ldots$ to range over T (the terms), and let $L, M, N, P, \ldots$ range over $\mathcal{T} \cup T$. We call the elements of $\mathcal{T} \cup T$ *expressions*.

Lemma 2.1 $\mathcal{T} \cap T = \emptyset$

Proof: *Easy.* □

2.1 Variable manipulation

The free and bound variables in an expression M, denoted $FV(M)$ and $BV(M)$ respectively, are defined as follows:

Definition 2.2 *(Free Variables)*

1. $FV(\omega) = \omega$

2. $FV(\pi_{y:\sigma}.M) = FV(\sigma) \cup (FV(M) \setminus \{y\})$

3. $FV(Mt) = FV(M) \cup FV(t)$

Definition 2.3 *(Bound Variables)*

1. $BV(\omega) = \emptyset$

2. $BV(\pi_{y:\sigma}.M) = \{y\} \cup BV(\sigma) \cup BV(M)$

3. $BV(Mt) = BV(M) \cup BV(t)$

Remark 2.4 *Note that* $BV(M) \subseteq V$.

Now we define the type of a bound variable in an expression as follows:

Definition 2.5 *(Type of Bound Variables)*

- *If y occurs free in M, then all its occurrences are bound with type σ in $\pi_{y:\sigma}.M$ where $\pi \equiv \lambda$ if $M \in T$ and $\pi \equiv \Pi$ if $M \in \mathcal{T}$.*
- *If an occurrence of y is bound with type σ in M, then it is also bound with type σ in $\pi_{y':\sigma'}.M$, in Mt, and, in case $M \in T$, in $(M'M)$.*

As is usual in the λ-calculus, and for ease of the proofs that we will carry out, we assume Barendregt's variable convention. That is: names of bound variables will always be chosen such that they differ from the free ones in an expression, so that one wouldn't have $(\pi_{y:\sigma}.y)y$ but instead $(\pi_{z:\sigma}.z)y$. Such a convention is guaranteed via the use of variable renaming and is formally stated as follows:

Notation 2.6 *(Barendregt's Variable Convention)*
For every M, $BV(M) \cap FV(M) = \emptyset$.

Notation 2.7 *(Extended Variable Convention, VC)*
We extend Barendregt's Variable Convention with the following clause: For every M, if λ_y and λ_z occur in M then $y \not\equiv z$.

Remark 2.8 Note that the condition that names of bound variables be distinct is desirable in order to obtain that for a term obeying *VC*, also its subterms obey *VC*. Take for example the term $t \equiv \lambda_{y:\alpha}.\lambda_{y:\alpha'y}.y$. It is certainly the case that $BV(t) \cap FV(t) = \emptyset$, yet y occurs in the free and bound variables of $\lambda_{y:\alpha'y}.y$. Therefore, we impose the condition that names of bound variables be distinct in order to make sure that for every expression we write down (whether it is an expression or a subexpression of another expression), the free variables and the bound variables are disjoint.

It should be further noted that without variable renaming, we could not have VC. Therefore, we identify expressions modulo α-conversion. With VC moreover, we get the following:

Lemma 2.9 *In $\pi_{y:\sigma}.M$, $y \notin FV(\sigma)$.*
Proof: $BV(\pi_{y:\sigma}.M) = \{y\} \cup BV(\sigma) \cup BV(M)$ *and* $FV(\pi_{y:\sigma}.M) = FV(\sigma) \cup (FV(M) \setminus \{y\})$. $y \in BV(\pi_{y:\sigma}.M) \Longrightarrow^{VC} y \notin FV(\pi_{y:\sigma}.M) \Longrightarrow y \notin FV(\sigma) \cup (FV(M) \setminus \{y\}) \Longrightarrow y \notin FV(\sigma)$. □

Remark 2.10 Note here that with the identification of expressions modulo α-conversion, the notion of a bound variable becomes useless.

Notation 2.11 $M \equiv N$ means that M and N are the same expressions or can be obtained from each other by renaming bound variables. For example: $\pi_{y:\sigma}.y \equiv \pi_{z:\sigma}.z$ for z not free in σ. Now, if in clause 5 of Definition 2.12 below, $y \in FV(P)$, then we write $(\pi_{y:\sigma}.M)[\omega := P] \equiv (\pi_{z:\sigma}.M[y := z])[\omega := P]$ for z a fresh variable (note from Lemma 2.9 that $y \notin FV(\sigma)$). With this notation, we follow the lines of Barendregt in [Barendregt 92] in identifying expressions that differ only in the name of bound variables, rather than using α-conversion. That is, the identification is done in our mind rather than on paper.

2.2 Substitution and reduction

We introduce substitution, reduction and conversion by the following definitions:

Definition 2.12 *(Substitution)*
We define $M[\omega := P]$ to be the result of substituting P in M for all free occurrences of ω. In this definition, we assume that $FV(P) \cap BV(M) = \emptyset$; this is consistent with the variable convention. $M[\omega := P]$ is defined by induction as follows:

1. $\omega[\omega := P] \equiv P$
2. $\omega_1[\omega_2 := P] \equiv \omega_1$ *if* $\omega_1 \not\equiv \omega_2$
3. $(Mt)[\omega := P] \equiv M[\omega := P]t[\omega := P]$
4. $(\pi_{y:\sigma}.M)[y := P] \equiv \pi_{y:\sigma}.M$[1]
5. $(\pi_{y:\sigma}.M)[\omega := P] \equiv \pi_{y:\sigma[\omega:=P]}.M[\omega := P]$ *if* $y \not\equiv \omega$

Lemma 2.13 *(Substitution in Terms and Types)*

1. *If* $y \in V$, $N \in T$ *then*
   - $M \in T \Longrightarrow M[y := N] \in T$
   - $M \in \mathcal{T} \Longrightarrow M[y := N] \in \mathcal{T}$
2. *If* $\gamma \in \mathcal{V}$, $N \in \mathcal{T}$ *then*
   - $M \in T \Longrightarrow M[\gamma := N] \in T$
   - $M \in \mathcal{T} \Longrightarrow M[\gamma := N] \in \mathcal{T}$

Proof: *Both by simultaneous induction on the structure of M.* □

Lemma 2.14 *If $M, M_1, M_2 \in \mathcal{T} \cup T$, $\omega \not\equiv \omega'$ and $\omega \notin FV(M_2)$ then*

$$M[\omega := M_1][\omega' := M_2] \equiv M[\omega' := M_2][\omega := M_1[\omega' := M_2]].$$

Proof: *This is a corollary of Lemma 3.3 below.* □

[1]Note that y could not be free in σ according to VC, by Lemma 2.9.

Definition 2.15 *(One step Reduction $\rightarrow_\beta$, a relation on $\mathcal{T} \cup T$)*
One step reduction $\rightarrow_\beta$ on $\mathcal{T} \cup T$ is the least relation closed under the β-rule (rule 1, below) and the compatibility conditions (rules 2,3 and 4 below).

1. $(\pi_{y:\sigma}.M)t \rightarrow_\beta M[y := t]$
2. *If* $t \rightarrow_\beta t'$ *then* $Mt \rightarrow_\beta Mt'$
3. *If* $M \rightarrow_\beta N$ *then* $Mt \rightarrow_\beta Nt$ *and* $\pi_{y:\sigma}.M \rightarrow_\beta \pi_{y:\sigma}.N$
4. *If* $\sigma \rightarrow_\beta \sigma'$ *then* $\pi_{y:\sigma}.M \rightarrow_\beta \pi_{y:\sigma'}.M$

Definition 2.16 *(Reduction $\twoheadrightarrow_\beta$, a relation on $\mathcal{T} \cup T$)*
Reduction$\twoheadrightarrow_\beta$ on $\mathcal{T} \cup T$ is the reflexive and transitive closure of $\rightarrow_\beta$. That is, $\twoheadrightarrow_\beta$ is defined by the following rules:

1. $M \twoheadrightarrow_\beta M$
2. *If* $M \rightarrow_\beta N$ *then* $M \twoheadrightarrow_\beta N$
3. *If* $M \twoheadrightarrow_\beta N$ *and* $N \twoheadrightarrow_\beta L$ *then* $M \twoheadrightarrow_\beta L$

Definition 2.17 *(Conversion $=_\beta$, a relation on $\mathcal{T} \cup T$)*
Conversion $=_\beta$ on $\mathcal{T} \cup T$ is the least equivalence relation closed under $\twoheadrightarrow_\beta$. That is:

1. *If* $M \twoheadrightarrow_\beta N$ *then* $M =_\beta N$
2. *If* $M =_\beta N$ *then* $N =_\beta M$
3. *If* $M =_\beta N$ *and* $N =_\beta L$ *then* $M =_\beta L$

Lemma 2.18
Let $\succ$ be $\rightarrow_\beta$ or $\twoheadrightarrow_\beta$ or $=_\beta$. Now, if $M \in T$ (respectively $M \in \mathcal{T}$) and if $M \succ N$ then $N \in T$ (respectively $N \in \mathcal{T}$).
Proof: *Use Lemma 2.13 and induction.* □

Lemma 2.19 *($\rightarrow_\beta$-substitution lemma on $\mathcal{T} \cup T$)*
For $M, N \in \mathcal{T} \cup T, z \in V$ and $t' \in T$, if $M \rightarrow_\beta N$, then $M[z := t'] \rightarrow_\beta N[z := t']$.
Proof: *This is a corollary of Lemma 3.5 and Lemma 3.6 below.* □

Corollary 2.20 *($\twoheadrightarrow_\beta$-substitution lemma on $\mathcal{T} \cup T$)*
For $M, N \in \mathcal{T} \cup T, z \in V$ and $t' \in T$, if $M \twoheadrightarrow_\beta N$, then $M[z := t'] \twoheadrightarrow_\beta N[z := t']$.
Proof: *By induction on $\twoheadrightarrow_\beta$ using Lemma 2.19.* □

Definition 2.21 *(β-redexes, β-nf)*

- *An expression of the form $(\pi_{y:\sigma}.M)t$ is called a β-redex.*
- *If an expression M has no β-redexes as a subexpression then M is said to be in β-nf.*
- *If $t =_\beta t'$ where t' is in β-nf, then t is said to have a β-nf.*

3 The Church Rosser Theorem

To prove the Church Rosser Theorem (in short CR theorem), we shall follow the method presented in [Barendregt 84] working with types and terms alike. That is, even though we use a similar strategy to that of [Barendregt 84] to prove the CR theorem, the details will extend all the notions of reductions, substitution and all the proofs in order to treat types as well as terms in a unified manner. We start by extending $\mathcal{T}$ and T to the following:

$\underline{\mathcal{T}} = \mathcal{V} \mid (\Pi_{y:\underline{\mathcal{T}}}.\underline{\mathcal{T}}) \mid (\underline{\mathcal{T}}\,\underline{T}) \mid (\underline{\Pi}_{y:\underline{\mathcal{T}}}.\underline{\mathcal{T}})\underline{T}$

$\underline{T} = V \mid (\lambda_{y:\underline{\mathcal{T}}}.\underline{T}) \mid (\underline{T}\,\underline{T}) \mid (\underline{\lambda}_{y:\underline{\mathcal{T}}}.\underline{T})\underline{T}$

In this section, M, N and P range over $\underline{\mathcal{T}} \cup \underline{T}$. Moreover, $\sigma, \sigma', \ldots$ range over $\underline{\mathcal{T}}$ and $t, t', \ldots$ range over $\underline{T}$. Furthermore, we use $\underline{\pi}$ to range over $\{\underline{\lambda}, \underline{\Pi}\}$.

Remark 3.1 Note that when we write an expression M this will never indicate $\underline{\pi}_{y:\sigma}.M'$ for some M', even if M occurs with an argument N in MN.

We extend the definition of free and bound variables by adding to Definition 2.2, the first clause below and to Definition 2.3, the second clause below (recall here however remark 2.10).

$$\begin{array}{rcl} FV((\underline{\pi}_{y:\sigma}.M)t) & = & FV(\sigma) \cup (FV(M) \setminus \{y\}) \cup FV(t) \\ BV((\underline{\pi}_{y:\sigma}.M)t) & = & \{y\} \cup BV(\sigma) \cup BV(M) \cup BV(t) \end{array}$$

We still assume moreover the variable convention for $\underline{\mathcal{T}} \cup \underline{T}$ and consider expressions to be equivalent up to variable renaming.

3.1 Substitution and Reduction in $\underline{\mathcal{T}} \cup \underline{T}$

Substitution is exactly as in Definition 2.12 except that we add the following:

$((\underline{\pi}_{y:\sigma}.M)t')[\omega := P] = (\underline{\pi}_{y:\sigma[\omega:=P]}.M[\omega := P])(t'[\omega := P])$ if $y \not\equiv \omega$

$((\underline{\pi}_{y:\sigma}.M)t')[y := P] = (\underline{\pi}_{y:\sigma}.M)t'$.[2]

Now a similar version of Lemma 2.13 holds for $\underline{\mathcal{T}} \cup \underline{T}$. That is,

Lemma 3.2 *(Substitution in Terms and Types)*

1. *If $y \in V$, $N \in \underline{T}$ then*
   - $M \in \underline{T} \Longrightarrow M[y := N] \in \underline{T}$
   - $M \in \underline{\mathcal{T}} \Longrightarrow M[y := N] \in \underline{\mathcal{T}}$
2. *If $\gamma \in \mathcal{V}$, $N \in \underline{\mathcal{T}}$ then*
   - $M \in \underline{T} \Longrightarrow M[\gamma := N] \in \underline{T}$
   - $M \in \underline{\mathcal{T}} \Longrightarrow M[\gamma := N] \in \underline{\mathcal{T}}$

Proof: *Both by simultaneous induction on the structure of M.* □

Lemma 3.3 *If $M, M_1, M_2 \in \underline{\mathcal{T}} \cup \underline{T}$, $\omega \not\equiv \omega'$ and $\omega \notin FV(M_2)$ then*

$$M[\omega := M_1][\omega' := M_2] \equiv M[\omega' := M_2][\omega := M_1[\omega' := M_2]].$$

Proof: *By induction on the length of terms and types in $\underline{\mathcal{T}} \cup \underline{T}$.*

[2]The second clause is in accordance with the variable convention as $y \notin FV(t')$.

- *$M \equiv \omega$ then lhs $\equiv M_1[\omega' := M_2] \equiv$ rhs.*
- *$M \equiv \omega'$ then lhs $\equiv M_2 \equiv$ rhs as $\omega \notin FV(M_2)$.*
- *$M \equiv \omega''$ and $\omega'' \not\equiv \omega$ and $\omega'' \not\equiv \omega'$ then lhs $\equiv$ rhs $\equiv \omega''$.*
- *Assume the property holds for M, t then obviously it holds for Mt, i.e. $(Mt)[\omega := M_1][\omega' := M_2] \equiv (Mt)[\omega' := M_2][\omega := M_1[\omega' := M_2]]$*
- *Assume the property holds for M, then let us show it holds for $\pi_{y:\sigma}.M$.*
  - *case $\omega \not\equiv y$ and $\omega' \not\equiv y$ then*
    $(\pi_{y:\sigma}.M)[\omega := M_1][\omega' := M_2] \equiv$
    $\pi_{y:\sigma[\omega:=M_1][\omega':=M_2]}.M[\omega := M_1][\omega' := M_2] \equiv^{IH}$
    $\pi_{y:\sigma[\omega':=M_2][\omega:=M_1[\omega':=M_2]]}.M[\omega' := M_2][\omega := M_1[\omega' := M_2]] \equiv$
    $(\pi_{y:\sigma}.M)[\omega' := M_2][\omega := M_1[\omega' := M_2]]$
  - *case $\omega \equiv y$ then*
    $(\pi_{y:\sigma}.M)[y := M_1][\omega' := M_2] \equiv$
    $\pi_{y:\sigma[\omega':=M_2]}.M[\omega' := M_2]$ *and*
    $(\pi_{y:\sigma}.M)[\omega' := M_2][y := M_1[\omega' := M_2]] \equiv$
    $\pi_{y:\sigma[\omega':=M_2]}.M[\omega' := M_2]$.
  - *case $\omega' \equiv y$ and $y \notin FV(M_1)$ then*
    $(\pi_{y:\sigma}.M)[\omega := M_1][y := M_2] \equiv$
    $\pi_{y:\sigma[\omega:=M_1]}.M[\omega := M_1]$
    Moreover, $(\pi_{y:\sigma}.M)[y := M_2][\omega := M_1[y := M_2]] \equiv$
    $(\pi_{y:\sigma}.M)[\omega := M_1[y := M_2]] \equiv$
    $\pi_{y:\sigma[\omega:=M_1]}.M[\omega := M_1]$
  - *case $\omega' \equiv y$ and $y \in FV(M_1)$ then* $(\pi_{y:\sigma}.M)[\omega := M_1][y := M_2] \equiv$
    $(\pi_{z:\sigma}.M[y := z])[\omega := M_1][y := M_2]$ *(for fresh z)* $\equiv$
    $(\pi_{z:\sigma[\omega:=M_1]}.M[y := z][\omega := M_1])[y := M_2] \equiv$
    $\pi_{z:\sigma[\omega:=M_1][y:=M_2]}.M[y := z][\omega := M_1][y := M_2] \equiv^{IH}$
    $\pi_{z:\sigma[y:=M_2][\omega:=M_1[y:=M_2]]}.M[y := z][y := M_2][\omega := M_1[y := M_2]] \equiv$
    (since $y \notin FV(\sigma)$ by VC and $y \notin FV(M[y := z])$)
    $\pi_{z:\sigma[\omega:=M_1[y:=M_2]]}.M[y := z][\omega := M_1[y := M_2]]$.
    Moreover, $(\pi_{y:\sigma}.M)[y := M_2][\omega := M_1[y := M_2]] \equiv$
    $(\pi_{z':\sigma}.M[y := z'])[y := M_2][\omega := M_1[y := M_2]]$ *(for fresh z')* $\equiv$
    $(\pi_{z':\sigma[y:=M_2]}.M[y := z'][y := M_2])[\omega := M_1[y := M_2]] \equiv$
    (since $y \notin FV(\sigma)$ and $y \notin FV(M[y := z'])$)
    $(\pi_{z':\sigma}.M[y := z'])[\omega := M_1[y := M_2]] \equiv$
    $\pi_{z':\sigma[\omega:=M_1[y:=M_2]]}.M[y := z'][\omega := M_1[y := M_2]]$
- *For $((\underline{\pi}_{y:\sigma}.M)t)[\omega := M_1][\omega' := M_2]$ use a similar proof.*

□

We extend Definitions 2.15 and 2.16 to the following:

Definition 3.4 *(Extended Reduction $\to_{\underline{\beta}}$ and $\twoheadrightarrow_{\underline{\beta}}$)*

1. $(\pi_{y:\sigma}.M)t \to_{\underline{\beta}} M[y := t]$
2. $(\underline{\pi}_{y:\sigma}.M)t \to_{\underline{\beta}} M[y := t]$
3. *If* $t \to_{\underline{\beta}} t'$ *then* $(\underline{\pi}_{y:\sigma}.M)t \to_{\underline{\beta}} (\underline{\pi}_{y:\sigma}.M)t'$ *and* $Pt \to_{\underline{\beta}} Pt'$
4. *If* $M \to_{\underline{\beta}} N$ *then*
   $Mt \to_{\underline{\beta}} Nt$, $\pi_{y:\sigma}.M \to_{\underline{\beta}} \pi_{y:\sigma}.N$ *and* $(\underline{\pi}_{y:\sigma}.M)t \to_{\underline{\beta}} (\underline{\pi}_{y:\sigma}.N)t$
5. *If* $\sigma \to_{\underline{\beta}} \sigma'$ *then* $\pi_{y:\sigma}.P \to_{\underline{\beta}} \pi_{y:\sigma'}.P$ *and* $(\underline{\pi}_{y:\sigma}.P)t \to_{\underline{\beta}} (\underline{\pi}_{y:\sigma'}.P)t$
6. $\twoheadrightarrow_{\underline{\beta}}$ *is the transitive and reflexive closure of* $\to_{\underline{\beta}}$.

The relation $\to_{\underline{\beta}}$ on $\underline{\mathcal{T}} \cup \underline{T}$ is indeed an extension of the relation $\to_\beta$ on $\mathcal{T} \cup T$:

Lemma 3.5
Let $M, N \in \mathcal{T} \cup T$. *Then:* $M \to_\beta N$ *iff* $M \to_{\underline{\beta}} N$.
Proof: *By induction on* $M \to_\beta N$ *or* $M \to_{\underline{\beta}} N$, *respectively.* □

Lemma 3.6 *($\to_{\underline{\beta}}$-substitution lemma on $\underline{\mathcal{T}} \cup \underline{T}$)*
For $M, N \in \underline{\mathcal{T}} \cup \underline{T}$, $z \in V$ *and* $t' \in \underline{T}$, *if* $M \to_{\underline{\beta}} N$, *then* $M[z := t'] \to_{\underline{\beta}} N[z := t']$.
Proof: *By induction on* $M \to_{\underline{\beta}} N$.

- *Case* $(\pi_{y:\sigma}.M)t \to_{\underline{\beta}} M[y := t]$,
  1. *Case* $z \not\equiv y$:
     $((\pi_{y:\sigma}.M)t)[z := t'] \equiv (\pi_{y:\sigma[z:=t']}.M[z := t'])t[z := t'] \to_{\underline{\beta}}$
     $M[z := t'][y := t[z := t']] \equiv M[y := t][z := t']$ *by Lemma 3.3, since* $y \notin FV(t')$ *by VC.*
  2. *Case* $z \equiv y$:
     $((\pi_{y:\sigma}.M)t)[y := t'] \equiv (\pi_{y:\sigma}.M)t \to_{\underline{\beta}}$
     $M[y := t] \equiv M[y := t][y := t']$ *since* $y \notin FV(t)$ *by VC.*
- *Case* $(\underline{\pi}_{y:\sigma}.M)t \to_{\underline{\beta}} M[y := t]$ *is similar to the above case.*
- *The other cases are easy.*

□

Corollary 3.7 *($\twoheadrightarrow_{\underline{\beta}}$-substitution lemma on $\underline{\mathcal{T}} \cup \underline{T}$)*
For $M, N \in \underline{\mathcal{T}} \cup \underline{T}$, $z \in V$ *and* $t' \in \underline{T}$, *if* $M \twoheadrightarrow_{\underline{\beta}} N$, *then* $M[z := t'] \twoheadrightarrow_{\underline{\beta}} N[z := t']$.
Proof: *By induction on* $\to_{\underline{\beta}}$ *using Lemma 3.6*

□

3.2 The relations between $\underline{\mathcal{T}} \cup \underline{T}$ and $\mathcal{T} \cup T$

Definition 3.8 *The map* $|\ |: \underline{\mathcal{T}} \cup \underline{T} \longrightarrow \mathcal{T} \cup T$ *is defined as the erasing of all underlinings.*

Definition 3.9 $\phi : \underline{\mathcal{T}} \cup \underline{T} \longrightarrow \mathcal{T} \cup T$ *is defined as follows:*

1. $\phi(\omega) \equiv \omega$
2. $\phi(Mt) \equiv \phi(M)\phi(t)$[3]

[3]Note here that M is not $\underline{\pi}_{y:\sigma}.N$.

3. $\phi((\underline{\pi}_{y:\sigma}.M)t) \equiv \phi(M)[y := \phi(t)]$

4. $\phi(\pi_{y:\sigma}.M) \equiv \pi_{y:\phi(\sigma)}.\phi(M)$

We denote $\mid M \mid \equiv N$ and $\phi(M) \equiv N$ by $M \to^{||} N$ and $M \to^{\phi} N$ respectively.

Lemma 3.10 *For every $M \in \underline{\mathcal{T}} \cup \underline{T}$, $FV(\phi(M)) \subseteq FV(M)$.*
Proof: *By induction on the structure of M.*

□

In what follows, read dashed lines as a quest for existence, or a proof and non dashed lines as hypotheses.

Lemma 3.11 *For every $M, N \in \mathcal{T} \cup T$ and $M' \in \underline{\mathcal{T}} \cup \underline{T}$, if $M' \to^{||} M$ and $M \twoheadrightarrow_\beta N$ then $(\exists N')[M' \twoheadrightarrow_\beta N' \wedge N' \to^{||} N]$. This is pictured as follows:*

$$\begin{tikzcd}
M' \arrow[r, two heads, dashed, "\underline{\beta}"'] \arrow[d, "||"'] & N' \arrow[d, dashed, "||"] \\
M \arrow[r, two heads, "\beta"'] & N
\end{tikzcd}$$

Proof: *Clearly this property holds for $\to_\beta$:*
If $M \to_\beta N$ is the result of contracting in M a redex obtaining N, then N' can be obtained from M' by contracting the corresponding redex in M'. Now we prove by induction on the definition of $\twoheadrightarrow_\beta$ that it holds for $\twoheadrightarrow_\beta$.

- *If $M \twoheadrightarrow_\beta M$ obvious.*
- *If $M \twoheadrightarrow_\beta N$ comes from $M \to_\beta N$ then from above.*
- *If $M \twoheadrightarrow_\beta N$ comes from $M \twoheadrightarrow_\beta N_1$ and $N_1 \twoheadrightarrow_\beta N$ and property holds for $M \twoheadrightarrow_\beta N_1$ and $N_1 \twoheadrightarrow_\beta N$ then:*

$$\begin{tikzcd}
M' \arrow[r, two heads, dashed, "\underline{\beta}"'] \arrow[d, "||"'] & N'_1 \arrow[r, two heads, dashed, "\underline{\beta}"'] \arrow[d, dashed, "||"'] & N' \arrow[d, dashed, "||"] \\
M \arrow[r, two heads, "\beta"'] & N_1 \arrow[r, two heads, "\beta"'] & N
\end{tikzcd}$$

□

Lemma 3.12 *For all ω, M, and P in $\underline{\mathcal{T}} \cup \underline{T}$ with $FV(P) \cap BV(M) = \emptyset$, we have $\phi(M)[\omega := \phi(P)] \equiv \phi(M[\omega := P])$.*
Proof: *By induction on M*

- *Case ω:*
  $\omega[\omega := P] \equiv P$ *so* $\phi(\omega[\omega := P]) \equiv \phi(P)$
  and $\phi(\omega)[\omega := \phi(P)] \equiv \omega[\omega := \phi(P)] \equiv \phi(P)$
- *Case $\omega_1 \not\equiv \omega$:*
  $\omega_1[\omega := P] \equiv \omega_1$ *so*
  $\phi(\omega_1[\omega := P]) \equiv \phi(\omega_1) \equiv \phi(\omega_1)[\omega := \phi(P)]$

- *Case Mt:*
  $Mt[\omega := P] \equiv M[\omega := P]t[\omega := P]$ *where IH holds for M, t. Hence*
  $\phi(Mt)[\omega := \phi(P)] \equiv^{\phi}$
  $(\phi(M)\phi(t))[\omega := \phi(P)] \equiv^{sub}$
  $(\phi(M)[\omega := \phi(P)])(\phi(t)[\omega := \phi(P)]) \equiv^{IH}$
  $\phi(M[\omega := P])\phi(t[\omega := P]) \equiv^{\phi}$
  $\phi(M[\omega := P]t[\omega := P]) \equiv^{sub}$
  $\phi(Mt[\omega := P])$

- *Case $\pi_{y:\sigma}.M$ and $\omega \equiv y$:*
  $(\pi_{y:\sigma}.M)[y := P] \equiv \pi_{y:\sigma}.M$ *so*
  $\phi((\pi_{y:\sigma}.M)[y := P]) \equiv \phi(\pi_{y:\sigma}.M) \equiv$
  $\pi_{y:\phi(\sigma)}.\phi(M) \equiv$
  $(\pi_{y:\phi(\sigma)}.\phi(M))[y := \phi(P)] \equiv$
  $\phi(\pi_{y:\sigma}.M)[y := \phi(P)]$

- *Case $\pi_{y:\sigma}.M$ and $\omega \not\equiv y$:*
  $(\pi_{y:\sigma}.M)[\omega := P] \equiv \pi_{y:\sigma[\omega:=P]}.M[\omega := P]$ *so*
  $\phi(\pi_{y:\sigma}.M)[\omega := \phi(P)] \equiv^{\phi}$
  $(\pi_{y:\phi(\sigma)}.\phi(M))[\omega := \phi(P)] \equiv^{sub}$
  $\pi_{y:\phi(\sigma)[\omega:=\phi(P)]}.(\phi(M)[\omega := \phi(P)]) \equiv^{IH}$
  $\pi_{y:\phi(\sigma[\omega:=P])}.\phi(M[\omega := P]) \equiv^{\phi}$
  $\phi(\pi_{y:\sigma[\omega:=P]}.M[\omega := P]) \equiv$
  $\phi((\pi_{y:\sigma}.M)[\omega := P])$

- *Case $(\underline{\pi}_{y:\sigma}.M)t'$ and $\omega \equiv y$:*
  $((\underline{\pi}_{y:\sigma}.M)t')[y := P] \equiv (\underline{\pi}_{y:\sigma}.M)t'$, *so*

  - $\phi((\underline{\pi}_{y:\sigma}.M)t')[y := \phi(P)] \equiv^{\phi}$
    $\phi(M)[y := \phi(t')][y := \phi(P)] \equiv$
    $\phi(M)[y := \phi(t')]$ *because $y \notin FV(t')$ due to the variable convention.*
  - $\phi(((\underline{\pi}_{y:\sigma}.M)t')[y := P]) \equiv$
    $\phi((\underline{\pi}_{y:\sigma}.M)t') \equiv^{\phi}$
    $\phi(M)[y := \phi(t')]$.

- *Case $(\underline{\pi}_{y:\sigma}.M)t'$ and $\omega \not\equiv y$:*
  $((\underline{\pi}_{y:\sigma}.M)t')[\omega := P] \equiv (\underline{\pi}_{y:\sigma[\omega:=P]}.M[\omega := P])(t'[\omega := P])$ *so*
  $\phi((\underline{\pi}_{y:\sigma}.M)t')[\omega := \phi(P)] \equiv^{\phi}$
  $\phi(M)[y := \phi(t')][\omega := \phi(P)] \equiv^{Lemmas\ 3.3\ and\ 3.10}$
  (note that $y \notin FV(P)$ since $FV(P) \cap BV((\underline{\pi}_{y:\sigma}.M)t') = \emptyset$)
  $\phi(M)[\omega := \phi(P)][y := \phi(t')[\omega := \phi(P)]] \equiv$ *by IH*
  $\phi(M[\omega := P])[y := \phi(t'[\omega := P])] \equiv^{\phi}$
  $\phi((\underline{\pi}_{y:\sigma[\omega:=P]}.M[\omega := P])(t'[\omega := P])) \equiv$
  $\phi(((\underline{\pi}_{y:\sigma}.M)t')[\omega := P])$.

□

Lemma 3.13 *For $M, M' \in \underline{\mathcal{T}} \cup \mathcal{T}$, if $M \rightarrow_{\underline{\beta}} M'$ then $\phi(M) \rightarrow_{\beta} \phi(M')$. That is:*

$$\begin{tikzcd}
M \arrow[rr, two heads, "\underline{\beta}"'] \arrow[d, "\phi"'] & & M' \arrow[d, "\phi"] \\
\phi(M) \arrow[rr, dashed, two heads, "\beta"'] & & \phi(M')
\end{tikzcd}$$

Proof: *Induction on $M \twoheadrightarrow_{\underline{\beta}} M'$:*

- $(\pi_{y:\sigma}.M)t \to_{\underline{\beta}} M[y := t]$ *then*
  $\phi((\pi_{y:\sigma}.M)t) \equiv \phi((\pi_{y:\sigma}.M))\phi(t) \equiv (\pi_{y:\phi(\sigma)}.\phi(M))\phi(t) \to_\beta$
  $\phi(M)[y := \phi(t)] \equiv^{VC \Rightarrow FV(t) \cap BV(M) = \emptyset, Lemma\ 3.12} \phi(M[y := t])$

- $(\underline{\pi}_{y:\sigma}.M)t \to_{\underline{\beta}} M[y := t]$ *then*
  $\phi((\underline{\pi}_{y:\sigma}.M)t) \to_\beta \phi(M)[y := \phi(t)] \equiv^{VC \Rightarrow FV(t) \cap BV(M) = \emptyset, Lemma\ 3.12} \phi(M[y := t])$

- *If* $t \to_{\underline{\beta}} t'$ *implies* $\phi(t) \to_\beta \phi(t')$ *then*
  - $\phi(Pt) \equiv \phi(P)\phi(t) \to_\beta \phi(P)\phi(t') \equiv \phi(Pt')$
  - $\phi((\underline{\pi}_{y:\sigma}.P)t) \to_\beta \phi((\underline{\pi}_{y:\sigma}.P)t')$

- *If* $M \to_{\underline{\beta}} N$ *implies* $\phi(M) \to_\beta \phi(N)$ *then*
  - $\phi(Mt) \to_\beta \phi(Nt)$
  - $\phi(\pi_{y:\sigma}.M) \to_\beta \phi(\pi_{y:\sigma}.N)$
  - $\phi((\underline{\pi}_{y:\sigma}.M)t) \to_\beta \phi((\underline{\pi}_{y:\sigma}.N)t)$

- *If* $\sigma \to_{\underline{\beta}} \sigma'$ *implies* $\phi(\sigma) \to_\beta \phi(\sigma')$ *then*
  - $\phi(\pi_{y:\sigma}.P) \to_\beta \phi(\pi_{y:\sigma'}.P)$
  - $\phi((\underline{\pi}_{y:\sigma}.P)t) \to_\beta \phi((\underline{\pi}_{y:\sigma'}.P)t)$

- *If* $M \twoheadrightarrow_{\underline{\beta}} M$ *then obviously* $\phi(M) \twoheadrightarrow_\beta \phi(M)$

- *If* $M \twoheadrightarrow_{\underline{\beta}} M'$ *comes from* $M \to_{\underline{\beta}} M'$ *then* $\phi(M) \twoheadrightarrow_\beta \phi(M')$

- *If* $M \twoheadrightarrow_{\underline{\beta}} M'$ *comes from* $M \twoheadrightarrow_{\underline{\beta}} M''$ *and* $M'' \twoheadrightarrow_{\underline{\beta}} M'$ *and the induction hypothesis holds for* M, M'' *and* M'', M' *then* $\phi(M) \twoheadrightarrow_\beta \phi(M')$

$\square$

Lemma 3.14 *For* $M \in \underline{\mathcal{T}} \cup \underline{T}$ *and* $M_1, M_2 \in \mathcal{T} \cup T$, *if* $M \to^{||} M_1$ *and* $M \to^{\phi} M_2$ *then* $M_1 \twoheadrightarrow_\beta M_2$. *That is:*

$$\begin{tikzcd}
 & M \arrow[dl, "||"'] \arrow[dr, "\phi"] & \\
M_1 \arrow[rr, dashed, two heads, "\beta"'] & & M_2
\end{tikzcd}$$

Proof: *By induction on $M \in \underline{\mathcal{T}} \cup \underline{T}$*

- *If $M \equiv \omega$ obvious.*
- *If $M \equiv M't$ where*

$$\begin{tikzcd}
& M' \arrow[dl, "||"'] \arrow[dr, "\phi"] & \\
M'_1 \arrow[rr, dashed, two heads, "\beta"'] & & M'_2
\end{tikzcd}
\qquad
\begin{tikzcd}
& t \arrow[dl, "||"'] \arrow[dr, "\phi"] & \\
t_1 \arrow[rr, dashed, two heads, "\beta"'] & & t_2
\end{tikzcd}$$

then $\mid M't \mid \equiv \mid M' \mid\mid t \mid \equiv M'_1t_1$ and $\phi(M't) \equiv \phi(M')\phi(t) \equiv M'_2t_2$. Hence

$$\begin{tikzcd}
& M't \arrow[dl, "||"'] \arrow[dr, "\phi"] & \\
M'_1t_1 \arrow[rr, dashed, two heads, "\beta"'] & & M'_2t_2
\end{tikzcd}$$

- *If $M \equiv \pi_{y:\sigma}.M'$ where*

$$\begin{tikzcd}
& M' \arrow[dl, "||"'] \arrow[dr, "\phi"] & \\
M'_1 \arrow[rr, dashed, two heads, "\beta"'] & & M'_2
\end{tikzcd}
\qquad
\begin{tikzcd}
& \sigma \arrow[dl, "||"'] \arrow[dr, "\phi"] & \\
\sigma_1 \arrow[rr, dashed, two heads, "\beta"'] & & \sigma_2
\end{tikzcd}$$

then

$$\begin{tikzcd}
& \pi_{y:\sigma}.M' \arrow[dl, "||"'] \arrow[dr, "\phi"] & \\
\pi_{y:\sigma_1}.M'_1 \arrow[rr, dashed, two heads, "\beta"'] & & \pi_{y:\sigma_2}.M'_2
\end{tikzcd}$$

- *If $M \equiv (\underline{\pi}_{y:\sigma}.M')t$ where*

$$\begin{tikzcd}
& M' \arrow[dl, "||"'] \arrow[dr, "\phi"] & \\
M'_1 \arrow[rr, dashed, two heads, "\beta"'] & & M'_2
\end{tikzcd}
\qquad
\begin{tikzcd}
& \sigma \arrow[dl, "||"'] \arrow[dr, "\phi"] & \\
\sigma_1 \arrow[rr, dashed, two heads, "\beta"'] & & \sigma_2
\end{tikzcd}
\qquad
\begin{tikzcd}
& t \arrow[dl, "||"'] \arrow[dr, "\phi"] & \\
t_1 \arrow[rr, dashed, two heads, "\beta"'] & & t_2
\end{tikzcd}$$

then $(\pi_{y:\sigma_1}.M_1')t_1 \twoheadrightarrow_\beta (\pi_{y:\sigma_2}.M_2')t_2 \twoheadrightarrow_\beta M_2'[y := t_2]$ *and hence*

$$\begin{tikzcd}
& (\underline{\pi}_{y:\sigma}.M')t \arrow[dl, "||"] \arrow[dr, "\phi"] & \\
(\pi_{y:\sigma_1}.M_1')t_1 \arrow[rr, dashed, two heads, "\beta"'] & & M_2'[y := t_2]
\end{tikzcd}$$

□

3.3 The theorem and its corollaries

First we start by proving the strip lemma which will be used in the proof of the CR theorem. Then we show the theorem and three of its corollaries.

Lemma 3.15 *(Strip Lemma) For* $M, M_1, M_2, M_3 \in \mathcal{T} \cup T$ *we have:*

$$\begin{tikzcd}
M \arrow[r, two heads, "\beta"'] \arrow[d, "\beta"'] & M_2 \arrow[d, dashed, "\beta"] \\
M_1 \arrow[r, dashed, two heads, "\beta"'] & M_3
\end{tikzcd}$$

Proof: *Let* M_1 *be the result of contracting* $R \equiv (\pi_{y:\sigma}.M')M''$ *in* M. *Let* $M_1' \in \underline{\mathcal{T}} \cup \underline{T}$ *be obtained from* M *by replacing* R *by* $R' \equiv (\underline{\pi}_{y:\sigma}.M')M''$. *Then* $\mid M_1' \mid \equiv M$ *and* $\phi(M_1') \equiv M_1$. *By Lemmas 3.11, 3.13 and 3.14 we have* M_2', M_3 $(\equiv \phi(M_2'))$ *and the following diagram:*

$$\begin{tikzcd}
M \arrow[rr, two heads, "\beta"'] \arrow[dd, "\beta"'] & & M_2 \arrow[dd, dashed, "\beta"'] & \\
& M_1' \arrow[ul, "||"] \arrow[dl, "\phi"] \arrow[rr, dashed, two heads, "\beta"] & & M_2' \arrow[ul, "||"'] \arrow[dl, "\phi"] \\
M_1 \arrow[rr, dashed, two heads, "\beta"'] & & M_3 &
\end{tikzcd}$$

□

Theorem 3.16 *(The Church Rosser Theorem)*
For $M, N_1, N_2 \in \mathcal{T} \cup T$, *if* $M \twoheadrightarrow_\beta N_1$ *and* $M \twoheadrightarrow_\beta N_2$ *then there exists* $N_3 \in \mathcal{T} \cup T$ *such that* $N_1 \twoheadrightarrow_\beta N_3$ *and* $N_2 \twoheadrightarrow_\beta N_3$

Proof: $M \twoheadrightarrow_\beta N_1$ *then* $M \equiv M_0 \rightarrow_\beta M_1 \rightarrow_\beta M_2 \rightarrow_\beta \ldots \rightarrow_\beta M_n \equiv N_1$. *Hence from Lemma 3.15 we have:*

$$\begin{tikzcd}
M \arrow[rr, two heads, "\beta"'] \arrow[d, "\beta"'] & & N_2 \arrow[d, dashed, "\beta"] \\
M_1 \arrow[rr, dashed, two heads, "\beta"'] \arrow[d, dashed, "\beta"'] & & \cdot \arrow[d, dashed, "\beta"] \\
\cdot \arrow[rr, dashed, two heads, "\beta"'] \arrow[d, dashed, no head] & & \cdot \arrow[d, dashed, no head] \\
\cdot \arrow[d, dashed, "\beta"'] & & \cdot \arrow[d, dashed, "\beta"] \\
\cdot \arrow[rr, dashed, two heads, "\beta"'] \arrow[d, dashed, "\beta"'] & & \cdot \arrow[d, dashed, "\beta"] \\
N_1 \arrow[rr, dashed, two heads, "\beta"'] & & N_3
\end{tikzcd}$$

□

Corollary 3.17 *For $M, N \in \mathcal{T} \cup T$, if $M =_\beta N$ then there exists $L \in \mathcal{T} \cup T$ such that $M \twoheadrightarrow_\beta L$ and $N \twoheadrightarrow_\beta L$*

Proof: *by induction on $=_\beta$.*

If $M =_\beta N$ comes from $M \twoheadrightarrow_\beta N$ then take $L \equiv N$

If $M =_\beta N$ comes from $N =_\beta M$ where property holds for $N =_\beta M$, nothing to prove.

If $M =_\beta N$ comes from $M =_\beta L$ and $L =_\beta N$ where property holds for $M =_\beta L$ and $L =_\beta N$ then there exists L_1, L_2 such that:

$$\begin{tikzcd}
M \arrow[dr, two heads] & & L \arrow[dl, two heads] \arrow[dr, two heads] & & N \arrow[dl, two heads] \\
& L_1 & & L_2 &
\end{tikzcd}$$

As $L \twoheadrightarrow_\beta L_1$ and $L \twoheadrightarrow_\beta L_2$ then by CR there exists L_3 such that:

$$\begin{tikzcd}
L_1 \arrow[dr, two heads] & & L_2 \arrow[dl, two heads] \\
& L_3 &
\end{tikzcd}$$

Hence $M \twoheadrightarrow_\beta L_3$ and $N \twoheadrightarrow_\beta L_3$. □

Corollary 3.18

1. *For $M, N \in \mathcal{T} \cup T$, if M has N as β-nf then $M \twoheadrightarrow_\beta N$*
2. *An expression has at most one β-nf.*

Proof:

1. *M has N as β-nf then $M =_\beta N$ and N is a β-nf. Hence by corollary above, there exists L such that $M \twoheadrightarrow_\beta L$ and $N \twoheadrightarrow_\beta L$. This implies that $N \equiv L$ up to renaming of variables and so $M \twoheadrightarrow_\beta N$.*
2. *If M has N_1 and N_2 as β-nf then $M \twoheadrightarrow_\beta N_1$ and $M \twoheadrightarrow_\beta N_2$ and so there exists M' such that:*

$$\begin{tikzcd}
N_1 \arrow[dr] & & N_2 \arrow[dl] \\
& M' &
\end{tikzcd}$$

But as N_1, N_2 are in β-nf then $N_1 \equiv M' \equiv N_2$ and hence $N_1 \equiv N_2$.

□

Corollary 3.19 *If $M =_\beta \pi_{y:\sigma}.M'$ and $N =_\beta \sigma$ then*
$(\exists \sigma_1, M'')[M \twoheadrightarrow_\beta \pi_{y:\sigma_1}.M''$ *and* $N \twoheadrightarrow_\beta \sigma_1]$.
Proof: *By Corollary 3.17, $(\exists \sigma')[N \twoheadrightarrow_\beta \sigma'$ and $\sigma \twoheadrightarrow_\beta \sigma']$.*
Hence of course, $M =_\beta \pi_{y:\sigma'}.M'$.
We apply again Corollary 3.17 to get
$(\exists \sigma_1, M'')[M \twoheadrightarrow_\beta \pi_{y:\sigma_1}.M''$ *and* $\pi_{y:\sigma'}.M' \twoheadrightarrow_\beta \pi_{y:\sigma_1}.M''$ *and* $\sigma' \twoheadrightarrow_\beta \sigma_1]$.
Hence $M \twoheadrightarrow_\beta \pi_{y:\sigma_1}.M''$ and $N \twoheadrightarrow_\beta \sigma_1$. □

Corollary 3.20 *($=_\beta$-substitution lemma)*
For $M, N \in \mathcal{T} \cup T$, $z \in V$ and $t' \in T$, if $M =_\beta N$, then $M[z := t'] =_\beta N[z := t']$.
Proof: *If $M =_\beta N$, then there exists M' such that $M \twoheadrightarrow_\beta M'$ and $N \twoheadrightarrow_\beta M'$ by Corollary 3.17. By Lemma 2.20, $M[z := t'] \twoheadrightarrow_\beta M'[z := t']$ and $N[z := t'] \twoheadrightarrow_\beta M'[z := t']$. Hence, $M[z := t'] =_\beta N[z := t']$.* □

4 Statements and Contexts

Fundamental in typed lambda calculus is the relation "t has type σ". This relation is formalized as the statement $t : \sigma$. In associating types to terms, contexts play an important role. The following definitions concern statements and contexts.

Definition 4.1 *(Statement, subject, type)*
A statement is of the form $t : \sigma$ with $t \in T$, $\sigma \in \mathcal{T}$.
t is the subject, σ is the type of the statement $t : \sigma$.

Definition 4.2 *(Contexts)*
Γ is a context if Γ is a finite linearly ordered set of statements with (term) variables as subjects.

We let $CONS$ be the collection of all contexts. Contexts are written as lists of pairs $(y : \sigma)$ where $y \in V$ and $\sigma \in \mathcal{T}$. We write $\Gamma(y : \sigma)$ for the context obtained by appending $(y : \sigma)$ at the end of the list Γ. Notations like $\Gamma(y : \sigma)\Gamma''$ etc. are used in the same manner.

We use Γ, $\Gamma', \Gamma_1, \Gamma_2, \ldots$ as meta variables for contexts.

Definition 4.3 *(Domain and Range of contexts)*
For $\Gamma \in CONS$, we define the domain of Γ, $dom(\Gamma)$ and the range of Γ, $ran(\Gamma)$ as follows:

- $dom(\Gamma) = \{y \in V; (\exists \sigma \in \mathcal{T})[(y : \sigma) \in \Gamma]\}$
- $ran(\Gamma) = \{\sigma \in \mathcal{T}; (\exists y \in V)[(y : \sigma) \in \Gamma]\}$

Definition 4.4 *(Free Variables of contexts)*
For $\Gamma \in CONS$, we define the free variables of Γ, $FV(\Gamma)$, to be $\cup_{\sigma \in ran(\Gamma)} FV(\sigma)$.

Definition 4.5 *(Bound Variables of contexts)*
For $\Gamma \in CONS$, we define the bound variables of Γ, $BV(\Gamma)$, to be $\cup_{\sigma \in ran(\Gamma)} BV(\sigma)$.

Definition 4.6 *(Variables of contexts)*
For $\Gamma \in CONS$, we define the variables of Γ, $VAR(\Gamma)$ to be $FV(\Gamma) \cup BV(\Gamma) \cup dom(\Gamma)$.

Definition 4.7 *(Substitution in contexts)*
For $\Gamma \in CONS$, $\Gamma = (y_1 : \sigma_1) \ldots (y_n : \sigma_n)$,
$\Gamma[\omega := N] = (y_1 : \sigma_1[\omega := N]) \ldots (y_n : \sigma_n[\omega := N])$.

Definition 4.8 *(One-step reduction of contexts)*
For $\Gamma, \Gamma' \in CONS$, we say $\Gamma \to_\beta \Gamma'$ if the following holds:
$\Gamma = (y_1 : \sigma_1) \ldots (y_n : \sigma_n)$ and $\Gamma' = (y_1 : \sigma'_1) \ldots (y_n : \sigma'_n)$ with $\sigma_i \to_\beta \sigma'_i$ for some $i \in \{1, \ldots, n\}$.

Definition 4.9 *(Reduction of contexts)*
$\Gamma \twoheadrightarrow_\beta \Gamma'$ is the reflexive transitive closure of $\Gamma \to_\beta \Gamma'$.

Note that we use $\to_\beta$ and $\twoheadrightarrow_\beta$ to mean both reduction of contexts and reduction of expressions. This should not lead to confusion.

Example 4.10

1. $(y : (\Pi_{z:\sigma}.\gamma)z')(y' : (\Pi_{z:\sigma}.\gamma)z'(\lambda_{z_1.\sigma}.z_1)z') \to_\beta (y : \gamma)(y' : (\Pi_{z:\sigma}.\gamma)z'(\lambda_{z_1.\sigma}.z_1)z')$

2. $(y : (\Pi_{z:\sigma}.\gamma)z')(y' : (\Pi_{z:\sigma}.\gamma)z'(\lambda_{z_1.\sigma}.z_1)z') \to_\beta (y : (\Pi_{z:\sigma}.\gamma)z')(y' : (\Pi_{z:\sigma}.\gamma)z'z')$

3. $(y : (\Pi_{z:\sigma}.\gamma)z')(y' : (\Pi_{z:\sigma}.\gamma)z'(\lambda_{z_1.\sigma}.z_1)z') \twoheadrightarrow_\beta (y : (\Pi_{z:\sigma}.\gamma)z')(y' : \gamma z')$

4. $(y : (\Pi_{z:\sigma}.\gamma)z')(y' : (\Pi_{z:\sigma}.\gamma)z'(\lambda_{z_1.\sigma}.z_1)z') \twoheadrightarrow_\beta (y : \gamma)(y' : \gamma z')$

Definition 4.11 *(Restriction of contexts to sets of variables)*
If $\Gamma \in CONS$ and $S \subseteq V$ then $\Gamma \lceil S$ is the restriction of Γ to S, that is the list Γ' obtained from Γ by removing all $(y : \sigma)$ from Γ with $y \notin S$.

Remark 4.12 *Note that $(\Gamma \lceil S') \lceil S \equiv \Gamma \lceil (S \cap S')$. Note moreover that $(\Gamma\Gamma') \lceil S \equiv (\Gamma \lceil S)(\Gamma' \lceil S)$.*

4.1 Context ordering

We need an ordering relation on contexts:

Definition 4.13 *(Ordering of contexts)*
$\Gamma \sqsubseteq^1 \Gamma'$ *if* $\Gamma \equiv \Gamma_1\Gamma_2$, $\Gamma' \equiv \Gamma_1(y:\sigma)\Gamma_2$ *and* $y \notin dom(\Gamma_1)$
The relation $\sqsubseteq$ *is the reflexive and transitive closure of* $\sqsubseteq^1$.

Example 4.14 $(z:\sigma)(y:\sigma') \sqsubseteq^1 (y:\sigma)(z:\sigma)(y:\sigma')$ and
$(z:\sigma)(y:\sigma') \sqsubseteq^1 (z:\sigma)(y:\sigma)(y:\sigma')$ for $y \not\equiv z$.
Furthermore, $(y_1:\sigma_1)(y_2:\sigma_2) \sqsubseteq (y_1:\sigma_2)(y_1:\sigma_1)(y_2:\sigma_1)(y_3:\sigma_2)(y_2:\sigma_2)(y_4:\sigma_1)$ for $y_i \not\equiv y_j$ if $i \neq j$.

The motivation of the condition $y \notin dom(\Gamma_1)$ will be given in Lemma 4.28. Be careful not to confuse $\sqsubseteq$ with set inclusion which we write as $\subseteq$. In fact, $\Gamma \sqsubseteq \Gamma' \Rightarrow \Gamma \subseteq \Gamma'$, but the reverse is not true. The reverse however is true in the following case:

Lemma 4.15 *If* $dom(\Gamma) \cap dom(\Gamma') = \emptyset$ *then* $\Gamma \sqsubseteq \Gamma\Gamma'$.
Proof: *By induction on the length of* Γ'. □

Lemma 4.16
If $\Gamma \sqsubseteq \Gamma'$, *then* $\Gamma(y:\sigma) \sqsubseteq \Gamma'(y:\sigma)$.
Proof:

- *Case* $\Gamma \sqsubseteq^1 \Gamma'$ *since* $\Gamma \equiv \Gamma_1\Gamma_2 \sqsubseteq^1 \Gamma_1(z:\sigma')\Gamma_2 \equiv \Gamma'$ *and* $z \notin dom(\Gamma_1)$.
  Then $\Gamma(y:\sigma) \equiv \Gamma_1\Gamma_2(y:\sigma) \sqsubseteq^1 \Gamma_1(z:\sigma')\Gamma_2(y:\sigma) \equiv \Gamma'(y:\sigma)$ *since* $z \notin dom(\Gamma_1)$, *both if* $z \equiv y$ *and if* $z \not\equiv y$.
- *Case* $\Gamma \sqsubseteq \Gamma'$ *by reflexivity or transitivity.*
  These cases are trivial.

□

Now, a notion which helps one understand $\sqsubseteq$ is that of *part*:

Definition 4.17 *(Part)*
A context Γ *is a part of another context* Γ' *if* Γ *is a sequence consisting of some statements of* Γ' *written in the order in which they occurred in* Γ'. *We use* $\leq$ *as a notation for 'is a part of'.*

For example, $(y_2:\sigma_2)(y_4:\sigma_4) \leq (y_1:\sigma_1)(y_2:\sigma_2)(y_3:\sigma_3)(y_4:\sigma_4)$ for $y_i \not\equiv y_j$ for $i \neq j$.

Definition 4.18 *For* Γ *a context and* $(y:\sigma)^\circ$ *a particular occurrence of* $(y:\sigma) \in \Gamma$ *we define* $L((y:\sigma)^\circ, \Gamma)$ *to be the context formed from the beginning (to the left) of* Γ *until (and excluding) this occurrence of* $(y:\sigma)$.

Example 4.19
$L((y_2:\sigma_2),(y_1:\sigma_1)^1(y_2:\sigma_2)(y_1:\sigma_1)^2(y_3:\sigma_3)) = (y_1:\sigma_1)$,
$L((y_1:\sigma_1)^1,(y_1:\sigma_1)^1(y_2:\sigma_2)(y_1:\sigma_1)^2(y_3:\sigma_3)) = \emptyset$,
$L((y_1:\sigma_1)^2,(y_1:\sigma_1)^1(y_2:\sigma_2)(y_1:\sigma_1)^2(y_3:\sigma_3)) = (y_1:\sigma_1)(y_2:\sigma_2)$.

Note that it is possible that $\Gamma \leq \Gamma'$ but $\Gamma \not\sqsubseteq \Gamma'$. For example, $(y:\sigma) \leq (y:\sigma)(y:\sigma')$. It is the case however that if $\Gamma \sqsubseteq \Gamma'$ then $\Gamma \leq \Gamma'$. Here is the lemma which gives the relationship between $\sqsubseteq$ and the easier notion $\leq$.

Lemma 4.20
$\Gamma \sqsubseteq \Gamma'$ iff $\Gamma \leq \Gamma'$ and $\forall (y:\sigma)^\circ \in \Gamma' \setminus \Gamma, y \notin dom(L((y:\sigma)^\circ, \Gamma')) \cap dom(\Gamma)$.
Proof:
$\Rightarrow$ *By induction on $\Gamma \sqsubseteq \Gamma'$.*

- *Case $\Gamma \sqsubseteq^1 \Gamma'$ then $\Gamma \equiv \Gamma_1\Gamma_2 \sqsubseteq \Gamma_1(y:\sigma)\Gamma_2 \equiv \Gamma'$. Hence $\Gamma \leq \Gamma'$. Furthermore, $(y:\sigma)$ is the only occurrence in $\Gamma' \setminus \Gamma$, and as $\Gamma \sqsubseteq^1 \Gamma'$ then $y \notin dom(\Gamma_1) = dom(L((y:\sigma), \Gamma')) \cap dom(\Gamma)$.*
- *Case $\Gamma \sqsubseteq \Gamma$ obvious.*
- *Case $\Gamma \sqsubseteq \Gamma_1$ and $\Gamma_1 \sqsubseteq \Gamma'$ use IH and transitivity of $\leq$ to get $\Gamma \leq \Gamma'$. Furthermore, let $(y:\sigma)^\circ \in \Gamma' \setminus \Gamma$.*
  - *Case $(y:\sigma)^\circ \notin \Gamma_1$ then $(y:\sigma)^\circ \in \Gamma' \setminus \Gamma_1$. Hence by IH, $y \notin dom(L((y:\sigma)^\circ, \Gamma')) \cap dom(\Gamma_1) \supseteq^{\Gamma \leq \Gamma_1} dom(L((y:\sigma)^\circ, \Gamma')) \cap dom(\Gamma)$.*
  - *Case $(y:\sigma)^\circ \in \Gamma_1$ then $(y:\sigma)^\circ \in \Gamma_1 \setminus \Gamma$. Hence by IH, $y \notin dom(L((y:\sigma)^\circ, \Gamma_1)) \cap dom(\Gamma) =^{\Gamma_1 \leq \Gamma'} dom(L((y:\sigma)^\circ, \Gamma')) \cap dom(\Gamma)$.*

$\Leftarrow$ *By induction on the size of $\Gamma' \setminus \Gamma$, $|\Gamma' \setminus \Gamma|$.*

- *Case $\Gamma_1\Gamma_2 \leq \Gamma_1(y:\sigma)\Gamma_2$ and $y \notin dom(L((y:\sigma), \Gamma_1(y:\sigma)\Gamma_2)) \cap dom(\Gamma_1\Gamma_2) = dom(\Gamma_1)$, then $\Gamma_1 \sqsubseteq^1 \Gamma_2$.*
- *Assume IH holds for any Γ, Γ' where $|\Gamma' \setminus \Gamma| = n$. Take Γ, Γ' such that $|\Gamma' \setminus \Gamma| = n+1$, $\Gamma \leq \Gamma'$ and $\forall (y:\sigma)^\circ \in \Gamma' \setminus \Gamma$, $y \notin dom(L((y:\sigma)^\circ, \Gamma')) \cap dom(\Gamma)$. As $|\Gamma' \setminus \Gamma| = n+1$, let $(z:\sigma_1)$ be the leftmost element in $\Gamma' \setminus \Gamma$. Hence $\Gamma' = \Gamma'_1(z:\sigma_1)\Gamma'_2$ and $z \notin dom(\Gamma'_1) \cap dom(\Gamma) = dom(\Gamma'_1)$. As $\Gamma \leq \Gamma'$ and $(z:\sigma_1) \notin \Gamma$, then $\Gamma \leq \Gamma'_1\Gamma'_2$. Furthermore, $\forall (y:\sigma)^\circ \in \Gamma' \setminus \Gamma$, $y \notin dom(L((y:\sigma)^\circ, \Gamma')) \cap dom(\Gamma)$ implies $\forall (y:\sigma)^\circ \in (\Gamma'_1\Gamma'_2) \setminus \Gamma$, $y \notin dom(L((y:\sigma)^\circ, \Gamma'_1\Gamma'_2)) \cap dom(\Gamma)$ can be seen as follows: Let $(y:\sigma)^\circ \in (\Gamma'_1\Gamma'_2) \setminus \Gamma$.*
  - *Case $(y:\sigma)^\circ \in \Gamma'_1$ then $L((y:\sigma)^\circ, \Gamma') = L((y:\sigma)^\circ, \Gamma'_1\Gamma'_2) = L((y:\sigma)^\circ, \Gamma'_1)$.*
  - *Case $(y:\sigma)^\circ \in \Gamma'_2$ then $dom(L((y:\sigma)^\circ, \Gamma')) = dom(L((y:\sigma)^\circ, \Gamma'_1\Gamma'_2)) \cup \{z\}$.*
    - *Case $z \neq y$ then as $y \notin dom(L((y:\sigma)^\circ, \Gamma')) \cap dom(\Gamma)$, then $y \notin dom(L((y:\sigma)^\circ, \Gamma'_1\Gamma'_2)) \cap dom(\Gamma)$.*
    - *Case $z = y$ then as $(z:\sigma)^\circ \in (\Gamma'_1\Gamma'_2) \setminus \Gamma$, $(z:\sigma)^\circ \in \Gamma' \setminus \Gamma$. Hence $z \notin dom(L((z:\sigma)^\circ, \Gamma')) \cap dom(\Gamma)$. But $(z:\sigma_1) \in L((z:\sigma)^\circ, \Gamma')$. Hence $z \notin dom(\Gamma)$. And so, $z \notin dom(L((y:\sigma)^\circ, \Gamma'_1\Gamma'_2)) \cap dom(\Gamma)$.*

  Hence by IH, $\Gamma \sqsubseteq \Gamma'_1\Gamma'_2$. Furthermore, as $z \notin dom(\Gamma'_1)$, then $\Gamma'_1\Gamma'_2 \sqsubseteq \Gamma'_1(z:\sigma_1)\Gamma'_2$. Hence by transitivity of $\sqsubseteq$, $\Gamma \sqsubseteq \Gamma'$.

□

Corollary 4.21
If $\forall y_i, y_j \in dom(\Gamma'), i \neq j \Rightarrow y_i \neq y_j$ and $\Gamma \leq \Gamma'$ then $\Gamma \sqsubseteq \Gamma'$.
Proof: *Apply Lemma 4.20* □

Now if, for example, $\Gamma \equiv \Gamma_1(y : \sigma_1)\Gamma_2(y : \sigma_2)\Gamma_3$, with $y \not\in dom(\Gamma_3)$, then the statement meaningful to a free y in Γ_3 is the rightmost, viz. $(y : \sigma_2)$, as we will see below. The following lemma shows that context ordering preserves this property.

Lemma 4.22 *If $\Gamma \sqsubseteq \Gamma'$ and $(y : \sigma)$ is the rightmost statement in Γ whose subject is y, then $(y : \sigma)$ is the rightmost statement in Γ' whose subject is y.*
Proof: *By induction on $\Gamma \sqsubseteq \Gamma'$.* □

This lemma is important. It says that if y gets type σ in Γ and if $\Gamma \sqsubseteq \Gamma'$ then y gets type σ in Γ'. I.e. Γ' knows everything that Γ knows together with some information about variables which do not belong to $dom(\Gamma)$. We can capture this information by defining the binding structure of a context-and-expression pair:

Definition 4.23 *(The Binding Structure of a Context-and-Expression Pair; Companion expression)*
We say that a variable occurrence y is free, respectively bound (with type σ) in the pair $(\Gamma, M) \equiv ((y_1 : \sigma_1)\dots(y_n : \sigma_n), M)$ iff the corresponding occurrence of y is free, respectively bound (with type σ) in the expression
$\pi_{y_1:\sigma_1}.\dots.\pi_{y_n:\sigma_n}.M$ *where $\pi = \lambda$ if $M \in T$ and $\pi = \Pi$ if $M \in \mathcal{T}$.*
This expression is called the companion expression of (Γ, M).

Example 4.24 Let $(\Gamma, M) \equiv ((y_1 : \sigma_1)\dots(y_n : \sigma_n), M)$. If $M \equiv y_i$ and $y_i \not\in \{y_{i+1}, \dots, y_n\}$, then y_i is bound with type σ_i in the pair (Γ, y_i) with Γ as above. If $M \equiv y$ and $y \not\in \{y_1, y_2, \dots, y_n\}$, then y is free in (Γ, y). If $M \equiv (\lambda_{y:\sigma}.y)$, then y is bound in (Γ, M) with type σ (as it already was in M itself). If $\sigma_2 \equiv (\sigma \to \tau)y_1$, then the occurrence of y_1 in σ_2 is bound in (Γ, M) with type σ_1.

Definition 4.25

$$\begin{array}{lcl} FV(\Gamma, M) & = & \textit{the free variables of the companion expression of } M \\ BV(\Gamma, M) & = & \textit{the bound variables of the companion expression of } M \end{array}$$

We define a notion of α-reduction between context-and-expression pairs:

Definition 4.26 *(Variants of Context-and-expression Pairs)*
(Γ, M) is α-equivalent with (or an α-variant of) (Γ', M') if the corresponding companion expressions are α-equivalent. (Recall that we use α-equivalence in our mind rather than on paper.)

Lemma 4.27

- $y \in dom(\Gamma) \Rightarrow y \not\in FV(\Gamma, M)$.
- $FV(\Gamma, M) = (FV(\Gamma) \cup FV(M)) \setminus dom(\Gamma)$.
- $BV(\Gamma, M) = BV(\Gamma) \cup BV(M) \cup dom(\Gamma)$

Proof:

- *By the variable convention which holds for $\pi_{y_1:\sigma_1}.\dots.\pi_{y_n:\sigma_n}.M$, where again $\Gamma \equiv (y_1 : \sigma_1)\dots(y_n : \sigma_n)$.*

- *There may be free variables in Γ which are also elements of $dom(\Gamma)$, but these variables are bound in (Γ, M) and we have VC holds for the companion expression of (Γ, M).*
- *Obvious.*

$\Box$

Note that Definition 4.23 establishes the binding pattern in a pair (Γ, M): each occurrence of a variable in $BV(\Gamma, M)$ which is neither a subscript of a π in M nor a domain variable of Γ, is linked to an occurrence of the same variable which *is* such a subscript or domain variable. This linkage can be found by inspecting the companion expression of (Γ, M). Now it can be shown that context ordering does not disturb this binding pattern:

Lemma 4.28
Let $\Gamma \sqsubseteq \Gamma'$. If an occurrence of a bound variable y is linked in (Γ, M) to a certain other occurrence of this y in (Γ, M) (being either a subscript of a π or a domain variable), then this is also the case in (Γ', M), for the corresponding occurrences.

Proof: *It is sufficient to investigate the case $\Gamma \equiv \Gamma_1\Gamma_2 \sqsubseteq^1 \Gamma_1(y:\sigma)\Gamma_2 \equiv \Gamma'$. The binding pattern can only be disturbed when the inserted domain variable y "takes over" the binding for an already present bound occurrence of y. (Note that y cannot be free in (Γ', M) by Lemma 4.27.) Now it cannot be the case that an occurrence of y in (Γ, M) becomes linked to the inserted domain variable y, instead of its original linkage. The reason is the restriction $y \notin dom(\Gamma_1)$ which is a consequence of $\sqsubseteq^1$.* $\Box$

A consequence of this lemma is that context ordering does not influence types of bound variables:

Corollary 4.29 *If $\Gamma \sqsubseteq \Gamma'$ and if y is bound with type σ in (Γ, M), then y is bound with type σ in (Γ', M).* $\Box$

4.2 Well-Behaved contexts

In certain circumstances, which will become clear below, we need a condition on the variables in a context. Such a condition says that all variables in the domain of a context must be distinct and that in a context $\Gamma(y:\sigma)\Gamma'$, $FV(\sigma) \cap dom((y:\sigma)\Gamma') = \emptyset$.

The intuition is that we wish to be free to substitute σ for a variable in Γ' or for a variable 'depending on' Γ', without running the risk that a free variable of σ becomes unintentionally bound. See [Barendregt 91] where a similar discussion is given for contexts.

Definition 4.30 *(Well-behaved contexts)*
A context $\Gamma \equiv (y_1:\sigma_1)\ldots(y_n:\sigma_n)$ is well-behaved, and we write $WB(\Gamma)$, if for all $i,j,k \in \{1,\ldots,n\}$:

1. $y_i = y_j \Rightarrow i = j$,
2. $FV(\sigma_k) \cap \{y_k,\ldots,y_n\} = \emptyset$. *That is,* $y_i \in FV(\sigma_k) \Rightarrow 1 \le i < k$.

We give a few lemmas concerning the well-behavedness of contexts.

Lemma 4.31

1. *If $WB(\Gamma)$ and $\Gamma' \le \Gamma$ then $WB(\Gamma')$.*

2. *If $WB(\Gamma)$, $y \notin dom(\Gamma)$, $y \notin FV(\Gamma)$ and $y \notin FV(\sigma)$, then $WB(\Gamma(y:\sigma))$.*

3. *If $WB(\Gamma)$ and $\Gamma \twoheadrightarrow_\beta \Gamma'$ then $WB(\Gamma')$.*

Proof:

1. *Easy.*

2. *Let $\Gamma \equiv (y_1 : \sigma_1)\ldots(y_n : \sigma_n)$. Then for all $k \in \{1,\ldots,n\}$: $FV(\sigma_k) \cap \{y_k,\ldots,y_n\} = \emptyset$. Hence, since $y \notin FV(\sigma_k)$, $FV(\sigma_k) \cap \{y_k,\ldots,y_n,y\} = \emptyset$. Moreover, $FV(\sigma) \cap \{y\} = \emptyset$ and $y \not\equiv y_i$ for any $i \in \{1,\ldots,n\}$. So $WB(\Gamma(y:\sigma))$.*

3. *Assume $\Gamma \rightarrow_\beta \Gamma'$ and comes from $\Gamma = \Gamma_1(y:\sigma)\Gamma_2$ where $\sigma \rightarrow_\beta \sigma'$ and $\Gamma' = \Gamma_1(y:\sigma')\Gamma_2$. Then use the fact that $FV(\sigma') \subseteq FV(\sigma)$ and $FV(\sigma) \cap (\{y\} \cup dom(\Gamma_2)) = \emptyset$.*

   The case $\Gamma \twoheadrightarrow_\beta \Gamma'$ is now a trivial consequence.

□

Corollary 4.32

1. *If $WB(\Gamma_1\Gamma_2)$ then $WB(\Gamma_1)$.*

2. *If $WB(\Gamma_1\Gamma_2)$ then $WB(\Gamma_1\Gamma_2')$ where Γ_2' is a prefix of Γ_2.*

3. *If $WB(\Gamma)$ and $\Gamma' \sqsubseteq \Gamma$ then $WB(\Gamma')$.*

Proof: *All of 1, 2, and 3 are corollaries of Lemma 4.31, part 1.* □

Lemma 4.33
If $WB(\Gamma(y:\sigma))$, then $\Gamma \sqsubseteq^1 \Gamma(y:\sigma)$.

Proof: *Note that $WB(\Gamma(y:\sigma))$ implies that $y \notin dom(\Gamma)$.*

□

Corollary 4.34
If $WB(\Gamma_1\Gamma_2)$, then $\Gamma_1 \sqsubseteq \Gamma_1\Gamma_2$.

Proof: *Induction on the length of the list Γ_2. Note that by Corollary 4.32, part 2, all $\Gamma_1\Gamma_2'$ with Γ_2' prefix of Γ_2 are well-behaved. Lemma 4.33 plus transitivity of $\sqsubseteq$ give the desired result.* □

Lemma 4.35 *If $WB(\Gamma')$ then $\Gamma \sqsubseteq \Gamma'$ iff $\Gamma \leq \Gamma'$.*

Proof: *This is a corollary of Lemma 4.20.* □

Corollary 4.36 *If $WB(\Gamma)$ and $\Gamma_1 \leq \Gamma_2 \leq \Gamma$, then $\Gamma_1 \sqsubseteq \Gamma_2$.*

Proof: *Use Lemmas 4.31 and 4.35.* □

The following lemma shows that each context-and-expression pair is α-equivalent with a context-and-expression pair which has a well-behaved context. Hence, the restriction to well-behaved contexts is not an essential restriction.

Lemma 4.37
For each (Γ, M) there is an α-variant (Γ', M') such that $WB(\Gamma')$.

Proof: *Assume $\Gamma \equiv (y_1 : \sigma_1)\ldots(y_n : \sigma_n)$. Replace all occurrences of the first domain variable y_1 and of all occurrences of y_1 linked to this y_1 in (Γ, M), by a fresh variable y_1' and repeat this procedure successively for the other domain variables $y_2,\ldots y_n$. It is clear that we obtain a pair (Γ', M') with α-equivalent companion expression and such that $WB(\Gamma')$.* □

5 A Typing Operator for $\lambda_{\rightarrow\tau}$

We let τ be a canonical type operator in $\lambda_{\rightarrow\tau}$. That is τ takes a context Γ in $CONS$ and a term t in T of $\lambda_{\rightarrow\tau}$ and gives the type of t with respect to Γ, according to the following typing rules:

Definition 5.1 *(Canonical Type Operator)*
$\tau : CONS \times T \longrightarrow T$ is defined as follows:

1. *$\tau(\Gamma, y) \equiv \sigma$ if y is bound with type σ in (Γ, y)*
2. *$\tau(\Gamma, tt') \equiv \tau(\Gamma, t)t'$*
3. *$\tau(\Gamma, \lambda_{y:\sigma}.t) \equiv \Pi_{y:\sigma}.\tau(\Gamma(y:\sigma), t)$*

Here clause 2 may not be obvious at first sight. In fact, one may have expected $\tau(\Gamma, tt')$ to be defined as $\tau(\Gamma, t)\tau(\Gamma, t')$. This certainly cannot be the case for many reasons. First, recall that a type applies to a term and not to a type. Second, if we allow types as arguments to types, then we must let Π range over type variables instead of only term variables. Furthermore, think about the intuition behind such a definition. When we look for the type of t, where t has the form $\lambda_{y:\sigma}.t''$ then we obtain a Π-type $\Pi_{y:\sigma}.\sigma'$. Now certainly the type of tt' must be $\sigma'[y := t']$, which is a reduct of $(\Pi_{y:\sigma}.\sigma')t'$.

Here moreover, we should draw some attention to the power of typing in $\lambda_{\rightarrow\tau}$. We will show below that any term which is typable in Church's $\lambda_{\rightarrow}$ is also typable in $\lambda_{\rightarrow\tau}$. This should not be surprising as $\lambda_{\rightarrow}$ types fewer terms than other systems (see [Barendregt 92], and [BH 90]). We can type more terms however in $\lambda_{\rightarrow\tau}$. For instance, we will see in Example 5.4 that $\omega \equiv \lambda_{y:\sigma}.yy$ is typable. Such a term cannot be typed in $\lambda_{\rightarrow}$. Furthermore, the fixed point operator $t \equiv \lambda_{y_1:(\Pi_{z:\sigma}.\sigma)}(\lambda_{y_2:\sigma}.y_1(y_2y_2))(\lambda_{y_2:\sigma}.y_1(y_2y_2))$ has a type which reduces to $\Pi_{y_1:(\Pi_{z:\sigma}.\sigma)}\sigma$ (see Example 5.4). This makes sense: it says that the type of the fixed point operator is $(\sigma \rightarrow \sigma) \rightarrow \sigma$. Such a term however is not typable in $\lambda_{\rightarrow}$ nor in the second order system λ_2 (see [Barendregt 92]). Now this brings the question of strong normalisation. We say that an expression M is strongly normalising iff all reduction sequences starting with M terminate. Now $\omega\omega$ is not strongly normalising. Nor is the term $(\lambda_{y:\sigma}.z)(\omega\omega)$ even though this term has z as a normal form. Furthermore, we can type (via τ) such a non strongly normalising term. That is, $\tau(\emptyset, \omega\omega) = (\Pi_{y:\sigma}.\sigma y)\omega$ (see Example 5.4). This should not lead to problems however. That is, we conjecture that $\lambda_{\rightarrow\tau}$ is strongly normalising in the following form:

For all $M \in \mathcal{T} \cup T$, for all $\Gamma \in CONS$, if $\Gamma \vdash M$ then M is strongly normalising and if $M \in T$ then $\tau(\Gamma, M)$ is strongly normalising.

Remark 5.2 The variable convention also holds for all pairs (Γ, t). For example, $((y : \sigma z), \lambda_{z:\sigma'}.y)$ is not allowed, since for this (Γ, t) both $z \in BV(\Gamma, t)$ and $z \in FV(\Gamma, t)$.

Remark 5.3 *Note that τ is a partial function. For example $\tau(\emptyset, x)$ doesn't exist. We write $\restriction \tau(\Gamma, t)$ for $\tau(\Gamma, t)$ defined.*

Example 5.4

1. $\omega \equiv \lambda_{y:\sigma}.(yy)$ has type $\tau(\emptyset, \lambda_{y:\sigma}.(yy)) \equiv \Pi_{y:\sigma}.\tau(y : \sigma, yy) \equiv \Pi_{y:\sigma}.(\tau(y : \sigma, y)y) \equiv \Pi_{y:\sigma}.(\sigma y)$. Moreover, $y \notin FV(\sigma)$ by Lemma 2.9. If we allow η-conversion in our system, $\tau(\emptyset, \omega)$ would convert to σ.

2. $\tau(\emptyset, \lambda_{y_1:(\Pi_{z:\sigma}.\sigma)}.(\lambda_{y_2:\sigma}.y_1(y_2y_2))(\lambda_{y_2:\sigma}.y_1(y_2y_2))) \equiv$
   $\Pi_{y_1:(\Pi_{z:\sigma}.\sigma)}.((\Pi_{y_2:\sigma}.(\Pi_{z:\sigma}.\sigma)(y_2y_2))(\lambda_{y_2:\sigma}.y_1(y_2y_2))) \to_\beta$
   $\Pi_{y_1:(\Pi_{z:\sigma}.\sigma)}.((\Pi_{y_2:\sigma}.\sigma)(\lambda_{y_2:\sigma}.y_1(y_2y_2))) \to_\beta \Pi_{y_1:(\Pi_{z:\sigma}.\sigma)}.\sigma$
   Furthermore, by VC, $y_1 \notin FV(\Pi_{z:\sigma}.\sigma)$ and $z \notin \sigma$.
   Hence we write $\Pi_{y_1:(\Pi_{z:\sigma}.\sigma)}.\sigma$ as $(\sigma \to \sigma) \to \sigma$.

Remark 5.5 Note that $FV(\tau(\Gamma, t)) \neq FV(\Gamma, t)$. For example, consider the pair $(\Gamma, t) \equiv ((y_1, \alpha_1)(y_2, \alpha_2 y_1)(y_3, \alpha_3 y), y_2)$. Then $\tau(\Gamma, t) \equiv \alpha_2 y_1$. Now $y_1 \in FV(\tau(\Gamma, t))$, but $y_1 \notin FV(\Gamma, t)$. Also, $y_2 \in FV(\Gamma, t)$, but $y_2 \notin FV(\tau(\Gamma, t))$.

The following lemmas show that τ is a well-behaved typing operator. That is, it satisfies the weakening, reduction, restriction and substitution lemmas. In particular, the following weakening lemma states that if $\Gamma \sqsubseteq \Gamma'$ and $\tau(\Gamma, t)$ is defined then $\tau(\Gamma', t)$ is defined and is equal to $\tau(\Gamma, t)$.

Lemma 5.6 *(τ-weakening)*
If $\Gamma \sqsubseteq \Gamma'$ and $\uparrow \tau(\Gamma, t)$ then $\uparrow \tau(\Gamma', t)$ and $\tau(\Gamma, t) \equiv \tau(\Gamma', t)$.
Proof: *By induction on t.*

- *If $t \equiv y$ then y is bound with type σ in (Γ, t) iff y is bound with type σ in (Γ', t), by Lemma 4.28. Hence, $\tau(\Gamma, t) \equiv \tau(\Gamma', t)$.*
- *If $t \equiv t_1 t_2$ then $\tau(\Gamma, t) \equiv \tau(\Gamma, t_1 t_2) \equiv \tau(\Gamma, t_1)t_2 \equiv^{IH} \tau(\Gamma', t_1)t_2 \equiv \tau(\Gamma', t)$.*
- *If $t \equiv \lambda_{y:\sigma}.t'$ then by induction hypotheses (see Lemma 4.16):*
  $\tau(\Gamma(y : \sigma), t') \equiv \tau(\Gamma'(y : \sigma), t')$. It follows that
  $\tau(\Gamma, t) \equiv \Pi_{y:\sigma}.\tau(\Gamma(y : \sigma), t') \equiv^{IH} \Pi_{y:\sigma}.\tau(\Gamma'(y : \sigma), t') \equiv \tau(\Gamma', \lambda_{y:\sigma}.t')$.

□

The following lemma states that if Γ reduces to Γ' and if $\tau(\Gamma, t)$ is defined then $\tau(\Gamma', t)$ is defined and $\tau(\Gamma, t)$ reduces to $\tau(\Gamma', t)$.

Lemma 5.7 *(Contect-reduction for τ)*
If $\Gamma \twoheadrightarrow_\beta \Gamma'$ and $\uparrow \tau(\Gamma, t)$ then $\uparrow \tau(\Gamma', t)$ and $\tau(\Gamma, t) \twoheadrightarrow_\beta \tau(\Gamma', t)$.
Proof: *By induction on $\tau(\Gamma, t) \equiv \sigma$, as given in Definition 5.1.*

- *If $\tau(\Gamma, y) \equiv \sigma$ since y is bound with type σ in (Γ, y), then use Definitions 4.9 and 5.1.*
- *If $\tau(\Gamma, tt') \equiv \tau(\Gamma, t)t'$, then $\uparrow \tau(\Gamma, t)$, so by IH, $\uparrow \tau(\Gamma', t)$ and $\tau(\Gamma, t) \twoheadrightarrow_\beta \tau(\Gamma', t)$. Hence $\uparrow \tau(\Gamma', tt')$ and $\tau(\Gamma, tt') \equiv \tau(\Gamma, t)t' \twoheadrightarrow_\beta \tau(\Gamma', t)t' \equiv \tau(\Gamma', tt')$.*
- *If $\tau(\Gamma, \lambda_{y:\sigma}.t') \equiv \Pi_{y:\sigma}.\tau(\Gamma(y : \sigma), t')$, then*
  (as $\Gamma(y : \sigma) \twoheadrightarrow_\beta \Gamma'(y : \sigma)$ and $\uparrow \tau(\Gamma(y : \sigma), t')$), it holds by IH that
  $\uparrow \tau(\Gamma'(y : \sigma), t')$ and $\tau(\Gamma(y : \sigma), t') \twoheadrightarrow_\beta \tau(\Gamma'(y : \sigma), t')$. Hence $\uparrow \tau(\Gamma', \lambda_{y:\sigma}.t')$ and
  $\tau(\Gamma, \lambda_{y:\sigma}.t') \equiv \Pi_{y:\sigma}(\tau(\Gamma(y : \sigma), t')) \twoheadrightarrow_\beta \Pi_{y:\sigma}(\tau(\Gamma'(y : \sigma), t')) \equiv \tau(\Gamma', \lambda_{y:\sigma}.t')$.

□

The following lemma states that if $\uparrow \tau(\Gamma, t)$ then $\uparrow \tau(\Gamma \upharpoonright FV(t), t)$ and $\tau(\Gamma \upharpoonright FV(t), t) \equiv \tau(\Gamma, t)$.

Lemma 5.8 *(τ-restriction)*
If $\uparrow \tau(\Gamma, t)$ then $\tau(\Gamma \upharpoonright FV(t), t) \equiv \tau(\Gamma, t)$.
Proof: *By induction on t.*

- *If $\tau(\Gamma, y) \equiv \sigma$ since y is bound with type σ in (Γ, y), then inspection of the corresponding companion expressions shows that y is bound with type σ in $(\Gamma \upharpoonright \{y\}, y)$, so $\tau(\Gamma, y) \equiv \sigma \equiv \tau(\Gamma \upharpoonright FV(y), y)$.*
- $\tau(\Gamma, t_1 t_2) \equiv \tau(\Gamma, t_1)t_2 \equiv^{IH} \tau(\Gamma \upharpoonright FV(t_1), t_1)t_2 \equiv^{Remark\ 4.12}$
$\tau((\Gamma \upharpoonright FV(t_1 t_2)) \upharpoonright FV(t_1), t_1)t_2 \equiv^{IH} \tau(\Gamma \upharpoonright FV(t_1 t_2), t_1)t_2 \equiv \tau(\Gamma \upharpoonright FV(t_1 t_2), t_1 t_2)$
- *We have to show that $\tau(\Gamma, \lambda_{y:\sigma}.t) \equiv \tau(\Gamma \upharpoonright FV(\lambda_{y:\sigma}.t), \lambda_{y:\sigma}.t)$.*
First note that
$\tau(\Gamma, \lambda_{y:\sigma}.t) \equiv$
$\Pi_{y:\sigma}.\tau(\Gamma(y:\sigma), t) \equiv^{IH}$
$\Pi_{y:\sigma}.\tau(\Gamma(y:\sigma) \upharpoonright FV(t), t)$.
Now also
$\tau(\Gamma \upharpoonright FV(\lambda_{y:\sigma}.t), \lambda_{y:\sigma}.t) \equiv$
$\Pi_{y:\sigma}.\tau(\Gamma(y:\sigma) \upharpoonright FV(t), t)$,
since:
$\tau(\Gamma \upharpoonright FV(\lambda_{y:\sigma}.t), \lambda_{y:\sigma}.t) \equiv$
$\Pi_{y:\sigma}.\tau((\Gamma \upharpoonright FV(\lambda_{y:\sigma}.t))(y:\sigma), t) \equiv$
$\Pi_{y:\sigma}.\tau(\Gamma \upharpoonright (FV(\sigma) \cup (FV(t) \backslash \{y\}))(y:\sigma), t) \equiv^{IH}$
$\Pi_{y:\sigma}.\tau(\Gamma \upharpoonright (FV(\sigma) \cup (FV(t) \backslash \{y\}))(y:\sigma) \upharpoonright FV(t), t) \equiv$

  1. *case $y \in FV(t)$:*
  $\Pi_{y:\sigma}.\tau((\Gamma \upharpoonright (FV(\sigma) \cup (FV(t) \backslash \{y\})) \upharpoonright FV(t))(y:\sigma), t) \equiv^{Remark\ 4.12}$
  $\Pi_{y:\sigma}.\tau(\Gamma \upharpoonright ((FV(\sigma) \cup (FV(t) \backslash \{y\})) \cap FV(t))(y:\sigma), t) \equiv^{y \notin FV(\sigma)\ by\ VC}$
  $\Pi_{y:\sigma}.\tau(\Gamma \upharpoonright (FV(t) \backslash \{y\})(y:\sigma), t) \equiv^{Since \Gamma \upharpoonright (FV(t) \backslash \{y\})(y:\sigma) \sqsubseteq \Gamma(y:\sigma) \upharpoonright FV(t);\ see Lemma\ 5.6}$
  $\Pi_{y:\sigma}.\tau(\Gamma(y:\sigma) \upharpoonright FV(t), t)$.
  2. *case $y \notin FV(t)$ (then $FV(t) \backslash \{y\} = FV(t)$):*
  $\Pi_{y:\sigma}.\tau(\Gamma \upharpoonright (FV(\sigma) \cup FV(t)) \upharpoonright FV(t), t) \equiv^{Remark\ 4.12}$
  $\Pi_{y:\sigma}.\tau(\Gamma \upharpoonright (FV(t), t) \equiv$
  $\Pi_{y:\sigma}.\tau(\Gamma(y:\sigma) \upharpoonright (FV(t), t)$.

□

Corollary 5.9 *If $\uparrow \tau(\Gamma, t)$ then there is a $\Gamma_t \sqsubseteq \Gamma$ such that $\uparrow \tau(\Gamma_t, t)$, $\tau(\Gamma_t, t) \equiv \tau(\Gamma, t)$ and for all Γ_1, Γ_2, y and σ with $\Gamma_t \equiv \Gamma_1(y:\sigma)\Gamma_2$, we have $y \in FV(t)$.*
Proof: *Take $\Gamma_t \equiv \Gamma \upharpoonright FV(t)$.* □

Lemma 5.10 *(τ-Substitution Lemma)*

1. *If* $\uparrow \tau(\Gamma, t)$ *then* $\tau(\Gamma[\gamma := \sigma'], t[\gamma := \sigma']) \equiv \tau(\Gamma, t)[\gamma := \sigma']$.

2. *Assume* $WB(\Gamma(y : \sigma')\Gamma')$.
   If $\tau(\Gamma(y : \sigma')\Gamma', t) \equiv \sigma$, $y \notin FV(t')$ *and* $\tau(\Gamma, t') \sim \sigma'$, *then*
   $\tau((\Gamma\Gamma')[y := t'], t[y := t']) \sim \sigma[y := t']$, *for* $\sim$ *being* $\twoheadrightarrow_\beta, =_\beta$ *or* $\equiv$.

Proof:

1. *By induction on the length of* t

   - *If* $\tau(\Gamma, y) \equiv \sigma$ *since* y *is bound with type* σ *in* (Γ, y), *then*
     $\tau(\Gamma[\gamma := \sigma'], y[\gamma := \sigma']) \equiv \tau(\Gamma[\gamma := \sigma'], y) \equiv \sigma[\gamma := \sigma']$ *because* y *is then bound with type* $\sigma[\gamma := \sigma']$ *in* $\Gamma[\gamma := \sigma']$.
   - *If* $\tau(\Gamma, t_1t_2) \equiv \tau(\Gamma, t_1)t_2$ *where IH holds for* t_1, t_2, *then*
     $\tau(\Gamma[\gamma := \sigma'], t_1t_2[\gamma := \sigma']) \equiv \tau(\Gamma[\gamma := \sigma'], t_1[\gamma := \sigma'])t_2[\gamma := \sigma'] \equiv^{IH}$
     $(\tau(\Gamma, t_1)[\gamma := \sigma'])t_2[\gamma := \sigma'] \equiv (\tau(\Gamma, t_1)t_2)[\gamma := \sigma'] \equiv \tau(\Gamma, t_1t_2)[\gamma := \sigma']$.
   - *If* $\tau(\Gamma, \lambda_{y:\sigma}.t) \equiv \Pi_{y:\sigma}.\tau(\Gamma(y : \sigma), t)$ *where IH holds for* t, *then*
     $\tau(\Gamma[\gamma := \sigma'], (\lambda_{y:\sigma}.t)[\gamma := \sigma']) \equiv \tau(\Gamma[\gamma := \sigma'], (\lambda_{y:\sigma[\gamma:=\sigma']}.t[\gamma := \sigma'])) \equiv$
     $\Pi_{y:\sigma[\gamma:=\sigma']}.\tau(\Gamma[\gamma := \sigma'](y : \sigma[\gamma := \sigma']), t[\gamma := \sigma']) \equiv$
     $\Pi_{y:\sigma[\gamma:=\sigma']}.\tau(\Gamma(y : \sigma)[\gamma := \sigma'], t[\gamma := \sigma']) \equiv^{IH}$
     $\Pi_{y:\sigma[\gamma:=\sigma']}.(\tau(\Gamma(y : \sigma), t)[\gamma := \sigma'] \equiv$
     $(\Pi_{y:\sigma}.\tau(\Gamma(y : \sigma), t))[\gamma := \sigma'] \equiv \tau(\Gamma, \lambda_{y:\sigma}.t)[\gamma := \sigma']$.

2. *By induction on the length of* t. *We give the proof for* $\twoheadrightarrow_\beta$. *Note that* $y \notin FV(\Gamma)$ *and* $y \notin FV(\sigma')$ *by* $WB(\Gamma(y : \sigma')\Gamma'$.

   - *If* $t \equiv y$ *then*
     $\tau((\Gamma\Gamma')[y := t'], y[y := t']) \equiv \tau(\Gamma(\Gamma'[y := t']), t') \equiv^{Lemmas\ 4.15\ and\ 5.6}$
     $\tau(\Gamma, t') \twoheadrightarrow_\beta \sigma' \equiv \sigma'[y := t'] \equiv (\tau(\Gamma(y : \sigma')\Gamma', y))[y := t']$.
   - *If* $t \equiv z$ *where* $y \not\equiv z$ *and* $\Gamma \equiv \Gamma_1(z : \sigma'')\Gamma_2$, *then*
     $\tau((\Gamma\Gamma')[y := t'], z[y := t']) \equiv \tau(\Gamma(\Gamma'[y := t']), z) \equiv \sigma'' \equiv$
     $\sigma''[y := t'] \equiv (\tau(\Gamma(y : \sigma')\Gamma', z))[y := t']$.
   - *If* $t \equiv z$ *where* $y \not\equiv z$ *and* $\Gamma' \equiv \Gamma_1(z : \sigma'')\Gamma_2$, *then*
     $\tau((\Gamma\Gamma')[y := t'], z[y := t']) \equiv \tau(\Gamma(\Gamma'[y := t']), z) \equiv$
     $\sigma''[y := t'] \equiv (\tau(\Gamma(y : \sigma')\Gamma', z))[y := t']$.
   - *If* $t \equiv \lambda_{z:\sigma_1}.t_1$ *where* z *is a fresh variable*[4] *then*
     $\tau((\Gamma\Gamma')[y := t'], (\lambda_{z:\sigma_1}.t_1)[y := t']) \equiv$
     $\tau(\Gamma(\Gamma'[y := t']), \lambda_{z:\sigma_1[y:=t']}.t_1[y := t']) \equiv$
     $\Pi_{z:\sigma_1[y:=t']}.\tau((\Gamma\Gamma'(z : \sigma_1))[y := t'], t_1[y := t']) \twoheadrightarrow_\beta^{IH5}$
     $\Pi_{z:\sigma_1[y:=t']}.(\tau(\Gamma(y : \sigma')\Gamma'(z : \sigma_1), t_1)[y := t']) \equiv$
     $(\Pi_{z:\sigma_1}.\tau(\Gamma(y : \sigma')\Gamma'(z : \sigma_1), t_1))[y := t'] \equiv$
     $\tau(\Gamma(y : \sigma')\Gamma', \lambda_{z:\sigma_1}.t_1)[y := t']$.
   - *If* $t \equiv t_1t_2$ *then*
     $\tau((\Gamma\Gamma')[y := t'], t_1t_2[y := t']) \equiv$
     $\tau((\Gamma\Gamma')[y := t'], t_1[y := t']t_2[y := t']) \equiv$

[4]Without loss of generality.
[5]Note that $WB(\Gamma(y : \sigma')\Gamma'(z : \sigma_1))$ because z was fresh and the variable condition holds for $\lambda_{z:\sigma_1}.t_1$

$$
\begin{aligned}
&\tau((\Gamma\Gamma')[y := t'], t_1[y := t'])t_2[y := t'] \twoheadrightarrow_\beta^{IH}\\
&(\tau(\Gamma(y : \sigma')\Gamma', t_1)[y := t'])t_2[y := t'] \equiv\\
&(\tau(\Gamma(y : \sigma')\Gamma', t_1)t_2)[y := t'] \equiv\\
&(\tau(\Gamma(y : \sigma')\Gamma', t_1 t_2))[y := t'].
\end{aligned}
$$

□

6 Consistency in $\lambda_{\to\tau}$

Note that our canonical type operator τ, can be defined for some term without being defined for all its subterms. This can be seen from the following example:

Example 6.1 *Let $t \equiv (\lambda_{y:\sigma}.y)z$ where $z \not\equiv y$. Then $\tau(\emptyset, t) \equiv (\Pi_{y:\sigma}.\sigma)z \twoheadrightarrow_\beta \sigma[y := z] \equiv \sigma$ as $y \notin FV(\sigma)$. But $\tau(\emptyset, z)$ is not defined.*

For this reason, we introduce the relation $\vdash$ which takes a context and an expression (rather than only a term) and checks the well typedness of the expression. We take $\vdash$ to range over contexts and expressions rather than contexts and terms because a term might involve a type, and hence $\vdash$ also needs to check which are the types that are consistent within a term. When $\Gamma \vdash M$, we say that M is consistent in Γ.

First, we state the following convention:

Convention 6.2 *(WB-convention)*
All contexts Γ occurring in expressions $\Gamma \vdash M$ are well-behaved.

The relation $\vdash$ is defined as follows:

Definition 6.3 *($\vdash$)*

1. $$\frac{}{\Gamma \vdash \gamma}$$

2. $$\frac{\Gamma \vdash \sigma}{\Gamma(y : \sigma)\Gamma' \vdash y}$$

3. $$\frac{\Gamma \vdash \sigma \qquad \Gamma(y : \sigma) \vdash M}{\Gamma \vdash \pi_{y:\sigma}.M} \qquad \textit{with } \pi \equiv \lambda \textit{ if } M \in T \textit{ and } \pi \equiv \Pi \textit{ if } M \in \mathcal{T}$$

4. $$\frac{\Gamma \vdash t \qquad \Gamma \vdash t' \qquad \tau(\Gamma, t) \twoheadrightarrow_\beta \Pi_{y:\sigma_1}.\sigma_2 \qquad \tau(\Gamma, t') \twoheadrightarrow_\beta \sigma_1}{\Gamma \vdash tt'}$$

5. $$\frac{\Gamma \vdash \sigma \qquad \Gamma \vdash t \qquad \sigma \twoheadrightarrow_\beta \Pi_{y:\sigma_1}.\sigma_2 \qquad \tau(\Gamma, t) \twoheadrightarrow_\beta \sigma_1}{\Gamma \vdash \sigma t}$$

By this definition, we rule out Example 6.1. In fact $\emptyset \not\vdash t$ where $t \equiv (\lambda_{y:\sigma}.y)z$.

Remark 6.4 Note that in 2 and 3 above, $y \notin dom(\Gamma)$ and $y \notin FV(\sigma)$ by the WB-convention. Also, $y \notin FV(\Gamma)$. Recall moreover that we identify α-equal expressions. For example, if $\Gamma \vdash \pi_{y:\sigma}.M$, then also $\Gamma \vdash \pi_{z:\sigma}.M[y := z]$ for $z \notin FV(\sigma) \cup FV(M)$.

Example 6.5 The following can be derived in $\lambda_{\rightarrow\tau}$. (We use $\sigma_1 \rightarrow \sigma_2$ as an abbreviation for $\Pi_{y:\sigma_1}.\sigma_2$ in the case that $y \notin FV(\sigma_2)$.) Let σ, σ' and $\sigma'' \in \mathcal{T}$, let moreover

$$\begin{aligned}
&\Gamma_1 \equiv y_1 : \sigma \rightarrow \sigma', y_2 : \sigma' \rightarrow \sigma'', y_3 : \sigma,\\
&\Gamma_2 \equiv y_1 : \sigma \rightarrow \sigma', y_2 : \sigma' \rightarrow \sigma'',\\
&\Gamma_3 \equiv y_1 : \sigma \rightarrow \sigma'
\end{aligned}$$

Then,

$$\begin{array}{ll}
\Gamma_1 \vdash y_1 y_3 & \text{with } \tau(\Gamma_1, y_1 y_3) \equiv (\sigma \rightarrow \sigma')y_3 \twoheadrightarrow_\beta \sigma', \\
\Gamma_1 \vdash y_2(y_1 y_3) & \text{with } \tau(\Gamma_1, y_2(y_1 y_3)) \equiv (\sigma' \rightarrow \sigma'')(y_1 y_3) \twoheadrightarrow_\beta \sigma'', \\
\Gamma_2 \vdash \lambda_{y_3:\sigma}.y_2(y_1 y_3) & \text{with } \tau(\Gamma_2, \lambda_{y_3:\sigma}.y_2(y_1 y_3)) \equiv \\
& \Pi_{y_3:\sigma}.(\sigma' \rightarrow \sigma'')(y_1 y_3) \twoheadrightarrow_\beta \sigma \rightarrow \sigma'', \\
\Gamma_1 \vdash \lambda_{y_2:\sigma' \rightarrow \sigma''}.\lambda_{y_3:\sigma}.y_2(y_1 y_3) & \text{with } \tau(\Gamma_1, \lambda_{y_2:\sigma' \rightarrow \sigma''}.\lambda_{y_3:\sigma}.y_2(y_1 y_3)) \equiv \\
& \Pi_{y_2:\sigma' \rightarrow \sigma''}.\Pi_{y_3:\sigma}.(\sigma' \rightarrow \sigma'')(y_1 y_3) \twoheadrightarrow_\beta \\
& (\sigma' \rightarrow \sigma'') \rightarrow (\sigma \rightarrow \sigma''), \\
\vdash \lambda_{y_1:\sigma \rightarrow \sigma'}.\lambda_{y_2:\sigma' \rightarrow \sigma''}.\lambda_{y_3:\sigma}.y_2(y_1 y_3) & \text{with } \tau(\emptyset, \lambda_{y_1:\sigma \rightarrow \sigma'}.\lambda_{y_2:\sigma' \rightarrow \sigma''}.\lambda_{y_3:\sigma}.y_2(y_1 y_3)) \equiv \\
& \Pi_{y_1:\sigma \rightarrow \sigma'}.\Pi_{y_2:\sigma' \rightarrow \sigma''}.\Pi_{y_3:\sigma}.(\sigma' \rightarrow \sigma'')(y_1 y_3) \twoheadrightarrow_\beta \\
& (\sigma \rightarrow \sigma') \rightarrow ((\sigma' \rightarrow \sigma'') \rightarrow (\sigma \rightarrow \sigma''))
\end{array}$$

The following lemma relates $\vdash$ and τ.

Lemma 6.6 *(Well-typedness of consistent terms)*
For every $t \in \mathcal{T}$, $\Gamma \in CONS$, if $\Gamma \vdash t$ then $\vdash \tau(\Gamma, t)$.

Proof: *By induction on $\Gamma \vdash t$.*

- *If $t \equiv y$, then $\Gamma \vdash t$ comes from Definition 6.3, clause 2, and $\tau(\Gamma, t) \equiv \sigma$.*
- *If $t \equiv \lambda_{y:\sigma}.t'$, then $\Gamma \vdash t$ comes from Definition 6.3, clause 3, and the induction hypothesis holds for $\Gamma(y : \sigma) \vdash t'$. Hence we get from IH that $(\exists \sigma')[\tau(\Gamma(y : \sigma), t') \equiv \sigma']$. Now, $\tau(\Gamma, \lambda_{y:\sigma}.t') \equiv \Pi_{y:\sigma}.\tau(\Gamma(y : \sigma), t') \equiv \Pi_{y:\sigma}.\sigma'$.*
- *If $t \equiv t_1 t_2$, then $\Gamma \vdash t_1 t_2$ comes from Definition 6.3, clause 4, and the induction hypothesis holds for $\Gamma \vdash t_1$ and $\Gamma \vdash t_2$. Hence, as $\Gamma \vdash t_1$, we have $(\exists \sigma_1)[\tau(\Gamma, t_1) \equiv \sigma_1]$. Now, $\tau(\Gamma, t_1 t_2) \equiv \tau(\Gamma, t_1)t_2 \equiv \sigma_1 t_2$.*

□

Now if we go back to the previous section, we see that, even though τ satisfied some of the desirable lemmas such as weakening and substitution, other lemmas that are important in type theory are not satisfied by τ. For example, there are no restrictions on the free term variables used in a term. Moreover, the type of a term is not necessarily "preserved" when the term is reduced. The use of the derivation rules of Definition 6.3 give more satisfying results: see the Basis Lemma and the Subject Reduction Lemma, which follow below.

Lemma 6.7 *(Basis Lemma)*
If $\Gamma \vdash M$ then $FV(M) \cap V \subseteq dom(\Gamma)$[6].

Proof: *By induction on $\Gamma \vdash M$.*

- *The basic case (clauses 1 and 2 of Definition 6.3) is trivial.*
- *If $\Gamma \vdash \pi_{y:\sigma}.t$ is the result of clause 3 then (induction:) $FV(t) \cap V \subseteq dom(\Gamma(y:\sigma))$ and $FV(\sigma) \cap V \subseteq dom(\Gamma)$, so $FV(\pi_{y:\sigma}.t) \cap V = (FV(\sigma) \cap V) \cup ((FV(t) \setminus \{y\}) \cap V) \subseteq dom(\Gamma) \cup dom(\Gamma) = dom(\Gamma)$.*
- *Clauses 4 and 5 of Definition 6.3 are also trivial to check using the Induction Hypotheses.*

$\square$

Note that this does not hold for $\vdash \tau(\Gamma, M)$ instead of $\Gamma \vdash M$. For example, $\tau((y:\sigma), yz) \equiv \sigma z$ but $FV(yz) \cap V \not\subseteq dom(\Gamma)$.

Corollary 6.8
If $\Gamma \vdash t$, then $FV(\Gamma, t) \cap V \subseteq FV(\Gamma) \cap V$.

Proof: *Use that $FV(\Gamma, t) = (FV(\Gamma) \cup FV(t)) \setminus dom(\Gamma)$.*

Lemma 6.9 *(Generation Lemma)*
For all expressions M, for all contexts Γ, if $\Gamma \vdash M$ then:

1. *If $M \equiv y$ then $(\exists \Gamma', \Gamma'')[\Gamma \equiv \Gamma'(y : \tau(\Gamma, y))\Gamma'']$ such that $y \notin dom(\Gamma'')$*
2. *If $M \equiv \pi_{y:\sigma}.M'$ then $\Gamma \vdash \sigma$ and $\Gamma(y:\sigma) \vdash M'$*
3. *If $M \equiv t_1 t_2$ then $\Gamma \vdash t_1$, $\Gamma \vdash t_2$ and $(\exists \sigma_1, \sigma_2, y)[\tau(\Gamma, t_1) \twoheadrightarrow_\beta \Pi_{y:\sigma_1}.\sigma_2$ and $\tau(\Gamma, t_2) \twoheadrightarrow_\beta \sigma_1]$*[7]
4. *If $M \equiv \sigma t$ then $\Gamma \vdash \sigma$, $\Gamma \vdash t$ and $(\exists \sigma_1, \sigma_2, y)[\sigma \twoheadrightarrow_\beta \Pi_{y:\sigma_1}.\sigma_2$ and $\tau(\Gamma, t) \twoheadrightarrow_\beta \sigma_1]$.*

Proof: *By cases on the derivation $\Gamma \vdash M$*

1. *If $M \equiv y$, then $\Gamma \vdash M$ comes from clause 2 of Definition 6.3, so $(\exists \sigma, \Gamma', \Gamma'')[\Gamma \equiv \Gamma'(y : \sigma)\Gamma'']$, with $y \notin dom(\Gamma'')$ by the WB-convention. Hence $\tau(\Gamma, y) \equiv \tau(\Gamma'(y:\sigma)\Gamma'', y) \equiv \sigma$.*
2. *If $M \equiv \pi_{y:\sigma}.M'$, then $\Gamma \vdash M$ comes from clause 3, so $\Gamma(y:\sigma) \vdash M'$ and $\Gamma \vdash \sigma$.*
3. *If $M \equiv t_1 t_2$, then $\Gamma \vdash M$ comes from clause 4. Hence $(\exists \sigma_1, \sigma_2, y)[\tau(\Gamma, t_1) \twoheadrightarrow_\beta \Pi_{y:\sigma_1}.\sigma_2 \wedge \tau(\Gamma, t_2) \twoheadrightarrow_\beta \sigma_1]$, $\Gamma \vdash t_1$ and $\Gamma \vdash t_2$.*
4. *If $M \equiv \sigma t$, then $\Gamma \vdash M$ comes from clause 5 and the proof is similar to that of clause 4.*

$\square$

[6]Note that if we replace $\Gamma \vdash M$ by $\tau(\Gamma, t) \equiv \sigma$ only, then we don't necessarily get $FV(t) \cap V \subseteq dom(\Gamma)$. Take for instance example 6.1 above.

[7]Note that we have conversion $\twoheadrightarrow_\beta$ and not equivalence $\equiv$.

The following lemma states that subexpressions of consistent expressions are also consistent.

Lemma 6.10 *(Correctness of subexpressions)*
If $\Gamma \vdash M$ *and* M' *is a subexpression of* M *then* $(\exists \Gamma')[\Gamma\Gamma' \vdash M']$.
Proof: *By induction on* $\Gamma \vdash M$.

- *If clauses 1 or 2 apply then obvious.*
- *If clause 3 applies, i.e. if* $\Gamma \vdash \pi_{y:\sigma}.M$ *where* $\Gamma \vdash \sigma$ *and* $\Gamma(y:\sigma) \vdash M$, *then*
  - *case* $M' \equiv y$: *As* $\Gamma \vdash \sigma$, *then we get from clause 2, Definition 6.3, that* $\Gamma(y:\sigma) \vdash y$. *(Note that* $WB(\Gamma(y:\sigma))$*).*
  - *case* M' *is a subexpression of* M, *then use IH on* $\Gamma(y:\sigma) \vdash M$.
  - *case* M' *is a subexpression of* σ, *then use IH on* $\Gamma \vdash \sigma$.
- *If clause 4 applies, i.e. if* $t \equiv t_1 t_2$ *then* t *is a subexpression of* t_1 *or* t *is a subexpression of* t_2. *But* $\Gamma \vdash t$ *comes from amongst other things,* $\Gamma \vdash t_1$ *and* $\Gamma \vdash t_2$. *Now from the induction hypothesis the rest follows.*
- *If clause 5 applies, then use same argument as that of clause 4.*

$\square$

With the help of this lemma, we can prove the following:

Lemma 6.11 *If* $\Gamma \vdash M$, *then* M *fulfills the variable condition.*
Proof: *By induction on the derivation of* $\Gamma \vdash M$.

- *If clauses 1 or 2 apply then obvious.*
- *If clause 3 applies, then* M *fulfills the variable condition by induction. Now* $y \notin FV(\sigma)$ *since* $WB(\Gamma(y:\sigma))$ *by the WB-convention, so also* $\pi_{y:\sigma}.M$ *fulfills the variable convention.*
- *If clause 4 applies, i.e. if* $t \equiv t_1 t_2$, *then* $\Gamma \vdash t_1$ *and* $\Gamma \vdash t_2$, *and both* t_1 *and* t_2 *fulfill the variable condition by induction. By the Basis Lemma, all free term-variables in both* t_1 *and* t_2 *are elements of* $dom(\Gamma)$. *Now assume that* y *is free in* t_1 *and bound in* t_2. *Then* y *occurs in a subexpression* $\lambda_{y:\sigma}.t'$ *of* t_2. *By Correctness of Subexpressions,* $\Gamma\Gamma' \vdash \lambda_{y:\sigma}.t'$. *Hence, by the Generation Lemma,* $\Gamma\Gamma'(y:\sigma) \vdash t'$. *But since* y *is free in* t_1, $y \in dom(\Gamma)$ *and thus* $\Gamma\Gamma'(y:\sigma)$ *is not well-behaved, contradicting the WB-convention.*
- *If clause 5 applies, then use the same argument as that of clause 4.*

$\square$

The following lemma uses the well-behavedness of contexts in a derivation of $\Gamma \vdash M$.

Lemma 6.12 *(Weakening)*
If $\Gamma \vdash M$, $\Gamma \sqsubseteq \Gamma'$ *and* $WB(\Gamma')$, *then* $\Gamma' \vdash M$.
Proof: *By induction on the derivation of* $\Gamma \vdash M$, *Definition 6.3. First, rename bound variables in* M *such that* $dom(\Gamma') \cap BV(M) = \emptyset$ *and* $FV(\Gamma') \cap BV(M) = \emptyset$ *(see Remark 6.4). It is sufficient to prove the lemma for the case* $\Gamma \sqsubseteq^1 \Gamma'$.

- *Basic case where $\Gamma \vdash \gamma$ is obvious.*
- *Assume $\Gamma \equiv \Gamma_1(y:\sigma)\Gamma_2$ and $\Gamma_1(y:\sigma)\Gamma_2 \vdash y$ comes from $\Gamma_1 \vdash \sigma$. Then*
  - *either $\Gamma' \equiv \Gamma_1'(y:\sigma)\Gamma_2$ with $\Gamma_1 \equiv \Gamma_1''\Gamma_1'''$ and $\Gamma_1' \equiv \Gamma_1''(z:\sigma')\Gamma_1'''$,*
  - *or $\Gamma' \equiv \Gamma_1(y:\sigma)\Gamma_2'$ with $\Gamma_2 \equiv \Gamma_2''\Gamma_2'''$ and $\Gamma_2' \equiv \Gamma_2''(z:\sigma')\Gamma_2'''$.*

  In the first case, by induction from $\Gamma_1 \vdash \sigma$: $\Gamma_1' \vdash \sigma$ (note that $WB(\Gamma_1')$ by Corollary 4.32, part 1). So $\Gamma' \vdash y$.
  In the second case, $\Gamma' \vdash y$ follows immediately from $\Gamma_1 \vdash \sigma$.
- *Assume $\Gamma \equiv \Gamma_1\Gamma_2 \sqsubseteq^1 \Gamma_1(z:\sigma')\Gamma_2 \equiv \Gamma'$, $\Gamma \vdash \pi_{y:\sigma}.M$ (for $y \not\equiv z$ and $y \notin FV(\sigma')$)*[8] *and comes from clause 3, so $\Gamma \vdash \sigma$ and $\Gamma(y:\sigma) \vdash M$. Now $\Gamma(y:\sigma) \sqsubseteq^1 \Gamma'(y:\sigma)$ by Lemma 4.15. Moreover, $WB(\Gamma'(y:\sigma))$ as $WB(\Gamma(y:\sigma))$ and $y \not\equiv z$ and $y \notin FV(\sigma')$. So, by induction: $\Gamma' \vdash \sigma$ and $\Gamma'(y:\sigma) \vdash M$. Hence $\Gamma' \vdash \pi_{y:\sigma}.M$.*
- *If $\Gamma \vdash Mt$ comes from $\Gamma \vdash M$, $\Gamma \vdash t$ and clauses 4 or 5, use IH to get $\Gamma' \vdash M$ and $\Gamma' \vdash t$. Moreover, use Lemma 5.6 to deduce $\tau(\Gamma',t) \equiv \tau(\Gamma,t)$ and if case applies to deduce $\tau(\Gamma',M) \equiv \tau(\Gamma,M)$. Hence $\Gamma' \vdash Mt$.*

$\square$

The following lemmas are needed in some of the proofs:

Lemma 6.13 *If $WB(\Gamma)$ and $\Gamma\lceil X \vdash M$ then $\Gamma\lceil X' \vdash M$ for any $X' \supseteq X$.*
Proof: *$WB(\Gamma) \Rightarrow WB(\Gamma\lceil X')$ by Lemma 4.31. Furthermore, as $\Gamma\lceil X \le \Gamma\lceil X' \le \Gamma$ then by Corollary 4.36, $\Gamma\lceil X \sqsubseteq \Gamma\lceil X'$. Now using Lemma 6.12 we get $\Gamma\lceil X' \vdash M$.* $\square$

Remark 6.14 In the proofs of $\Gamma \vdash M$ below, we only show $WB(\Gamma)$ when it is not obvious that Γ is well-behaved. Otherwise, we don't mention anything about $WB(\Gamma)$.

The following lemma is important. It states that if t is consistent within a well behaved context Γ, then the type of t is also consistent in Γ.

Lemma 6.15 *(Correctness of types)*
If $\Gamma \vdash t$, then $\Gamma \vdash \tau(\Gamma,t)$.
Proof: *(Note that $\uparrow \tau(\Gamma,t)$ by Lemma 6.6.) Induction on $\Gamma \vdash t$.*

- *If $\Gamma \vdash t$ is $\Gamma'(y:\sigma)\Gamma'' \vdash y$ and comes from $\Gamma' \vdash \sigma$, then also $\Gamma'(y:\sigma)\Gamma'' \vdash \sigma$ by Lemmas 4.35 and 6.12 so $\Gamma \vdash \sigma \equiv \tau(\Gamma,y)$.*
- *If $\Gamma \vdash \lambda_{y:\sigma}.t$ comes from clause 3 of Definition 6.3, then $\Gamma(y:\sigma) \vdash t$ and $\Gamma \vdash \sigma$. By IH, $\Gamma(y:\sigma) \vdash \tau(\Gamma(y:\sigma),t)$. Hence $\Gamma \vdash \Pi_{y:\sigma}.\tau(\Gamma(y:\sigma,t)) \equiv \tau(\Gamma,\lambda_{y:\sigma}.t)$.*
- *If $\Gamma \vdash t_1t_2$ comes from clause 4 of Definition 6.3, then*
  $\Gamma \vdash t_1, \Gamma \vdash t_2$ and $(\exists\sigma_1,\sigma_2,y)[\tau(\Gamma,t_1) \twoheadrightarrow_\beta \Pi_{y:\sigma_1}.\sigma_2 \wedge \tau(\Gamma,t_2) \twoheadrightarrow_\beta \sigma_1]$.
  Hence from IH, $\Gamma \vdash \tau(\Gamma,t_1)$. Now $\tau(\Gamma,t_1t_2) \equiv \tau(\Gamma,t_1)t_2$.
  Using clause 5 of Definition 6.3, we get that $\Gamma \vdash \tau(\Gamma,t_1)t_2$.

[8]Note here that this condition is necessary. For example, $\emptyset \vdash \lambda_{y:\sigma}.y$ but $(y:\sigma) \not\vdash \lambda_{y:\sigma}.y$. Furthermore, $\emptyset \vdash \lambda_{y:\sigma}.y$ but $(z:\sigma y) \not\vdash \lambda_{y:\sigma}.y$. (See Remark 6.4.) The condition is satisfied since we started the proof with renaming bound variables in M.

□

The following lemma states that if M is consistent in a context Γ then it is consistent in the restriction of the context to the free variables of M and to those of its type (if applicable). In other words, if $\Gamma \vdash M$ then $\Gamma' \vdash M$ where Γ' is Γ from which statements S not relevant to the free variables of M or the free variables of the context to the right of M are removed.

Lemma 6.16
If $\Gamma(y:\sigma)\Gamma' \vdash M, y \notin FV(\Gamma')$ and $y \notin FV(M)$, then

1. *$\Gamma\Gamma' \vdash M$ and*
2. *If $\uparrow \tau(\Gamma(y:\sigma)\Gamma', M)$, then $\downarrow \tau(\Gamma\Gamma', M)$ and $\tau(\Gamma(y:\sigma)\Gamma', M) \equiv \tau(\Gamma\Gamma', M)$.*

Proof: *Simultaneously, by induction on $\Gamma(y:\sigma)\Gamma' \vdash M$.*

- *If $M \equiv \gamma$ then obvious.*
- *If $\Gamma_1(z:\sigma')\Gamma_2 \vdash z$ comes from $\Gamma_1 \vdash \sigma'$, then:*
  - *Case $\Gamma_1 \equiv \Gamma_1'(y:\sigma)\Gamma_1''$; by IH, $\Gamma_1'\Gamma_1'' \vdash \sigma'$, hence $\Gamma_1'\Gamma_1''(z:\sigma')\Gamma_2 \vdash z$. Moreover, $\tau(\Gamma_1'\Gamma_1''(z:\sigma')\Gamma_2, z) \equiv \tau(\Gamma_1(z:\sigma')\Gamma_2, z) \equiv \sigma'$.*
  - *Case $\Gamma_2 \equiv \Gamma_2'(y:\sigma)\Gamma_2''$, then immediately from $\Gamma_1 \vdash \sigma'$, $\Gamma_1(z:\sigma')\Gamma_2'\Gamma_2'' \vdash z$ and $\tau(\Gamma_1(z:\sigma')\Gamma_2'(y:\sigma)\Gamma_2'', z) \equiv \tau(\Gamma_1(z:\sigma')\Gamma_2'\Gamma_2'', z) \equiv \sigma'$.*
  - *Case $(z:\sigma') \equiv (y:\sigma)$ is impossible since $y \notin FV(M)$.*
- *If $\Gamma(y:\sigma)\Gamma' \vdash \pi_{z:\sigma'}.M$ comes from $\Gamma(y:\sigma)\Gamma' \vdash \sigma'$ and $\Gamma(y:\sigma)\Gamma'(z:\sigma') \vdash M$, then since $y \notin FV(\pi_{z:\sigma'}.M)$, $y \notin FV(\sigma')$ and $y \notin FV(M)$. Hence by IH: $\Gamma\Gamma' \vdash \sigma'$ and $\Gamma\Gamma'(z:\sigma') \vdash M$, so $\Gamma\Gamma' \vdash \pi_{z:\sigma'}.M$.*
  Moreover, $\tau(\Gamma(y:\sigma)\Gamma', \lambda_{z:\sigma'}.t) \equiv \Pi_{z:\sigma'}.\tau(\Gamma(y:\sigma)\Gamma'(z:\sigma'), t) \equiv^{IH}$ $\Pi_{z:\sigma'}.\tau(\Gamma\Gamma'(z:\sigma'), t) \equiv \tau(\Gamma\Gamma', \lambda_{z:\sigma'}.t)$.
- *If $\Gamma(y:\sigma)\Gamma' \vdash tt'$ comes from $\Gamma(y:\sigma)\Gamma' \vdash t$,*
  $\Gamma(y:\sigma)\Gamma' \vdash t'$, $\tau(\Gamma(y:\sigma)\Gamma', t) \twoheadrightarrow_\beta \Pi_{z:\sigma_1}.\sigma_2$ and $\tau(\Gamma(y:\sigma)\Gamma', t') \twoheadrightarrow_\beta \sigma_1$, then note that $y \notin FV(t)$ and $y \notin FV(t')$. Hence by IH: $\Gamma\Gamma' \vdash t$ and $\Gamma\Gamma' \vdash t'$. Moreover, $\tau(\Gamma\Gamma', t) \equiv^{IH} \tau(\Gamma(y:\sigma)\Gamma', t) \twoheadrightarrow_\beta \Pi_{z:\sigma_1}.\sigma_2$ and $\tau(\Gamma\Gamma', t') \equiv^{IH} \tau(\Gamma(y:\sigma)\Gamma', t') \twoheadrightarrow_\beta \sigma_1$, so $\Gamma\Gamma' \vdash tt'$.
  Also, $\tau(\Gamma(y:\sigma)\Gamma', tt') \equiv \tau(\Gamma(y:\sigma)\Gamma', t)t' \equiv^{IH} \tau(\Gamma\Gamma', t)t' \equiv \tau(\Gamma\Gamma', tt')$.
- *If $\Gamma(y:\sigma)\Gamma' \vdash \sigma t$ comes from $\Gamma(y:\sigma)\Gamma' \vdash \sigma$,*
  $\Gamma(y:\sigma)\Gamma' \vdash t$, $\sigma \twoheadrightarrow_\beta \Pi_{z:\sigma_1}.\sigma_2$ and $\tau(\Gamma(y:\sigma)\Gamma', t) \twoheadrightarrow_\beta \sigma_1$, then similar.

□

Corollary 6.17 *If $\Gamma \vdash M$ then there is a $\Gamma_M \sqsubseteq \Gamma$ such that $\Gamma_M \vdash M$ and for all Γ_1, Γ_2, y and σ with $\Gamma_M \equiv \Gamma_1(y:\sigma)\Gamma_2$, we have $y \in FV(\Gamma_2)$ or $y \in FV(M)$. Moreover, such a Γ_M is unique.*

Proof: *By induction on the length of* Γ. *Assume* $\Gamma \equiv (y_1 : \sigma_1) \dots (y_n : \sigma_n)$. *If for all* $i, y_i \in FV((y_{i+1} : \sigma_{i+1}) \dots (y_n : \sigma_n))$ *or* $y_i \in FV(M)$, *then we are ready. Otherwise, let* $(y_i : \sigma_i)$ *be the last statement in* Γ *such that* $y_i \notin FV((y_{i+1} : \sigma_{i+1}) \dots (y_n : \sigma_n))$ *and* $y_i \notin FV(M)$. *Then by Lemma 6.16,* $(y_1 : \sigma_1) \dots (y_{i-1} : \sigma_{i-1})(y_{i+1} : \sigma_{i+1}) \dots (y_n : \sigma_n) \vdash M$, *and we can apply induction. We leave it to the reader to show the uniqueness of* Γ_M. □

Compare this with Corollary 5.9 and note that $\Gamma_1(y : \sigma)\Gamma_2$ of the present corollary needs not only $y \in FV(M)$ but also $y \in FV(\Gamma_2)$. This is because our Γ's in $\Gamma \vdash M$ check types as well as terms. τ however only deals with terms.

Definition 6.18 *If* $\Gamma \vdash M$, *then* Γ_M *is the (unique) context described in the Corollary 6.17. We call* Γ_M *the context* relevant *for* $\Gamma \vdash M$. *When* Γ *and* Γ_M *are the same, we simply say* Γ *is relevant for* M.

Now the following lemma shows that consistency accommodates substitution.

Lemma 6.19 *($\vdash$-substitution Lemma)*
If $\tau(\Gamma, t) =_\beta \sigma, \Gamma \vdash t, \Gamma(y : \sigma)\Gamma' \vdash M$, *then* $\Gamma\Gamma'[y := t] \vdash M[y := t]$. *(Note that* $y \notin FV(\Gamma)$ *by the WB-convention.)*

Proof: *By induction on* $\Gamma(y : \sigma)\Gamma' \vdash M$.
Remark: as $\Gamma(y : \sigma)\Gamma'$ *is well-behaved, then from Remark 6.4,* $y \notin (FV(\Gamma) \cup FV(\sigma))$. *Moreover,*
as $\Gamma \vdash t$ *then* $FV(t) \cap V \subseteq dom(\Gamma)$ *by Lemma 6.7.*
Hence $FV(t) \cap dom(\Gamma') = \emptyset$. *Therefore,* $\Gamma\Gamma'[y := t]$ *is well-behaved.*

- *If* $M \equiv \gamma$ *then obvious.*
- *If* $\Gamma(y : \sigma)\Gamma' \vdash y$ *comes from* $\Gamma \vdash \sigma$, *then note that* $\Gamma \sqsubseteq \Gamma\Gamma'[y := t]$ *from Corollary 4.36. Now,* $\Gamma\Gamma'[y := t] \vdash t$, *using Lemma 6.12 and the remark in the beginning of this proof.*
- *If* $\Gamma_1(z : \sigma')\Gamma_2(y : \sigma)\Gamma' \vdash z$ *comes from* $\Gamma_1 \vdash \sigma'$, *then use the same remark.*
- *If* $\Gamma(y : \sigma)\Gamma_1(z : \sigma')\Gamma_2 \vdash z$ *comes from* $\Gamma(y : \sigma)\Gamma_1 \vdash \sigma'$ *then* $\Gamma\Gamma_1[y := t] \vdash \sigma'[y := t]$ *by IH.*
  Hence, $\Gamma\Gamma_1[y := t](z : \sigma'[y := t])\Gamma_2[y := t] \vdash z$ *by Definition 6.3 and the remark above.*
- *If* $\Gamma(y : \sigma)\Gamma' \vdash \lambda_{z:\sigma'}.M$ *comes from* $\Gamma(y : \sigma)\Gamma' \vdash \sigma'$ *and* $\Gamma(y : \sigma)\Gamma'(z : \sigma') \vdash M$ *then by IH,* $\Gamma\Gamma'[y := t] \vdash \sigma'[y := t]$ *and* $\Gamma\Gamma'[y := t](z : \sigma'[y := t]) \vdash M[y := t]$.
  Hence, $\Gamma\Gamma'[y := t] \vdash \lambda_{z:\sigma'[y:=t]}.M[y := t]$ *by Definition 6.3.*
- *If* $\Gamma(y : \sigma)\Gamma' \vdash t_1 t_2$ *comes from* $\Gamma(y : \sigma)\Gamma' \vdash t_1, \Gamma(y : \sigma)\Gamma' \vdash t_2, \tau(\Gamma(y : \sigma)\Gamma', t_1) \twoheadrightarrow_\beta \Pi_{y':\sigma_1}.\sigma_2$ *and* $\tau(\Gamma(y : \sigma)\Gamma', t_2) \twoheadrightarrow_\beta \sigma_1$ *then (take* $y' \not\equiv y$*):*
  - *by IH,* $\Gamma\Gamma'[y := t] \vdash t_1[y := t]$ *and* $\Gamma\Gamma'[y := t] \vdash t_2[y := t]$.
  - $\tau(\Gamma\Gamma'[y := t], t_1[y := t]) =_\beta \tau(\Gamma(y : \sigma)\Gamma', t_1)[y := t]$ *by Lemma 5.10*[9]
    $=_\beta \Pi_{y':\sigma_1}.\sigma_2[y := t]$ *by Corollary 3.20*
  - $\tau(\Gamma\Gamma'[y := t], t_2[y := t]) =_\beta \tau(\Gamma(y : \sigma)\Gamma', t_2)[y := t] =_\beta \sigma_1[y := t]$.

[9]Note that $y \notin FV(t)$ as $y \notin dom(\Gamma)$ by Lemma 6.7.

Hence by Corollary 3.19, $\exists \sigma_1', \sigma_2'$ such that
$\tau(\Gamma\Gamma'[y := t], t_1[y := t]) \twoheadrightarrow_\beta \Pi_{y:\sigma_1'}.\sigma_2'$ *and* $\tau(\Gamma\Gamma'[y := t], t_2[y := t]) \twoheadrightarrow_\beta \sigma_1'$.
Hence, $\Gamma\Gamma'[y := t] \vdash (t_1 t_2)[y := t]$.

- *If clause 5 of Definition 6.3 applies, then similar to above.*

□

The following lemma is not interesting on its own, but is needed to show the subject reduction theorem.

Lemma 6.20

1. *If* $\Gamma \vdash (\pi_{y:\sigma}.M)t$ *then* $\Gamma \vdash M[y := t]$.
2. *If* $\Gamma \vdash (\lambda_{y:\sigma}.t_1)t_2$ *then* $\tau(\Gamma, (\lambda_{y:\sigma}.t_1)t_2) =_\beta \tau(\Gamma, t_1[y := t_2])$.[10]

Proof:

1. *As* $\Gamma \vdash (\pi_{y:\sigma}.M)t$ *then by Lemma 6.9 we get that* $\tau(\Gamma, t) =_\beta \sigma$ *as follows:*
   - *Case* $M \equiv t_1$ *and* $\pi \equiv \lambda$:
     $\tau(\Gamma, \lambda_{y:\sigma}.t_1) \equiv \Pi_{y:\sigma}.\tau(\Gamma(y : \sigma), t_1) \twoheadrightarrow_\beta \Pi_{y:\sigma_1}.\sigma_2$ *and*
     $\tau(\Gamma, t) \twoheadrightarrow_\beta \sigma_1$. *Hence* $\sigma \twoheadrightarrow_\beta \sigma_1$ *and so* $\tau(\Gamma, t) =_\beta \sigma$.
   - *Case* $M \equiv \sigma'$ *and* $\pi \equiv \Pi$:
     $\Pi_{y:\sigma}.\sigma' \twoheadrightarrow_\beta \Pi_{y:\sigma_1}.\sigma_2$ *and*
     $\tau(\Gamma, t) \twoheadrightarrow_\beta \sigma_1$. *Hence* $\sigma \twoheadrightarrow_\beta \sigma_1$ *and so* $\tau(\Gamma, t) =_\beta \sigma$.

   Moreover, $\Gamma \vdash (\pi_{y:\sigma}.M)t$ *then by Lemma 6.9, we get* $\Gamma \vdash \pi_{y:\sigma}.M$ *and* $\Gamma \vdash t$.

   We apply Lemma 6.9 again to $\Gamma \vdash \pi_{y:\sigma}.M$ *and get:* $\Gamma(y : \sigma) \vdash M$. *Now,* $\Gamma \vdash t, \tau(\Gamma, t) =_\beta \sigma$, *and* $\Gamma(y : \sigma) \vdash M$. *So from Lemma 6.19, we get* $\Gamma \vdash M[y := t]$.

2. *First, we prove that* $\tau(\Gamma, t_2) =_\beta \sigma$ *in the same way as above.*
   Furthermore, $WB(\Gamma(y : \sigma))$ *as* $y \notin dom(\Gamma) \cup FV(\sigma)$ *and* $WB(\Gamma)$.
   Moreover, $y \notin FV(t_2)$ *by VC.*
   Hence, by Lemma 5.10, $\tau(\Gamma, t_1[y := t_2]) =_\beta \tau(\Gamma(y : \sigma), t_1)[y := t_2]$. *Also,*
   $\tau(\Gamma, (\lambda_{y:\sigma}.t_1)t_2) \equiv (\Pi_{y:\sigma}.\tau(\Gamma(y : \sigma), t_1))t_2 \rightarrow_\beta \tau(\Gamma(y : \sigma), t_1)[y := t_2]$.
   Hence $\tau(\Gamma, (\lambda_{y:\sigma}.t_1)t_2) =_\beta \tau(\Gamma, t_1[y := t_2])$.

□

In what follows, if $\tau(\Gamma, M)$ is undefined (in particular if $M \in \mathcal{T}$), we take $FV(\tau(\Gamma, M))$ to be empty.

Lemma 6.21 *If* $z \in FV(M) \cup FV(\tau(\Gamma(z : \sigma)\Gamma', M))$ *and* $\Gamma(z : \sigma)\Gamma' \vdash M$ *then* $\Gamma \vdash \sigma$.
Proof: *By induction on* $\Gamma(z : \sigma)\Gamma' \vdash M$.

[10]It is this β-convertibility which will disable proving Theorem 6.24 and its first corollary using $\twoheadrightarrow_\beta$ instead of $=_\beta$. (See the final case of the proof of Theorem 6.24.)

- $\Gamma(z:\sigma)\Gamma' \vdash \gamma$ *obvious.*
- *If* $\Gamma(z:\sigma)\Gamma' \vdash z$ *comes from* $\Gamma \vdash \sigma$ *obvious.*
- $\Gamma_1(y:\sigma')\Gamma_2(z:\sigma)\Gamma' \vdash y$ *is not applicable as* $FV(\sigma') \cap z = \emptyset$.
- *If* $\Gamma(z:\sigma)\Gamma_1(y:\sigma')\Gamma_2 \vdash y$ *comes from* $\Gamma(z:\sigma)\Gamma_1 \vdash \sigma'$ *where* $z \in FV(\sigma')$*, then use IH.*
- *If* $\Gamma(z:\sigma)\Gamma' \vdash \pi_{y:\sigma'}.M$ *comes from* $\Gamma(z:\sigma)\Gamma' \vdash \sigma'$ *and* $\Gamma(z:\sigma)\Gamma'(y:\sigma') \vdash M$ *then*
  - *case* $z \in FV(\sigma')$ *then use IH.*
  - *case* $z \in FV(M)$ *then use IH.*
  - *case* $z \in FV(\tau(\Gamma(z:\sigma)\Gamma', \Pi_{y:\sigma'}.M)$*, then* $\pi \equiv \lambda$ *and* $z \in FV(\Pi_{y:\sigma'}.\tau(\Gamma(z:\sigma)\Gamma'(y:\sigma'), M)$*. If* $z \in FV(\sigma')$*: see above. If* $z \in FV(\tau(\Gamma(z:\sigma)\Gamma'(y:\sigma'), M))$*, then use IH.*
- *If* $M \equiv M't$ *then use IH.*

□

Lemma 6.22 *If* $\Gamma(z:\sigma)\Gamma' \vdash M$*,* $\Gamma(z:\sigma)\Gamma'$ *is relevant for* M *and* $z \in FV(\Gamma')$*, then* $\Gamma \vdash \sigma$*.*

Proof: *By induction on the length of* Γ'*.*
Assume $\Gamma' \equiv \Gamma_1(y:\sigma')\Gamma_2$ *and* $z \in FV(\sigma')$*.*

- *If* $y \in FV(M)$*, then by Lemma 6.21:* $\Gamma(z:\sigma)\Gamma_1 \vdash \sigma'$*. Again by Lemma 6.21:* $\Gamma \vdash \sigma$*.*
- *If* $y \in FV(\Gamma_2)$ *then by IH:* $\Gamma(z:\sigma)\Gamma_1 \vdash \sigma'$*. Hence by Lemma 6.21:* $\Gamma \vdash \sigma$*.*
- *The case* $y \notin FV(M)$ *and* $y \notin FV(\Gamma_2)$ *cannot occur since* $\Gamma(z:\sigma)\Gamma_1(y:\sigma')\Gamma_2$ *is relevant for* M*; see Definition 6.18.*

□

Corollary 6.23 *If* $\Gamma \vdash M$*,* Γ *is relevant for* M *and* $\Gamma \equiv \Gamma_1(z:\sigma)\Gamma_2$*, then* $\Gamma_1 \vdash \sigma$*.*

Proof: *As* Γ *is relevant for* M*, then either* $z \in FV(\Gamma_2)$ *or* $z \in FV(M)$*. In the first case use Lemma 6.22. In the second case use Lemma 6.21.* □

The following theorem is important. It shows that our notions of consistency and of typing are compatible with that of reduction.

Theorem 6.24 *(Subject and Context Reduction Theorem)*

1. *If* $M \to_\beta M'$ *and* $\Gamma \vdash M$ *then* $\Gamma \vdash M'$ *and if* $M \in T$ *then* $\tau(\Gamma, M) =_\beta \tau(\Gamma, M')$*.*
2. *If* $\Gamma \to_\beta \Gamma'$ *and* $\Gamma \vdash M$ *then* $\Gamma' \vdash M$*.*

Proof: *Simultaneously by induction on the derivation* $\Gamma \vdash M$*.*

- *case* $\Gamma \vdash \gamma$ *then 1 is obvious. Moreover, 2 is obvious as* $WB(\Gamma')$ *from Lemma 4.31, part 3.*
- *case* $\Gamma(y:\sigma)\Gamma' \vdash y$ *comes from* $\Gamma \vdash \sigma$ *then 1 is obvious. Moreover,*
  - *case* $\Gamma \rightarrow_\beta \Gamma''$ *then by IH* $\Gamma'' \vdash \sigma$ *and so* $\Gamma''(y:\sigma)\Gamma' \vdash y$.
  - *case* $\sigma \rightarrow_\beta \sigma'$ *then by IH* $\Gamma \vdash \sigma'$ *and so* $\Gamma(y:\sigma')\Gamma' \vdash y$.
  - *case* $\Gamma' \rightarrow_\beta \Gamma''$ *then* $\Gamma(y:\sigma)\Gamma'' \vdash y$
- *case* $\Gamma \vdash \pi_{y:\sigma}.M$ *comes from* $\Gamma \vdash \sigma$ *and* $\Gamma(y:\sigma) \vdash M$ *then*
  - *case* $\sigma \rightarrow_\beta \sigma'$ *then by IH,* $\Gamma \vdash \sigma'$ *and* $\Gamma(y:\sigma') \vdash M$ *and hence,* $\Gamma \vdash \pi_{y:\sigma'}M$.
    Furthermore, if $\pi = \lambda$ *and* $M \equiv t$ *then* $\tau(\Gamma, \lambda_{y:\sigma}.t) \equiv \Pi_{y:\sigma}.\tau(\Gamma(y:\sigma),t) =_\beta \Pi_{y:\sigma'}.\tau(\Gamma(y:\sigma'),t)$ *by Lemma 5.7.*
  - *case* $M \rightarrow_\beta M'$ *then by IH* $\Gamma(y:\sigma) \vdash M'$ *and hence* $\Gamma \vdash \pi_{y:\sigma}.M'$.
    Furthermore, if $\pi = \lambda$, $M \equiv t$ *and* $M' \equiv t'$ *then*
    $\tau(\Gamma, \lambda_{y:\sigma}.t) \equiv \Pi_{y:\sigma}.\tau(\Gamma(y:\sigma),t) =_\beta \Pi_{y:\sigma}.\tau(\Gamma(y:\sigma),t')$ *by IH.*

  Moreover by IH, $\Gamma' \vdash \sigma$ *and* $\Gamma'(y:\sigma) \vdash M$, *hence* $\Gamma' \vdash \pi_{y:\sigma}.M$
- *case* $\Gamma \vdash t_1 t$ *comes from* $\Gamma \vdash t_1, \Gamma \vdash t, \tau(\Gamma,t_1) \twoheadrightarrow_\beta \Pi_{y:\sigma_1}.\sigma_2$ *and* $\tau(\Gamma,t) \twoheadrightarrow_\beta \sigma_1$. *Assume* $t_1 \rightarrow_\beta t_2$, *then by IH,* $\Gamma \vdash t_2$ *and* $\tau(\Gamma,t_2) =_\beta \tau(\Gamma,t_1)$. *Hence, by Corollary 3.19,* $(\exists\sigma'_1,\sigma'_2)[\tau(\Gamma,t_2) \twoheadrightarrow_\beta \Pi_{y:\sigma'_1}.\sigma'_2$ *and* $\tau(\Gamma,t) \twoheadrightarrow_\beta \sigma'_1]$.
  Now, from $\tau(\Gamma,t_2) \twoheadrightarrow_\beta \Pi_{y:\sigma'_1}.\sigma'_2, \tau(\Gamma,t) \twoheadrightarrow_\beta \sigma'_1, \Gamma \vdash t$ *and* $\Gamma \vdash t_2$ *we get* $\Gamma \vdash t_2 t$.
  Furthermore, $\tau(\Gamma,t_1 t) \equiv \tau(\Gamma,t_1)t =_\beta \tau(\Gamma,t_2)t$ *by IH.*
  Moreover, by IH and Lemma 5.7, $\Gamma' \vdash t_1, \Gamma' \vdash t, \tau(\Gamma',t_1) =_\beta \tau(\Gamma,t_1)$ *and* $\tau(\Gamma',t) =_\beta \tau(\Gamma,t)$.

  Hence, by Corollary 3.19, $(\exists\sigma'_1,\sigma'_2)[\tau(\Gamma',t_1) \twoheadrightarrow_\beta \Pi_{y:\sigma'_1}.\sigma'_2$ *and* $\tau(\Gamma',t) \twoheadrightarrow_\beta \sigma'_1]$.
  Now, from $\tau(\Gamma',t_1) \twoheadrightarrow_\beta \Pi_{y:\sigma'_1}.\sigma'_2, \tau(\Gamma',t) \twoheadrightarrow_\beta \sigma'_1, \Gamma' \vdash t$ *and* $\Gamma' \vdash t_1$ *we get* $\Gamma' \vdash t_1 t$.
- *case* $t_1 \rightarrow_\beta t_2$ *and* $\Gamma \vdash M t_1$ *then similar to above.*
- *case* $\sigma' \rightarrow_\beta \sigma''$ *and* $\Gamma \vdash \sigma' t$ *then similar to above.*
- *case* $(\pi_{y:\sigma}.M)t_2 \rightarrow_\beta M[y:=t_2]$ *and* $\Gamma \vdash (\pi_{y:\sigma}.M)t_2$ *then from Lemma 6.20,*
  $\Gamma \vdash M[y:=t_2]$. *Moreover, if* $M \in \mathcal{T}$ *then again we use Lemma 6.20 to get* $\tau(\Gamma,(\lambda_{y:\sigma}.M)t_2) =_\beta \tau(\Gamma, M[y:=t_2])$.

  Furthermore,
  - *case* $M \equiv t$ *then* $\Gamma \vdash (\lambda_{y:\sigma}.t)t_2$ *comes from* $\Gamma \vdash \lambda_{y:\sigma}.t$, $\Gamma \vdash t_2$, $\tau(\Gamma,\lambda_{y:\sigma}.t) \twoheadrightarrow_\beta \Pi_{z:\sigma_1}.\sigma_2$ *and* $\tau(\Gamma,t_2) \twoheadrightarrow_\beta \sigma_1$. *Now the fact that*
    $\Gamma' \vdash (\lambda_{y:\sigma}.t)t_2$ *follows from* $\Gamma \vdash (\lambda_{y:\sigma}.t)t_2$ *has already been treated above (case* $\Gamma \vdash t_1 t$*).*
  - *case* $M \equiv \sigma'$ *then similar to above.*

□

Corollary 6.25 *If $M \twoheadrightarrow_\beta M'$ and $\Gamma \vdash M$ then $\Gamma \vdash M'$ and if $M \in T$ then $\tau(\Gamma, M) =_\beta \tau(\Gamma, M')$.*

Proof: *By induction on $M \twoheadrightarrow_\beta M'$ where $\Gamma \vdash M$.*

- *If $M \twoheadrightarrow_\beta M$ then obvious.*
- *If $M \twoheadrightarrow_\beta M'$ comes from $M \to_\beta M'$ and if $\Gamma \vdash M$ then use Theorem 6.24.*
- *If $M \twoheadrightarrow_\beta M'$ comes from $M \twoheadrightarrow_\beta M_1$ and $M_1 \twoheadrightarrow_\beta M'$ and if $\Gamma \vdash M$ then by IH, $\Gamma \vdash M_1$ and $\Gamma \vdash M'$.*
  Also if $\tau(\Gamma, M) \equiv \sigma$ then by IH, $\tau(\Gamma, M_1) =_\beta \tau(\Gamma, M)$. Similarly by IH, $\tau(\Gamma, M') =_\beta \tau(\Gamma, M_1)$. Hence $\tau(\Gamma, M) =_\beta \tau(\Gamma, M')$.

□

Note here that the version of this corollary for the case $\vdash$ is replaced by τ does not hold.

Example 6.26 Take $t \equiv (\lambda_{y:\sigma}.y)x$. Then $\nvdash \tau(\emptyset, (\lambda_{y:\sigma}.y)x)$ and $\tau(\emptyset, (\lambda_{y:\sigma}.y)x) \twoheadrightarrow_\beta \sigma x$ yet $\tau(\emptyset, y[y := x]) = \tau(\emptyset, x)$ is undefined. Note however that in the case of $(\lambda_{y:\sigma}.z)(\omega\omega) \to_\beta z$, we have that $\tau((z : \sigma'), (\lambda_{y:\sigma}.z)(\omega\omega)) \to_\beta \sigma' \equiv \tau((z : \sigma'), z)$.

Now the following lemma is obvious. It states that if M is consistent in a context Γ, then it is consistent in any reduct of Γ.

Corollary 6.27
If $\Gamma_1 \twoheadrightarrow_\beta \Gamma_2$ and $\Gamma_1 \vdash M$ then $\Gamma_2 \vdash M$ and if $M \in T$, then $\tau(\Gamma_1, M) =_\beta \tau(\Gamma_2, M)$.

Proof: *By induction on the number of one step reductions of $\Gamma_1 \twoheadrightarrow_\beta \Gamma_2$, using Theorem 6.24 and Lemma 5.7.* □

Here note that the version of this corollary replacing $\vdash$ by τ holds, as has been shown in Lemma 5.7.

Finally, the following lemma is very useful. It shows that if t and t' are consistent and β-equal, then their types are also equal.

Lemma 6.28 *(Unicity of types)*
If $t =_\beta t'$, $\Gamma \vdash t$ and $\Gamma \vdash t'$ then $\tau(\Gamma, t) =_\beta \tau(\Gamma, t')$.

Proof: *By Corollary 3.17, $(\exists t'')[(t \twoheadrightarrow_\beta t'') \wedge (t' \twoheadrightarrow_\beta t'')]$. From Corollary 6.25 we get that $\tau(\Gamma, t'') =_\beta \tau(\Gamma, t)$ and $\tau(\Gamma, t'') =_\beta \tau(\Gamma, t')$. Hence $\tau(\Gamma, t) =_\beta \tau(\Gamma, t')$.* □

Note that it is possible that $\Gamma \vdash M$ and $M =_\beta M'$ without $\Gamma \vdash M'$. For example, $(y : \gamma) \vdash y$ and $a \equiv (\lambda_{x:\gamma}.\lambda_{y:\gamma'}.x)y(\lambda_{x:\gamma''}.xx) =_\beta y$ but $(y : \gamma) \nvdash a$.

Up to here, we have shown that our calculus is attractive. Terms and types are treated alike and β-conversion is used with both forms of expressions. Church-Rosser holds for the calculus and all the desirable typing conditions are satisfied. The following table summarises these properties for τ and $\vdash$.

Table 6.29 *(Properties of τ and $\vdash$)*

	τ	$\vdash$
Subject-Reduction	*No*	*Yes*
Context-Reduction	*Yes*	*Yes*
Restriction	*Yes*	*Yes*
Substitution	*Yes*	*Yes*
Basis	*No*	*Yes*
Generation	—	*Yes*
Subexpressions	—	*Yes*
Weakening	*Yes*	*Yes*
Unicity of Types	—	*Yes*

Next, we will interpret Church's $\lambda_{\rightarrow}$ in our calculus showing that a term t of $\lambda_{\rightarrow}$ is consistent iff the type of t via τ converges to the type of the term in $\lambda_{\rightarrow}$. We will show moreover that all the work that one carries out in $\lambda_{\rightarrow}$ can also be carried out in $\lambda_{\rightarrow\tau}$. Moreover, $\lambda_{\rightarrow\tau}$ gives a unified treatment of types and terms. Such a treatment can generalise to many known typing systems.

7 The relation of $\lambda_{\rightarrow\tau}$ to $\lambda_{\rightarrow}$

7.1 Church's $\lambda_{\rightarrow}$ and its interpretation in $\lambda_{\rightarrow\tau}$

We start by presenting the system $\lambda_{\rightarrow}$. We present types of $\lambda_{\rightarrow}$, $(\mathcal{T}_{\rightarrow})$, terms of $\lambda_{\rightarrow}$, $(T_{\rightarrow})$ and the typing rules of $\lambda_{\rightarrow}$.

Definition 7.1 *($\lambda_{\rightarrow}$)*
We use the same object and meta level notation as in $\lambda_{\rightarrow\tau}$ and define types and terms as follows:

$$\mathcal{T}_{\rightarrow} = \mathcal{V} \mid (\mathcal{T}_{\rightarrow} \rightarrow \mathcal{T}_{\rightarrow})$$
$$T_{\rightarrow} = V \mid (\lambda_{V:\mathcal{T}_{\rightarrow}}.T_{\rightarrow}) \mid (T_{\rightarrow}T_{\rightarrow})$$

We define statements here similarly to the way we defined them in Definition 4.1. Furthermore, we take a basis (instead of a context) to be a set of statements where the subjects are variables which occur at most once. So we no longer insist on the idea of a context as an ordered set for this section. This is consistent with our assumption that in a well-behaved context, all subject variables occur at most once. The second condition of a well-behaved context which says that in $\Gamma(y:\sigma)\Gamma'$, $FV(\sigma) \cap dom((y:\sigma)\Gamma') = \emptyset$ is satisfied because from Remark 7.4 below, $FV(\sigma) \cap V = \emptyset$. In fact, bases are the $\lambda_{\rightarrow}$ version of well-behaved contexts (see Corollary 7.12). Even stronger, bases of $\lambda_{\rightarrow}$ correspond to the simply typed contexts of $\lambda_{\rightarrow\tau}$ (see Corollary 7.13).

We take $\mathcal{K}$ to be the collection of all bases of $\lambda_{\rightarrow}$ and use the same meta-notation for contexts. That is $\Gamma, \Gamma', \Gamma_1, \Gamma_2 \ldots$ will range over elements in $\mathcal{K}$.

Definition 7.2 *The typing rules of $\lambda_{\rightarrow}$ are the following:*

1. $\Gamma \vdash_{\lambda_{\rightarrow}} y : \sigma \qquad$ *if* $(y:\sigma) \in \Gamma$

2. $$\frac{\Gamma \vdash_{\lambda_{\rightarrow}} t : (\sigma \rightarrow \sigma') \qquad \Gamma \vdash_{\lambda_{\rightarrow}} t' : \sigma}{\Gamma \vdash_{\lambda_{\rightarrow}} tt' : \sigma'}$$

3. $$\frac{\Gamma(y:\sigma)\vdash_{\lambda_\to} t:\sigma'}{\Gamma\vdash_{\lambda_\to}(\lambda_{y:\sigma}.t):(\sigma\to\sigma')}$$

Example 7.3 The following can be derived in $\lambda_\to$ (cf. Example 6.5).
Let σ,σ' and $\sigma''\in\mathcal{T}_\to$. Let moreover,

$$\begin{aligned}
&\Gamma_1\equiv y_1:\sigma\to\sigma', y_2:\sigma'\to\sigma'', y_3:\sigma\\
&\Gamma_2\equiv y_1:\sigma\to\sigma', y_2:\sigma'\to\sigma''\\
&\Gamma_3\equiv y_1:\sigma\to\sigma'
\end{aligned}$$

Then,

$$\begin{aligned}
&\Gamma_1\vdash y_1y_3:\sigma'\\
&\Gamma_1\vdash y_2(y_1y_3):\sigma''\\
&\Gamma_2\vdash \lambda_{y_3:\sigma}.y_2(y_1y_3):\sigma\to\sigma''\\
&\Gamma_3\vdash \lambda_{y_2:\sigma'\to\sigma''}.\lambda_{y_3:\sigma}.y_2(y_1y_3):(\sigma'\to\sigma'')\to(\sigma\to\sigma'')\\
&\vdash \lambda_{y_1:\sigma\to\sigma'}.\lambda_{y_2:\sigma'\to\sigma''}.\lambda_{y_3:\sigma}.y_2(y_1y_3):(\sigma\to\sigma')\to((\sigma'\to\sigma'')\to(\sigma\to\sigma''))
\end{aligned}$$

Remark 7.4 *Note that for $\sigma\in\mathcal{T}_\to$, we have $FV(\sigma)\subseteq\mathcal{V}$ and $BV(\sigma)=\emptyset$.*

We define an interpretation function from $\lambda_\to$ to $\lambda_{\to\tau}$ as follows

Definition 7.5 *$\mathcal{I}:(\mathcal{T}_\to\cup\mathcal{T}_\to)\longrightarrow(\mathcal{T}\cup\mathcal{T})$ is defined as follows:*

1. $\mathcal{I}(\gamma)\equiv\gamma$
2. $\mathcal{I}(y)\equiv y$
3. $\mathcal{I}(\sigma\to\sigma')\equiv\Pi_{y:\mathcal{I}(\sigma)}.\mathcal{I}(\sigma')$ *where y is fresh.*
4. $\mathcal{I}(tt')\equiv\mathcal{I}(t)\mathcal{I}(t')$
5. $\mathcal{I}(\lambda_{y:\sigma}.t)\equiv\lambda_{y:\mathcal{I}(\sigma)}.\mathcal{I}(t)$

Definition 7.6 *We extend $\mathcal{I}$ to $\mathcal{K}$ as follows:*
$\mathcal{I}(\{(y_1:\sigma_1)\dots(y_n:\sigma_n)\})\equiv(y_1:\mathcal{I}(\sigma_1))\dots(y_n:\mathcal{I}(\sigma_n)))$, *in some order.*

Note that even though $\mathcal{I}(M)$ is well-defined, $\mathcal{I}(\Gamma)$ is not well-defined, because the elements of a set can be listed in many different orders. However, this does not affect our main results, as Theorem 7.8 below shows.

Definition 7.7

- *A well-behaved context Γ in $\lambda_{\to\tau}$ is called* permutable *if $\Gamma\equiv(y_1:\sigma_1)\dots(y_n:\sigma_n)$ with $y_i\notin FV(\{\sigma_1,\dots,\sigma_n\})$ for all i.*
- *A permutation of a well-behaved context $\Gamma\equiv(y_1:\sigma_1)\dots(y_n:\sigma_n)$ is a context $\Gamma'\equiv(y_{i_1}:\sigma_{i_1})\dots(y_{i_n}:\sigma_{i_n})$ such that $i_1,\dots,i_n$ is a permutation of $1,\dots,n$.*

Theorem 7.8 *Let Γ be a permutable context in $\lambda_{\to\tau}$ (and hence well-behaved) and let Γ' be a permutation of Γ. Then for all M such that $\Gamma\vdash M$, we have:*

1. $\Gamma' \vdash M$ *and*

2. *If* $\uparrow \tau(\Gamma, M)$ *then* $\uparrow \tau(\Gamma', M)$ *and* $\tau(\Gamma, M) \equiv \tau(\Gamma', M)$.

Proof: *By simultaneous induction on* $\Gamma \vdash M$. *The only non-trivial case is that* $\Gamma \vdash M$ *is* $\Gamma_1(y : \sigma)\Gamma_2 \vdash y$ *as a consequence of* $\Gamma_1 \vdash \sigma$. *It follows from Lemma 6.16 that* $\emptyset \vdash \sigma$ *(induction on the number of statements in* Γ_1*). Let* $\Gamma' \equiv \Gamma'_1(y : \sigma)\Gamma'_2$ *be a permutation of* $\Gamma_1(y : \sigma)\Gamma_2$. *Then* $\emptyset \vdash \sigma$ *implies* $\Gamma'_1 \vdash \sigma$, *by Lemma 6.12. Hence* $\Gamma' \vdash y$ *and* $\tau(\Gamma', y) \equiv \tau(\Gamma, y) \equiv \sigma$. □

7.2 Some useful machinery

In this section we present some lemmas and remarks which will be used in proving the main lemmas and theorems concerning the interpretation of $\lambda_{\rightarrow}$ in $\lambda_{\rightarrow\tau}$.

Lemma 7.9 *For* $M \in (\mathcal{T}_{\rightarrow} \cup \mathcal{T}_{\rightarrow})$, *we have* $FV(\mathcal{I}(M)) = FV(M)$.
Proof: *Obvious.* □

Remark 7.10 Note that in $\mathcal{I}(\sigma \rightarrow \sigma') \equiv \Pi_{y:\mathcal{I}(\sigma)}.\mathcal{I}(\sigma')$, $y \notin VAR(\mathcal{I}(\sigma)) \cup VAR(\mathcal{I}(\sigma'))$, from the condition that y is fresh.

Corollary 7.11 *For* $\sigma \in \mathcal{T}_{\rightarrow}$, $FV(\mathcal{I}(\sigma)) \subseteq \mathcal{V}$.
Proof: *Obvious using Remark 7.4.* □

Corollary 7.12 $\Gamma \in \mathcal{K} \Leftrightarrow WB(\mathcal{I}(\Gamma))$.
Proof: *Obvious.* □

Corollary 7.13 $\Gamma \in \mathcal{K} \Rightarrow \mathcal{I}(\Gamma)$ *is permutable.*
Proof: *Use Corollary 7.12 and Remark 7.10.* □

Lemma 7.14 *For any* $\sigma \in \mathcal{T}_{\rightarrow}$ *and* $\Gamma \in CONS$, *if* $WB(\Gamma)$ *then* $\Gamma \vdash \mathcal{I}(\sigma)$.
Proof: *By induction on* $\mathcal{T}_{\rightarrow}$.

- *If* σ *is* γ *then according to Definition 6.3, clause 1,* $\Gamma \vdash \gamma$.
- *Assume* σ *is* $\sigma_1 \rightarrow \sigma_2$ *where IH holds for* σ_1 *and*
  σ_2 *and* $\mathcal{I}(\sigma_1 \rightarrow \sigma_2) \equiv \Pi_{y:\mathcal{I}(\sigma_1)}.\mathcal{I}(\sigma_2)$ *where* y *is fresh. Then by IH,*
  $\Gamma \vdash \mathcal{I}(\sigma_1)$ *and* $\Gamma(y : \mathcal{I}(\sigma_1)) \vdash \mathcal{I}(\sigma_2)$.
  Now apply clause 3 of Definition 6.3 to $\Gamma \vdash \mathcal{I}(\sigma_1)$ *and* $\Gamma(y : \mathcal{I}(\sigma_1)) \vdash \mathcal{I}(\sigma_2)$, *to obtain* $\Gamma \vdash \mathcal{I}(\sigma_1 \rightarrow \sigma_2)$.

□

Corollary 7.15 *For any* $\sigma \in \mathcal{T}$, *for any permutable* $\Gamma \in CONS$, $\Gamma \vdash \mathcal{I}(\sigma)$.
Proof: *Use Lemma 7.14.* □

Lemma 7.16 *If $\sigma \in \mathcal{T}_{\rightarrow}$ and $\mathcal{I}(\sigma) \twoheadrightarrow_\beta \sigma'$ then $\mathcal{I}(\sigma) \equiv \sigma'$.*
Proof: *By induction on $\sigma \in \mathcal{T}_{\rightarrow}$*

- *Case $\sigma \equiv \gamma$ then obvious.*
- *Assume $\sigma \equiv \sigma_1 \rightarrow \sigma_2$ where IH holds for σ_1 and σ_2. $\mathcal{I}(\sigma) \equiv \Pi_{y:\mathcal{I}(\sigma_1)}.\mathcal{I}(\sigma_2)$ where y is fresh and $\mathcal{I}(\sigma) \twoheadrightarrow_\beta \sigma'$.*

  Now this can only be possible if $\sigma' \equiv \Pi_{y:\sigma_1'}.\sigma_2'$ where $\mathcal{I}(\sigma_1) \twoheadrightarrow_\beta \sigma_1', \mathcal{I}(\sigma_2) \twoheadrightarrow_\beta \sigma_2'$ and y is still fresh.

  By IH, $\sigma_1' \equiv \mathcal{I}(\sigma_1)$ and $\sigma_2' \equiv \mathcal{I}(\sigma_2)$ and so, $\sigma' \equiv \mathcal{I}(\sigma_1 \rightarrow \sigma_2) \equiv \mathcal{I}(\sigma)$.

□

Lemma 7.17 *If $\sigma, \sigma' \in \mathcal{T}_{\rightarrow}$ and $\mathcal{I}(\sigma) \twoheadrightarrow_\beta \mathcal{I}(\sigma')$ then $\sigma' \equiv \sigma$.*
Proof: *By induction on $\sigma \in \mathcal{T}_{\rightarrow}$ using Lemma 7.16.* □

7.3 $\lambda_{\rightarrow\tau}$ generalises $\lambda_{\rightarrow}$

Here we shall prove that t is typable in $\lambda_{\rightarrow}$ iff $\mathcal{I}(t)$ is consistent in $\lambda_{\rightarrow\tau}$. Furthermore, if σ is the type of t in $\lambda_{\rightarrow}$ then the type of $\mathcal{I}(t)$ in $\lambda_{\rightarrow\tau}$ β-reduces to $\mathcal{I}(\sigma)$. Note here that the reason why we get β-reduction instead of equivalence is that we keep the whole structure of our terms and types. I.e. we don't assume the traditional lines of saying that if f has the arrow type $\sigma \rightarrow \sigma'$ and a has the type σ then fa has the type σ'. Rather we say that fa has type $(\sigma \rightarrow \sigma')a$. So we still have to perform a β-reduction. $\lambda_{\rightarrow}$ on the other hand, follows the traditional lines.

Lemma 7.18 *Let $\Gamma \in \mathcal{K}, t \in \mathcal{T}_{\rightarrow}$ and $\sigma \in \mathcal{T}_{\rightarrow}$. If $\Gamma \vdash_{\lambda_{\rightarrow}} t : \sigma$ then $\mathcal{I}(\Gamma) \vdash \mathcal{I}(t)$ and $\tau(\mathcal{I}(\Gamma), \mathcal{I}(t)) \twoheadrightarrow_\beta \mathcal{I}(\sigma)$.*
Proof: *By induction on the derivation $\Gamma \vdash_{\lambda_{\rightarrow}} t : \sigma$.*

- *$\Gamma \vdash_{\lambda_{\rightarrow}} y : \sigma$ comes from $(y : \sigma) \in \Gamma$, then $\mathcal{I}(\Gamma) \equiv \mathcal{I}(\Gamma')(y : \mathcal{I}(\sigma))\mathcal{I}(\Gamma'')$ for some Γ' and Γ'' with $\Gamma' \cup \Gamma'' \cup \{(y : \sigma)\} \equiv \Gamma$. Now, $WB(\mathcal{I}(\Gamma'))$ by Corollary 7.12, hence $\mathcal{I}(\Gamma') \vdash \mathcal{I}(\sigma)$ be Lemma 7.14. It follows that $\mathcal{I}(\Gamma')(y : \mathcal{I}(\sigma))\mathcal{I}(\Gamma'') \equiv \mathcal{I}(\Gamma) \vdash y$ by Definition 6.3 and $\tau(\mathcal{I}(\Gamma), y) \equiv \mathcal{I}(\sigma)$.*
- *$\Gamma \vdash_{\lambda_{\rightarrow}} tt' : \sigma'$ comes from $\Gamma \vdash_{\lambda_{\rightarrow}} t : \sigma \rightarrow \sigma'$ and $\Gamma \vdash_{\lambda_{\rightarrow}} t' : \sigma$ and where IH holds for t and t' then $\tau(\mathcal{I}(\Gamma), \mathcal{I}(t)) \twoheadrightarrow_\beta \mathcal{I}(\sigma \rightarrow \sigma') \equiv \Pi_{y:\mathcal{I}(\sigma)}.\mathcal{I}(\sigma'))$, where $y \notin \mathcal{I}(\sigma')$, and $\tau(\mathcal{I}(\Gamma), \mathcal{I}(t')) \twoheadrightarrow_\beta \mathcal{I}(\sigma)$.*

  Hence $\tau(\mathcal{I}(\Gamma), \mathcal{I}(tt')) \equiv \tau(\mathcal{I}(\Gamma), \mathcal{I}(t)\mathcal{I}(t')) \equiv \tau(\mathcal{I}(\Gamma), \mathcal{I}(t))\mathcal{I}(t') \twoheadrightarrow_\beta$

  $\Pi_{y:\mathcal{I}(\sigma)}.\mathcal{I}(\sigma'))\mathcal{I}(\sigma) \twoheadrightarrow_\beta \mathcal{I}(\sigma')[y := \mathcal{I}(\sigma)] \equiv \mathcal{I}(\sigma')$ as $y \notin FV(\mathcal{I}(\sigma'))$. Moreover, by IH we also have $\mathcal{I}(\Gamma) \vdash \mathcal{I}(t)$ and $\mathcal{I}(\Gamma) \vdash \mathcal{I}(t')$. Now use clause 5 of Definition 6.3 to get that $\mathcal{I}(\Gamma) \vdash \mathcal{I}(tt')$.
- *$\Gamma \vdash_{\lambda_{\rightarrow}} \lambda_{y:\sigma}.t : \sigma \rightarrow \sigma'$ comes from $\Gamma(y : \sigma) \vdash_{\lambda_{\rightarrow}} t : \sigma'$ and IH holds for t then $\tau(\mathcal{I}(\Gamma)(y : \mathcal{I}(\sigma)), \mathcal{I}(t)) \twoheadrightarrow_\beta \mathcal{I}(\sigma')$ and*

  $\tau(\mathcal{I}(\Gamma), \mathcal{I}(\lambda_{y:\sigma}.t)) \equiv \tau(\mathcal{I}(\Gamma), \lambda_{y:\mathcal{I}(\sigma)}.\mathcal{I}(t)) \equiv \Pi_{y:\mathcal{I}(\sigma)}.\tau(\mathcal{I}(\Gamma)(y : \mathcal{I}(\sigma)), \mathcal{I}(t)) \twoheadrightarrow_\beta$

$\Pi_{y:\mathcal{I}(\sigma)}.\mathcal{I}(\sigma') \equiv \mathcal{I}(\sigma \to \sigma')$.
Moreover, by IH, $\mathcal{I}(\Gamma)(y : \mathcal{I}(\sigma)) \vdash \mathcal{I}(t)$. *Moreover, from Lemma 7.14, we get that* $\mathcal{I}(\Gamma) \vdash \mathcal{I}(\sigma)$. *So now, apply clause 3 of Definition 6.3 to obtain that* $\mathcal{I}(\Gamma) \vdash \mathcal{I}(\lambda_{y:\sigma}.t)$.

□

Theorem 7.19 *Let* $\Gamma \in \mathcal{K}, t \in \mathcal{T}_{\to}$ *and* $\sigma \in \mathcal{T}_{\to}$. *If* $\Gamma \vdash_{\lambda_{\to}} t : \sigma$ *then*

1. $\mathcal{I}(\Gamma) \vdash \mathcal{I}(\sigma)$
2. $\mathcal{I}(\Gamma) \vdash \mathcal{I}(t)$
3. $\tau(\mathcal{I}(\Gamma), \mathcal{I}(t)) \twoheadrightarrow_\beta \mathcal{I}(\sigma)$.

Proof: *Use Lemmas 7.14 and 7.18.*

□

Theorem 7.20 *If* $t \in \mathcal{T}_{\to}$ *and* Γ *is a set of statements of* $\lambda_{\to}$ *then* $\mathcal{I}(\Gamma) \vdash \mathcal{I}(t)$ *implies* $(\exists\sigma \in \mathcal{T}_{\to})[\Gamma \vdash_{\lambda_{\to}} t : \sigma \wedge \tau(\mathcal{I}(\Gamma), \mathcal{I}(t)) \twoheadrightarrow_\beta \mathcal{I}(\sigma)]$.

Proof: *Note that* $\mathcal{I}(\Gamma) \vdash \mathcal{I}(t)$ *implies that* $WB(\mathcal{I}(\Gamma))$ *and hence by Corollary 7.12,* $\Gamma \in \mathcal{K}$. *Now the proof is by induction on* $t \in \mathcal{T}_{\to}$.

- *Case* $t \equiv y$ *then from Lemma 6.9,* $(\exists\Gamma'\Gamma'')[\mathcal{I}(\Gamma) \equiv \mathcal{I}(\Gamma')(y : \tau(\mathcal{I}(\Gamma), y))\mathcal{I}(\Gamma'')]$. *Hence,* $\exists\sigma \in \mathcal{T}_{\to}$ *such that* $\mathcal{I}(\sigma) \equiv \tau(\mathcal{I}(\Gamma), y)$ *and so* $\tau(\mathcal{I}(\Gamma), \mathcal{I}(t)) \equiv \tau(\mathcal{I}(\Gamma), y) \equiv \mathcal{I}(\sigma)$. *Moreover,* $\Gamma'(y : \sigma)\Gamma'' \vdash_{\lambda_{\to}} t : \sigma$ *as* $WB_{\to}(\Gamma'(y : \sigma)\Gamma'')$ *and* $y \notin dom(\Gamma'\Gamma'')$.

- *Case* $t \equiv \lambda_{y:\sigma}.t'$ *then from Lemma 6.9,* $\mathcal{I}(\Gamma) \vdash \mathcal{I}(\sigma)$ *and* $\mathcal{I}(\Gamma)(y : \mathcal{I}(\sigma)) \vdash \mathcal{I}(t')$. *Now by IH,* $(\exists\sigma' \in \mathcal{T}_{\to})[\tau(\mathcal{I}(\Gamma)(y : \mathcal{I}(\sigma)), \mathcal{I}(t')) \twoheadrightarrow_\beta \mathcal{I}(\sigma') \wedge \Gamma(y : \sigma) \vdash_{\lambda_{\to}} t' : \sigma'$. *Hence* $\tau(\mathcal{I}(\Gamma), \mathcal{I}(\lambda_{y:\sigma}.t')) \equiv \Pi_{y:\mathcal{I}(\sigma)}.\tau(\mathcal{I}(\Gamma)(y : \mathcal{I}(\sigma)), \mathcal{I}(t')) \to \Pi_{y:\mathcal{I}(\sigma)}.\mathcal{I}(\sigma') \equiv \mathcal{I}(\sigma \to \sigma')$ *and* $\Gamma \vdash_{\lambda_{\to}} \lambda_{y:\sigma}.t' : \sigma \to \sigma'$.

- *Case* $t \equiv t_1 t_2$. *If* $\mathcal{I}(\Gamma) \vdash \mathcal{I}(t_1 t_2)$ *then by lemma 6.9,* $\mathcal{I}(\Gamma) \vdash \mathcal{I}(t_1)$, $\mathcal{I}(\Gamma) \vdash \mathcal{I}(t_2)$, $\tau(\mathcal{I}(\Gamma), \mathcal{I}(t_1)) \twoheadrightarrow_\beta \Pi_{y:\sigma_1}.\sigma_2$ *and* $\tau(\mathcal{I}(\Gamma), \mathcal{I}(t_2)) \twoheadrightarrow_\beta \sigma_1$.
  But by IH, $\exists\sigma_1', \sigma_1'' \in \mathcal{T}_{\to}$ *such that:*
  $\tau(\mathcal{I}(\Gamma), \mathcal{I}(t_1)) \twoheadrightarrow_\beta \mathcal{I}(\sigma_1') \wedge \Gamma \vdash_{\lambda_{\to}} t_1 : \sigma_1'$
  $\tau(\mathcal{I}(\Gamma), \mathcal{I}(t_2)) \twoheadrightarrow_\beta \mathcal{I}(\sigma_1'') \wedge \Gamma \vdash_{\lambda_{\to}} t_2 : \sigma_1''$
  But by Church Rosser, $(\exists\sigma_2'')[\sigma_1 \twoheadrightarrow_\beta \sigma_2'' \wedge \mathcal{I}(\sigma_1'') \twoheadrightarrow_\beta \sigma_2'']$.
  As $\sigma_1'' \in \mathcal{T}_{\to}$ *then* $\sigma_2'' \equiv \mathcal{I}(\sigma_1'')$, *from Lemma 7.16, so* $\sigma_1 \twoheadrightarrow_\beta \mathcal{I}(\sigma_1'')$.
  Now, $\Pi_{y:\sigma_1}.\sigma_2 \twoheadrightarrow_\beta \Pi_{y:\sigma_1''}.\sigma_2 = \mathcal{I}(\sigma_1')$,
  hence again by Church Rosser, $(\exists\sigma_3)[\Pi_{y:\sigma_1''}.\sigma_2 \twoheadrightarrow_\beta \sigma_3 \wedge \mathcal{I}(\sigma_1') \twoheadrightarrow_\beta \sigma_3$,
  As $\sigma_1' \in \mathcal{T}_{\to}$ *then* $\sigma_3 \equiv \mathcal{I}(\sigma_1'')$ *from Lemma 7.16.*
  Hence, $\Pi_{y:\mathcal{I}(\sigma_1'')}.\sigma_2 \twoheadrightarrow_\beta \mathcal{I}(\sigma_1')$. *It follows that* $\mathcal{I}(\sigma_1')$ *must start with a* Π, *so* $\sigma_1' \equiv \sigma_3 \to \sigma_4$ *for some* σ_3 *and* $\sigma_4 \in \mathcal{T}_{\to}$.
  Then $\mathcal{I}(\sigma_1') \equiv \Pi_{z:\mathcal{I}(\sigma_3)}.\mathcal{I}(\sigma_4)$, *hence*
  $\Pi_{y:\mathcal{I}(\sigma_1'')}.\mathcal{I}(\sigma_2) \twoheadrightarrow_\beta \Pi_{z:\mathcal{I}(\sigma_3)}.\mathcal{I}(\sigma_4)$. *It follows that*
  $y \equiv z$, $\mathcal{I}(\sigma_1'') \twoheadrightarrow_\beta \mathcal{I}(\sigma_3)$ *(hence* $\sigma_1'' \equiv \sigma_3$ *by Lemma 7.17) and* $\sigma_2 \twoheadrightarrow_\beta \mathcal{I}(\sigma_4)$.
  Concluding:

1. *From* $\Gamma \vdash_{\lambda_\to} t_1 : \sigma_1'$ *(i.e.* $\Gamma \vdash_{\lambda_\to} t_1 : (\sigma_3 \to \sigma_4)$*, or* $\Gamma \vdash_{\lambda_\to} t_1 : (\sigma_1'' \to \sigma_4)$*) and* $\Gamma \vdash_{\lambda_\to} t_2 : \sigma_1''$*, we obtain* $\Gamma \vdash_{\lambda_\to} t_1 t_2 : \sigma_4$.
2. $\tau(\mathcal{I}(\Gamma), \mathcal{I}(t_1)) \twoheadrightarrow_\beta \Pi_{y:\mathcal{I}(\sigma_1'')}.\sigma_2 \twoheadrightarrow_\beta \Pi_{y:\mathcal{I}(\sigma_3)}.\mathcal{I}(\sigma_4)$*, so* $\tau(\mathcal{I}(\Gamma), \mathcal{I}(t_1 t_2)) \equiv (\tau(\mathcal{I}(\Gamma), \mathcal{I}(t_1))t_2 \twoheadrightarrow_\beta \mathcal{I}(\sigma_4)$.

□

Corollary 7.21 *If* $t \in \mathcal{T}_\to$ *and* $\Gamma \in \mathcal{K}$,
then $\mathcal{I}(\Gamma) \vdash \mathcal{I}(t)$ *iff* $(\exists \sigma \in \mathcal{T}_\to)[\Gamma \vdash_{\lambda_\to} t : \sigma \wedge \tau(\mathcal{I}(\Gamma), \mathcal{I}(t)) \twoheadrightarrow_\beta \mathcal{I}(\sigma) \wedge \mathcal{I}(\Gamma) \vdash \mathcal{I}(\sigma)]$
Proof: *Use Theorems 7.19 and 7.20.* □

Hence, if Γ' and t', belonging to $\lambda_{\to\tau}$, are images of Γ and t in $\lambda_\to$, i.e. $\Gamma' \equiv \mathcal{I}(\Gamma)$ and $t' \equiv \mathcal{I}(t)$, then $\Gamma' \vdash t'$ implies $\Gamma \vdash_{\lambda_\to} t : \sigma$ for some σ. There is a comparable theorem for *general* Γ' and t' in $\lambda_{\to\tau}$:
If $\Gamma' \vdash t'$, then there are Γ, t and σ in $\lambda_\to$ such that $\Gamma' \twoheadrightarrow_\beta \mathcal{I}(\Gamma), t' \twoheadrightarrow_\beta \mathcal{I}(t)$, $\tau(\Gamma', t') =_\beta \mathcal{I}(\sigma)$, and $\Gamma \vdash_{\lambda_\to} t : \sigma$.
In order to prove this, we first give a number of definitions and lemmas.

Definition 7.22

- *Let* $\sigma \in \mathcal{T}$. *We call* σ simple *(or a* simple type*) if there are no applications in* σ. *The set of all simple types in* $\mathcal{T}$ *is denoted by* $\mathcal{T}^s$.
- *A context* Γ *in* $\lambda_{\to\tau}$ *is called simple if all* $\sigma \in ran(\Gamma)$ *are simple.*

It follows that simple types in $\lambda_{\to\tau}$ can be constructed using the following abstract syntactic rule:

$$\mathcal{T}' = \mathcal{V} \mid (\Pi_{\mathcal{V}:\mathcal{T}'}.\mathcal{T}')$$

which is a restricted version of the syntactic rule:

$$\mathcal{T} = \mathcal{V} \mid (\Pi_{\mathcal{V}:\mathcal{T}}.\mathcal{T}) \mid (\mathcal{T}\mathcal{T})$$

of Section 2. Now we have the following:

Lemma 7.23 *All simple types are in normal form.*
Proof: *A simple type contains no application, hence no redexes.* □

Lemma 7.24 *If* $\sigma \in \mathcal{T}^s$ *and* $\Gamma \in CONS$ *then* $\Gamma \vdash \sigma$.
Proof: *By induction on* $\sigma \in \mathcal{T}^s$. □

It is also clear that simple types do not contain occurrences of terms $t \in \mathcal{T}$ (except for the binding variables y being a subscript of the Π). In particular, there are no occurrences of variables in a simple type, but for the mentioned binding variables. This means that all binding variables y (subscripts of Π's) actually bind nothing at all.
Hence, there is a well-defined backwards translation from $\lambda_{\to\tau}$ to $\lambda_\to$ for simple types:

Definition 7.25 *The mapping $\mathcal{J} : \mathcal{T}^s \longrightarrow \mathcal{T}_{\rightarrow}$ is defined as follows:*

$$\begin{array}{lll} \mathcal{J}(\gamma) & \equiv & \gamma \\ \mathcal{J}(\Pi_{y:\sigma}.\sigma') & \equiv & \sigma \rightarrow \sigma' \end{array}$$

Note that the mapping $\mathcal{I}$, being injective, defines an *embedding* of the types of $\lambda_{\rightarrow}$ in those of $\lambda_{\rightarrow\tau}$ (i.e. $\mathcal{T}$). The mapping $\mathcal{J}$ is the inverse of $\mathcal{I}$ on $\mathcal{T}^s$.

We will now show that every consistent type in $\lambda_{\rightarrow\tau}$ is β-equal to a simple type.

Lemma 7.26 *Let $\sigma \in \mathcal{T}$ be such that $\Gamma \vdash \sigma$ for some Γ. Then there is a $\sigma' \in \mathcal{T}^s$ such that $\sigma \twoheadrightarrow_\beta \sigma'$.*

Proof: *By induction on $\Gamma \vdash \sigma$.*

1. *Case $\Gamma \vdash \gamma$ is trivial.*
2. *Case $\Gamma \vdash \Pi_{y:\sigma}.\sigma_1$ comes from $\Gamma \vdash \sigma$ and $\Gamma(y : \sigma) \vdash \sigma_1$. By IH: $\sigma \twoheadrightarrow \sigma' \in \mathcal{T}^s$ and $\sigma_1 \twoheadrightarrow_\beta \sigma_1' \in \mathcal{T}^s$. Hence $\Pi_{y:\sigma}.\sigma_1 \twoheadrightarrow_\beta \Pi_{y:\sigma'}.\sigma_1' \in \mathcal{T}^s$.*
3. *Case $\Gamma \vdash \sigma t$ comes from $\Gamma \vdash \sigma$, $\Gamma \vdash t$, $\sigma \twoheadrightarrow_\beta \Pi_{y:\sigma_1}.\sigma_2$ and $\tau(\Gamma, t) \twoheadrightarrow_\beta \sigma_1$. Then by IH: $\sigma \twoheadrightarrow \sigma' \in \mathcal{T}^s$. By Theorem 3.16 and Lemma 7.23 it follows that $\Pi_{y:\sigma_1}.\sigma_2 \twoheadrightarrow_\beta \sigma'$. Hence, $\sigma' \equiv \Pi_{y:\sigma_1'}.\sigma_2' \in \mathcal{T}^s$. Hence $\sigma t \twoheadrightarrow_\beta (\Pi_{y:\sigma_1}.\sigma_2)t \twoheadrightarrow_\beta (\Pi_{y:\sigma_1'}.\sigma_2')t \twoheadrightarrow_\beta \sigma_2'[y := t] = \sigma_2'$ since σ_2' is also a simple type, containing no (free) variables.*

□

Corollary 7.27 *If $\Gamma \vdash \sigma$ then there is a $\sigma_1 \in \mathcal{T}_{\rightarrow}$ such that $\sigma \twoheadrightarrow_\beta \mathcal{I}(\sigma_1)$.* □

Next we concentrate on the terms of $\lambda_{\rightarrow\tau}$.

Definition 7.28 *We call a term $t \in \mathcal{T}$ a* Church-term *if all types occurring in t are simple. We denote the set of all Church-terms by $\mathcal{T}^{ch}$.*

There is a nice relation between terms in $\lambda_{\rightarrow\tau}$ and Church-terms:

Lemma 7.29 *Let $t \in \mathcal{T}$ such that $\Gamma \vdash t$ for some Γ. Then there is a $t' \in \mathcal{T}^{ch}$ such that $t \twoheadrightarrow_\beta t'$.*

Proof: *By induction on $\Gamma \vdash t$.*

1. *Case $\Gamma(y : \sigma)\Gamma' \vdash y$ is trivial.*
2. *Case $\Gamma \vdash \lambda_{y:\sigma}.t$ comes from $\Gamma \vdash \sigma$ and $\Gamma(y : \sigma) \vdash t$. By Lemma 7.26: there is $\sigma' \in \mathcal{T}^s$ such that $\sigma \twoheadrightarrow \sigma'$. Moreover, by IH: there is $t' \in \mathcal{T}^{ch}$ such that $t \twoheadrightarrow_\beta t'$. Hence, $\lambda_{y:\sigma}.t \twoheadrightarrow_\beta \lambda_{y:\sigma'}.t' \in \mathcal{T}^{ch}$.*
3. *Case $\Gamma \vdash tt_1$ comes from $\Gamma \vdash t, \Gamma \vdash t_1$, $\tau(\Gamma, t) \twoheadrightarrow_\beta \Pi_{y:\sigma_1}.\sigma_2$ and $\tau(\Gamma, t_1) \twoheadrightarrow_\beta \sigma_1$.*
   By IH: there is $t' \in \mathcal{T}^{ch}$ such that $t \twoheadrightarrow_\beta t'$ and $t_1' \in \mathcal{T}^{ch}$ such that $t' \twoheadrightarrow_\beta t_1'$. Hence $tt_1 \twoheadrightarrow_\beta t't_1' \in \mathcal{T}^{ch}$ since the application of t' to t_1' cannot introduce an application in the types occurring in $t't_1'$.

□

We can extend the mapping $\mathcal{J}$ of Definition 7.25 to Church-terms:

Definition 7.30 $\mathcal{J}$: $(\mathcal{T}^s \cup \mathcal{T}^{ch}) \longrightarrow \mathcal{T}_{\rightarrow}$ *is defined as follows:*

$$\begin{array}{lll} \mathcal{J}(\gamma) & \equiv & \gamma \\ \mathcal{J}(\Pi_{y:\sigma}.\sigma') & \equiv & \sigma \rightarrow \sigma' \\ \mathcal{J}(y) & \equiv & y \\ \mathcal{J}(\lambda_{y:\sigma}.t) & \equiv & \lambda_{y:\mathcal{J}(\sigma)}.\mathcal{J}(t) \\ \mathcal{J}(t_1 t_2) & \equiv & \mathcal{J}(t_1)\mathcal{J}(t_2) \end{array}$$

Note that $\mathcal{J}$ is well-defined. Note also that a Church-term t cannot contain a Π-redex of the form $(\Pi_{y:\sigma}.t_1)t_2$, since simple types contain no applications.

As the main result of this section, we prove the following theorem. Recall the definition of Γ_M, being the context relevant to $\Gamma \vdash M$ (see Definition 6.18).

Theorem 7.31 *Let* $\Gamma' \vdash t'$. *Then there* Γ, t *and* σ *in* $\lambda_{\rightarrow}$ *such that* $\Gamma \vdash_{\lambda_{\rightarrow}} t : \sigma$, $t' \twoheadrightarrow_{\beta} \mathcal{I}(t)$ *and* $\tau(\Gamma', t') =_{\beta} \mathcal{I}(\sigma)$. *Moreover,* $\Gamma'_{t'} \twoheadrightarrow_{\beta} \mathcal{I}(\Gamma)$.

Proof: *¿From* $\Gamma' \vdash t'$ *follows* $\Gamma'_{t'} \vdash t'$ *(Corollary 6.17).*

Let $\Gamma'_{t'}$ *be* $(y_1 : \sigma_1) \dots (y_n : \sigma_n)$. *Then* $(y_1 : \sigma_1) \dots (y_{i-1} : \sigma_{i-1}) \vdash \sigma_i$ *by Lemma 6.21. Hence there are* $\sigma'_i \in \mathcal{T}^s$ $(i = 1, \dots, n)$ *such that* $\sigma_i \twoheadrightarrow_{\beta} \sigma'_i$ *by Lemma 7.26. It follows that* $\Gamma'' \equiv (y_1 : \sigma'_1) \dots (y_n : \sigma'_n) \vdash t'$ *by Corollary 6.27. Take* $\Gamma \equiv (y_1 : \mathcal{J}(\sigma'_1)) \dots (y_n : \mathcal{J}(\sigma'_n))$. *Then clearly* $\mathcal{I}(\Gamma) \equiv \Gamma''$, *so* $\Gamma'_{t'} \twoheadrightarrow_{\beta} \mathcal{I}(\Gamma)$.
Moreover, by Lemma 7.29, there is $t'' \in \mathcal{T}^{ch}$ *such that* $t' \twoheadrightarrow_{\beta} t''$ *and* $\mathcal{I}(\Gamma) \vdash t''$ *by Corolaary 6.25. Take* $t \equiv \mathcal{J}(t'')$, *then* $t'' \equiv \mathcal{I}(t)$ *and* $\mathcal{I}(\Gamma) \vdash \mathcal{I}(t)$. *Then by Corollary 7.21, there is* $\sigma \in \mathcal{T}_{\rightarrow}$ *such that* $\Gamma \vdash_{\lambda_{\rightarrow}} t : \sigma$. *From Theorem 7.20:* $\tau(\mathcal{I}(\Gamma), \mathcal{I}(t)) \twoheadrightarrow_{\beta} \mathcal{I}(\sigma)$, *i.e.* $\tau(\Gamma'', t'') \twoheadrightarrow_{\beta} \mathcal{I}(\sigma)$. *By Corollary 6.27:* $\tau(\Gamma'', t') =_{\beta} \mathcal{I}(\sigma)$ *and it follows from Lemma 5.7 that* $\tau(\Gamma', t') =_{\beta} \mathcal{I}(\sigma)$.
The various entries in this proof can be pictured in Figure 1. □

Finally, note that even for Church-terms, τ and Γ do not coincide. That is, we know that if $\Gamma \vdash t$ then $\uparrow \tau(\Gamma, t)$. However, even if $t \in \mathcal{T}^{ch}$ and if $\uparrow \tau(\Gamma, t)$, we still don't get $\Gamma \vdash t$. In fact, look at Example 6.26 where $\uparrow \tau(\emptyset, t)$ yet $\emptyset \not\vdash t$, for $t \equiv (\lambda_{y:\sigma}.y)z$.

8 Conclusion

In this paper, we presented a calculus $\lambda_{\rightarrow\tau}$ where types and terms are treated alike and where β-conversion is used for both forms of expressions. We showed that the Church Rosser theorem holds for the calculus and extended it with typing and consistency operators. These operators satisfy most of the important results concerning typing, such as weakening and substitution. The aim of our calculus is to provide a general calculus which accommodates types and terms in a unified way and which preserves the characteristics of the typing systems. More importantly, compatibility of typing in our calculus holds for both abstraction and application whereas in most of the known type theories (especially in PTSs), compatibility holds only for abstraction. Furthermore, $\Gamma \vdash t : \sigma$ is split in two judgments: $\Gamma \vdash t$ and $\tau(\Gamma, t) =_{\beta} \sigma$ which is another step towards getting a fine structure for the typing relation and for separating the two distinct notions of whether a term has a type and of what is its type.

As an example, we interpreted Church's $\lambda_{\rightarrow}$ in our calculus and showed that in order to type a term in $\lambda_{\rightarrow}$, it is enough to show it consistent in $\lambda_{\rightarrow\tau}$. Similarly, one can interpret other

$$\begin{tikzcd}
 & \Gamma' \arrow[r, phantom, "\vdash"] \arrow[d, phantom, "\sqsupseteq"] & t' \arrow[d, phantom, "\equiv"] \\
 & \Gamma'_{t'} \arrow[r, phantom, "\vdash"] \arrow[d, "\beta"] & t' \arrow[d, "\beta"] \\
\mathcal{T}^s\text{-context:} & \Gamma'' \arrow[r, phantom, "\vdash"] \arrow[d, leftrightarrow, "\mathcal{I}"', "\mathcal{J}"] & t'' \in \mathcal{T}^s \arrow[d, leftrightarrow, "\mathcal{I}"', "\mathcal{J}"] \\
 & \Gamma \arrow[r, phantom, "\vdash"] & t : \sigma
\end{tikzcd}$$

Figure 1: Dependencies between $\lambda_{\rightarrow}$ and $\lambda_{\rightarrow\tau}$

typing systems in $\lambda_{\rightarrow\tau}$ and show similar results. The system $\lambda_{\rightarrow\tau}$ is the first which provides an extended treatment of unifying types and terms while preserving most of the desirable properties of typing systems. Furthermore, the line of this paper should be followed in the future to deal with other systems than $\lambda_{\rightarrow}$. We believe that the same startegy can be used for those systems of the Barendregt cube giving yet a more elegant structure of generalised type systems.

References

[Barendregt 84] Barendregt, H., *Lambda calculus: its syntax and semantics,* North-Holland, 1984.

[BH 90] Barendregt, H. and Hemerik, K., Types in lambda calculi and programming languages, in *European Symposium on Programming, Copenhagen*, Ed. N. Jones, LNCS, 432, 1-36, Springer, 1990.

[Barendregt 91] Barendregt, H. Introduction to generalised type systems, *Functional programming 1(2)*, 125-154, 1991.

[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.

[van Benthem Jutting 92] van Benthem Jutting, B., Postponement of Expansion in Pure Type Systems, University of Eindhoven, 1992.

[Church 1932] A set of postulates for the foundation of logic, *Annals of Math.* 33 (1932), 346–366 and 34 (1933), 839–864.

[Church 1940] A. Church, A formulation of the simple theory of types, *Journal of Symbolic Logic* 5 (1940), 56–68.

[Frege 1879] Frege, G., *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens* (Halle, Verlag von Louis Nebert, 1879). Reprint 1964 (Hildesheim, Georg Olms Verlagsbuchhaltung).

[GN 94] Geuvers, J.H. and Nederpelt, R.P., Typed and untyped λ-calculus, two chapters in *Logic, Language and Computer Science*, part II, Eds. de Swart, H.C.M. et al.

[Hilbert and Ackermann 1928] Hilbert, D. and Ackermann, W., *Grundzüge der theoretischen Logik* (Berlin, Springer Verlag, 1928).

[Kamareddine 89] Kamareddine, F., *Semantics in a Frege structure*, Ph.D. thesis, University of Edinburgh, 1989.

[Kamareddine 92a] Kamareddine, F., A system at the cross roads of logic and functional programming, *Science of Computer Programming 19*, 239-279, 1992.

[Kamareddine 92b] Kamareddine, F., λ-terms, logic, determiners and quantifiers, *Logic, Language and Information 1 (1)*, 79-103, 1992.

[Kamareddine 92c] Kamareddine, F., Set Theory and Nominalisation, Part I, *Logic and Computation 2 (5)*, 579-604, 1992.

[Kamareddine 92d] Kamareddine, F., Set Theory and Nominalisation, Part II, *Logic and Computation 2 (6)*, 687-707, 1992.

[Kamareddine 92e] Kamareddine, F., Are types needed for Natural Language?, Proceedings for the applied Logic conference, Amsterdam, December 1992. The proceedings will appear (in revised form) as a book published by Kluwer.

[KK 93] Kamareddine, F., and Klein, E., Polymorphism, Type containment and Nominalisation, *Logic, Language and Information 2*, 171-215, 1993.

[Kamareddine 93] Kamareddine, F., Non well foundedness and type freeness can unify the interpretation of functional application, to appear in *Logic, Language and Information*, 1993.

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

[KK 9x] Kamareddine, F., and Klein, E., Polymorphism and Logic in Programming and Natural languages, submitted for publication.

[KN 9x] Kamareddine, F., and Nederpelt, R.P., A useful lambda notation, submitted for publication.

[KN 9y] Kamareddine, F., and Nederpelt, R.P., A semantics for a fine λ-calculus with De Bruijn indices, submitted for publication.

[KN 9y] Kamareddine, F., and Nederpelt, R.P., *The Beauty of the λ-Calculus*, in preparation.

[Nederpelt 73] Nederpelt, R.P., *Strong normalisation in a typed lambda calculus with lambda structured types*, Ph.D. thesis, Eindhoven University of Technology, Department of Mathematics and Computer Science, 1973.

[Nederpelt 80] Nederpelt, R.P., An approach to theorem proving on the basis of a typed lambda-calculus, in *5th Conference on Automated Deduction*, Les Arcs, France, 1980, Eds. W. Bibel and R. Kowalski, LCNS, 87, 182-194, Springer, 1980.

[Nederpelt 87] Nederpelt, R.P., *De Taal van de Wiskunde*, Versluys, Almere, 1987.

[Nederpelt 90] Nederpelt, R.P., Type systems — basic ideas and applications, in: *CSN '90, Computing Science in the Netherlands 1990*, Stichting Mathematisch Centrum, Amsterdam, 1990.

[Nederpelt 92] Nederpelt, R.P., The fine structure of lambda calculus, Computing Science Note 92/07, Eindhoven University of Technology, 1992.

[NK 94] Nederpelt, R.P., and Kamareddine, F., A unified approach to type theory through a refined λ-calculus, paper presented at the 1992 conference on *Mathematical Foundations of Programming Semantics*, to appear in the proceedings.

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

[Whitehead and Russell 1910] Whitehead, A.N. and Russell, B., *Principia Mathematica* (Cambridge, Cambridge University Press, 1910/1913). Reprint 1960, same editor.

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.

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. Komatzky	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	The object-oriented paradigm, p. 28.