Comparing Cubes

Steffen van Bakel[1*] Luigi Liquori[2†] Simona Ronchi della Rocca[2]

Paweł Urzyczyn[3‡]

[1] Afdeling Informatica,
Universiteit Nijmegen,
Toernooiveld 1,
6525 ED Nijmegen, Nederland.
E-mail: steffen@cs.kun.nl.

[2] Dipartimento di Informatica,
Università degli Studi di Torino,
Corso Svizzera 185,
10145 Torino, Italia.
E-mail: {liquori, ronchi}@di.unito.it.

[3] Instytut Informatyki
Uniwersytetu Warszawskiego,
ul. Banacha 2,
02-097 Warszawa, Polska.
E-mail: urzy@mimuw.edu.pl.

Abstract

We study the *cube of type assignment systems*, as introduced in [10]. This cube is obtained from Barendregt's typed λ-cube [1] via a natural type erasing function E, that erases type information from terms. We prove that the systems in the former cube enjoy good computational properties, like subject reduction and strong normalization. We study the relationship between the two cubes, which leads to some unexpected results in the field of systems with dependent types.

Introduction

Types can be used as predicates for terms of λ-calculus in two different ways. Terms can be directly *decorated* with types, and then every term comes directly with a unique, intrinsic type. In this *fully typed* approach, a *typed system* is a set of rules for proving judgements of the shape $\Gamma_t \vdash_t M_t : \phi_t$, where M_t is a typed term, ϕ_t is a type, and Γ_t is a context. The meaning of such a judgement is: the term M_t has type ϕ_t under the context Γ_t, that contains the types of the free variables of M_t and ϕ_t. Alternatively, in the *type assignment* approach, types can be assigned to terms of the untyped λ-calculus by applying type assignment rules. A *type assignment system* is a set of rules for proving judgements of the shape $\Gamma \vdash M : \phi$, where M is a term of the untyped λ-calculus, and Γ assigns types to the free variables of M and ϕ. The meaning of such a judgement is: the term M has type ϕ under the context Γ, containing the types of free variable of M and ϕ. In this approach, each term has infinitely many typings.

The typed approach, called *à la Church* by Barendregt, gives rise to different typed languages. In these languages terms are decorated with types in different ways. Examples of typed λ-calculi are the simply typed one, the second order λ-calculus of Girard and Reynolds

*Supported by the Netherlands Organisation for the Advancement of Pure Research (N.W.O.).

†Partly supported by HCM project No. ERBCHRXCT920046 "Typed Lambda Calculus"

‡Partly supported by grants NSF CCR–9113196, KBN 2 1192 91 01 and by a grant from the Commission of The European Communities ERB–CIPA–CT92–2266(294).

[11, 15], and the calculus of constructions [5, 6]. Barendregt [1] gave a compact and appealing presentation of a class of typed systems, arranging them in a *cube*. In this cube, every vertex represents a different typed system. One vertex is the *origin* and represents the simply typed λ-calculus of Church; the edges represent the introduction of some new rules of type formation, namely *Polymorphism*, *Higher Order* and *Dependencies*. This three-dimensional structure allows for a deep comparative analysis of different typed λ-calculi.

It is well known (see [10, 12]) that some of the type assignment systems already known in the literature can be also defined through an *erasing function* that erases type information from terms in a typed system. For those systems, if $\mathcal{D}_t$ is a typed derivation of $\Gamma \vdash_t M_t : \phi$, and E is the erasing function, then by applying E to every judgement in $\mathcal{D}_t$, a valid type assignment derivation proving the judgement $\Gamma \vdash E(M_t) : \phi$ is obtained, where $E(M_t)$ is a term of the untyped λ-calculus. Vice versa, every type assignment derivation can be viewed as the result of the application of E to a typed one. In particular, the erasing function E induces an *isomorphism* between every typed system on the dependency-free side of Barendregt's cube and a corresponding type assignment system. For instance, the simply typed λ-calculus is isomorphic to the Curry type assignment system, the second order lambda calculus to the polymorphic type assignment system, and the higher order λ-calculus to the higher order type assignment system. These correspondences were independently defined by Curry [4], Leivant [14], and Giannini and Ronchi [9], but the induced erasing function is the same in all cases. In [10] the erasing function was extended in a natural way to all typed systems in Barendregt's cube, including the systems with dependent types, as studied in [3, 12]. The essential difference is that the domain of E was extended to include types too, since terms can occur in types.

This erasing function induces a *cube of type assignment systems*. Namely, for every typed system S_t in Barendregt's cube, there is a corresponding type assignment system S, whose rules are obtained from the ones of S_t via the extended erasing function E. Note that, in this setting, if $\Gamma_t \vdash_t M_t : \phi_t$ is a typed judgement, the corresponding type assignment judgement is $E(\Gamma_t) \vdash_t E(M_t) : E(\phi_t)$, where now $E(\phi_t)$ can be different from ϕ_t ($E(\Gamma_t)$ from Γ_t), in case ϕ_t is a dependent type (Γ_t contains dependent types). This cube is a compact presentation of a class of type assignment systems, which partially coincide with known ones (in the side of the cube without dependencies) and partially represents the first attempt of defining type assignment systems with term-dependencies. It was also observed in [10] that, surprisingly, the isomorphism between derivations in the corresponding vertices of typed and type assignment cubes is no longer true in presence of dependencies. Then the natural question arises: what is the relation between the two cubes? The authors of [10] conjectured that the relation is an isomorphism between judgements rather than derivations, that is, a judgement $\Gamma \vdash M : \phi$ is true in one of the type assignment systems if and only if, in the corresponding typed system, a judgement $\Gamma_t \vdash_t M_t : \phi_t$ can be proved such that $E(\Gamma_t) = \Gamma$, $E(M_t) \equiv M$ and $E(\phi_t) \equiv \phi$.

In this paper we disprove this conjecture, showing that it is true only for the systems without polymorphism. We also give a deep analysis of the type assignment cube, showing that the systems represented in it enjoy all the good properties we expect, like subject reduction and strong normalization of typable terms. Moreover, we define a new erasing

function E', that coincides with E when dependencies are not present. The main difference between E and E' is that, while E always erases type information in terms, E' is context dependent and erases type information from a term only if that term does not occur in a type; otherwise it leaves the term unchanged. Clearly a new type assignment cube can be defined starting from E'. This cube is isomorphic to the typed one, in the sense that every type assignment system defined in it is isomorphic to the typed system in the corresponding vertex of Barendregt's cube.

1 Two Cubes

We will present a stratified version of the systems in Barendregt's cube, already presented in [1], which will allow both the definition of the erasing function E and of the related cube of type assignment systems.

1.1 The Cube of Typed Systems

Definition 1.1.1 The sets of *typed λ-terms* (Λ_t), *typed constructors* ($Cons_t$) and *typed kinds* ($Kind_t$) are mutually defined by the following grammar, where M, ϕ and K are metavariables for terms, constructors and kinds respectively:

$$M ::= x \mid \lambda x{:}\phi.M \mid MM \mid \lambda\alpha{:}K.M \mid M\phi$$

$$\phi ::= \alpha \mid \Pi x{:}\phi.\phi \mid \Pi\alpha{:}K.\phi \mid \lambda x{:}\phi.\phi \mid \lambda\alpha{:}K.\phi \mid \phi\phi \mid \phi M$$

$$K ::= * \mid \Pi x{:}\phi.K \mid \Pi\alpha{:}K.K$$

The set T_t of *typed terms* is the union of the sets Λ_t, $Cons_t$ and $Kind_t$.

Notational conventions: In this paper, a *term* will be an *(un)typed λ-term*, a *constructor*, a *kind*, or a *sort*. The symbols M, N, P, Q, ... range over (un)typed λ-terms; ϕ, ψ, ξ, σ, τ, ... range over constructors; K ranges over kinds; s ranges over the set of *sorts*, that is $\{*, \Box\}$; A, B, C, D, ... range over arbitrary terms; x, y, z, ... range over λ-term-variables; α, β, γ, ... range over constructor-variables; a, b, c,... range over λ-term-variables and constructor-variables; and Γ ranges over contexts. All symbols can appear indexed. The symbol $\equiv$ denotes the syntactic identity of terms.

The notions of free and bound variables and of a subterm of a term are defined as usual, i.e. in $\Pi a{:}A.B$ and $\lambda a{:}A.B$ the variable a is considered bound, and the scope of the binding is B. Free variables of A remain free in $\Pi a{:}A.B$ and $\lambda a{:}A.B$, and the subterms of these terms include all subterms of A and B. The set of *subterms* of A is denoted by $\mathrm{ST}(A)$, and the set of free variables of A is denoted by $\mathrm{FV}(A)$. We will consider terms modulo α-conversion, i.e. we identify terms that differ only in the names of bound variables. Let $D[A_1/a_1, \ldots, A_n/a_n]$ denote the result of simultaneously substituting A_i to a_i in D ($1 \leq i \leq n$). We normally assume that no variable bound in D is free in any of the A_i's, and that the set $\{a_1, \ldots, a_n\}$ is disjoint from the set of bound variables of D.

Definition 1.1.2 Beta-reduction (denoted as $\twoheadrightarrow_\beta$) is defined as usual, i.e. as the contextual reflexive and transitive closure of the reduction rule $(\lambda a{:}A.B)C \rightarrow_\beta B[C/a]$. The symbol $=_\beta$ denotes beta-conversion, i.e. the least equivalence relation generated by $\twoheadrightarrow_\beta$.

Definition 1.1.3 i) A *statement* is an expression of the form: $M : \phi$, $\phi : K$, or $K : \Box$, where M is a typed λ-term, ϕ is a constructor, and K is a kind. The left part of the statement is called the *subject*, while the right part is called the *predicate*. A *declaration* is a statement whose subject is a variable.

ii) A *context* is a sequence of declarations, whose subjects are distinct. The empty context is denoted by $<>$. We write $a{:}A \in \Gamma$, if $a{:}A$ occurs in Γ. The *domain* of Γ, denoted by $\mathcal{D}om(\Gamma)$, is the set $\{a \mid \exists A\,[a{:}A \in \Gamma]\}$. If Γ_1 and Γ_2 are contexts such that $\mathcal{D}om(\Gamma_1) \cap \mathcal{D}om(\Gamma_2) = \emptyset$, then Γ_1, Γ_2 is a context obtained by concatenating Γ_1 to Γ_2. The set of free variables in a context is defined by: $\mathrm{FV}(\Gamma) = \bigcup_{a:A\in\Gamma} \mathrm{FV}(A)$.

Definition 1.1.4 *Barendregt's general typed system.* The following rules are used to derive *judgements* of the form $\Gamma \vdash_t A : B$, where Γ is a context and $A : B$ is a statement. The type assignment rules can be divided in four groups, depending of the subjects of the statements:

i) *Common rules*

$$(\mathit{Proj}) \quad \frac{\Gamma \vdash_t A : s \quad a \notin \mathcal{D}om(\Gamma)}{\Gamma, a{:}A \vdash_t a : A} \qquad (\mathit{Weak}) \quad \frac{\Gamma \vdash_t A : B \quad \Gamma \vdash_t C : s \quad c \notin \mathcal{D}om(\Gamma)}{\Gamma, c{:}C \vdash_t A : B}$$

$$(\mathit{Conv}) \quad \frac{\Gamma \vdash_t A : B \quad \Gamma \vdash_t C : s \quad B =_\beta C}{\Gamma \vdash_t A : C}$$

ii) *Typed term rules*

$$(I) \quad \frac{\Gamma, x{:}\phi \vdash_t M : \psi}{\Gamma \vdash_t \lambda x{:}\phi.M : \Pi x{:}\phi.\psi} \qquad (E) \quad \frac{\Gamma \vdash_t M : \Pi x{:}\phi.\psi \quad \Gamma \vdash_t N : \phi}{\Gamma \vdash_t MN : \psi[N/x]}$$

$$(I_K) \quad \frac{\Gamma, \alpha{:}K \vdash_t M : \phi}{\Gamma \vdash_t \lambda \alpha{:}K.M : \Pi \alpha{:}K.\phi} \qquad (E_K) \quad \frac{\Gamma \vdash_t M : \Pi \alpha{:}K.\phi \quad \Gamma \vdash_t \psi : K}{\Gamma \vdash_t M\psi : \phi[\psi/\alpha]}$$

iii) *Constructor rules*

$$(C{-}I_C) \quad \frac{\Gamma, x{:}\phi \vdash_t \psi : K}{\Gamma \vdash_t \lambda x{:}\phi.\psi : \Pi x{:}\phi.K} \qquad (C{-}E_C) \quad \frac{\Gamma \vdash_t \psi : \Pi x{:}\phi.K \quad \Gamma \vdash_t M : \phi}{\Gamma \vdash_t \psi M : K[M/x]}$$

$$(C{-}I_K) \quad \frac{\Gamma, \alpha{:}K_1 \vdash_t \psi : K_2}{\Gamma \vdash_t \lambda \alpha{:}K_1.\psi : \Pi \alpha{:}K_1.K_2} \qquad (C{-}E_K) \quad \frac{\Gamma \vdash_t \phi : \Pi \alpha{:}K_1.K_2 \quad \Gamma \vdash_t \psi : K_1}{\Gamma \vdash_t \phi\psi : K_2[\psi/\alpha]}$$

$$(C{-}F_C) \quad \frac{\Gamma, x{:}\phi \vdash_t \psi : *}{\Gamma \vdash_t \Pi x{:}\phi.\psi : *} \qquad (C{-}F_K) \quad \frac{\Gamma, \alpha{:}K \vdash_t \phi : *}{\Gamma \vdash_t \Pi \alpha{:}K.\phi : *}$$

iv) *Kind rules*

$$(Axiom)\quad \frac{}{<> \vdash_t * : \Box} \qquad (K\text{–}F_C)\quad \frac{\Gamma, x{:}\phi \vdash_t K : \Box}{\Gamma \vdash_t \Pi x{:}\phi.K : \Box}$$

$$(K\text{–}F_K)\quad \frac{\Gamma, \alpha{:}K_1 \vdash_t K_2 : \Box}{\Gamma \vdash_t \Pi \alpha{:}K_1.K_2 : \Box}$$

Lemma 1.1.5 Barendregt's general typed system derives judgements of the following shapes:

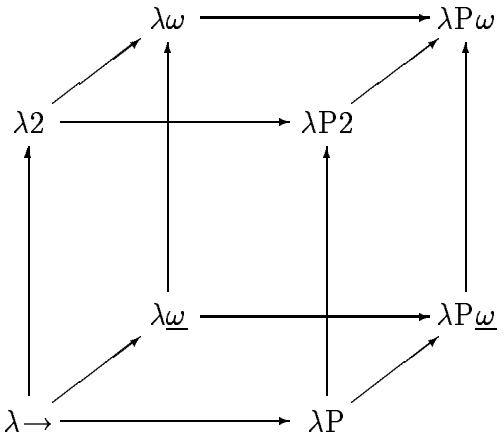
$$\Gamma \vdash_t M : \phi, \quad \Gamma \vdash_t \phi : K, \quad \text{or} \quad \Gamma \vdash_t K : \Box. \ \blacksquare$$

If $\Gamma \vdash_t M : \phi$ for a typed λ-term M, then $\Gamma \vdash_t \phi : *$ (see [1]), and ϕ is called a *type*, or to be more precise: a type with respect to the context Γ. We write $\mathcal{D}$: $\Gamma \vdash_t A : B$ when $\mathcal{D}$ is a derivation for the judgement $\Gamma \vdash_t A : B$, and $\mathcal{D}' \subseteq \mathcal{D}$ means that $\mathcal{D}'$ is a subderivation of $\mathcal{D}$.

Definition 1.1.6 i) Let the following sets of rules be defined by:
Base Rules $= \{(Axiom), (Proj), (Weak), (I), (E), (C\text{–}F_C)\}$,
Polymorphism $= \{(I_K), (E_K), (C\text{–}F_K)\}$,
Dependencies $= \{(C\text{–}I_C), (C\text{–}E_C), (K\text{–}F_C), (Conv)\}$,
Higher Order $= \{(C\text{–}I_K), (C\text{–}E_K), (K\text{–}F_K), (Conv)\}$.

ii) The eight typed systems in the Barendregt's cube can be represented by the set of derivation rules used in each system; they can be represented as vertices of the following cube:

$$\begin{aligned}
\lambda\!\!\rightarrow &= \textit{BaseRules}\\
\lambda\underline{\omega} &= \lambda\!\!\rightarrow \cup\, \textit{HigherOrder}\\
\lambda 2 &= \lambda\!\!\rightarrow \cup\, \textit{Polymorphism}\\
\lambda\omega &= \lambda 2 \cup \textit{HigherOrder}\\
\lambda \mathrm{P} &= \lambda\!\!\rightarrow \cup\, \textit{Dependencies}\\
\lambda \mathrm{P}\underline{\omega} &= \lambda\underline{\omega} \cup \textit{Dependencies}\\
\lambda \mathrm{P}2 &= \lambda 2 \cup \textit{Dependencies}\\
\lambda \mathrm{P}\omega &= \lambda\omega \cup \textit{Dependencies}
\end{aligned}$$



Let S be one of these eight systems. We write $\Gamma \vdash_S A : B$ to indicate that $\Gamma \vdash_t A : B$ can be derived using only the rules for S.

The properties of this cube are studied in [1, 8].

1.2 The Cube of Type Assignment Systems

In this subsection we will present the cube of type assignment systems as was first presented in [10]. The definition of the type assignment cube is based on the definition of an erasing function E that erases all type information from the typed terms. In fact, both the syntax of terms, and the rules of our type assignment systems are obtained directly from the corresponding syntax and rules of the typed systems, by applying a type erasure operation E, to be defined below. Note that, since terms can occur in both constructors and kinds, E can modify all typed objects. From now on, we will reserve the name *typed systems* (TS) for the systems of Barendregt's cube and we reserve the expression *type assignment systems* (TAS) for the systems to be defined below.

Definition 1.2.1 The sets of *untyped λ-terms* (Λ), *constructors* ($Cons$) and *kinds* ($Kind$) are mutually defined by the following grammar, where M, ϕ, and K are metavariables for terms, constructors and kinds respectively.

$$M ::= x \mid \lambda x.M \mid MM$$

$$\phi ::= \alpha \mid \Pi x{:}\phi.\phi \mid \Pi \alpha{:}K.\phi \mid \lambda x{:}\phi.\phi \mid \lambda \alpha{:}K.\phi \mid \phi\phi \mid \phi M$$

$$K ::= * \mid \Pi x{:}\phi.K \mid \Pi \alpha{:}K.K$$

The set T_u of *untyped terms* is the union of the sets Λ, $Cons$ and $Kind$.

Given the syntax of untyped terms, the following definition of E is natural: it erases all type information from typed λ-terms, also when they occur inside constructors or kinds.

Definition 1.2.2 The *erasing function* $E : T_t \to T_u$ is defined as follows:

$$\begin{array}{ll} E(a) & = a. \\ E(AB) & = \textit{if } B \in Cons_t \textit{ then } E(A) \textit{ else } E(A)E(B). \\ E(\Pi a{:}A.B) & = \Pi a{:}E\,(A).E\,(B). \\ E(\lambda a{:}A.B) & = \textit{if } B \in \Lambda_t \textit{ then if } A \in Kind_t \textit{ then } E(B) \textit{ else } \lambda a.E\,(B) \\ & \qquad\qquad \textit{else } \lambda a{:}E\,(A).E\,(B). \end{array}$$

The erasing function is extended to contexts in the obvious way, and we write $E(\Gamma)$. The notions of free variable and subterm are similar to their 'fully typed' counterparts.

Definition 1.2.3 Beta reduction on untyped terms can no longer be defined using a single generic rule as in Definition 1.1.2. Instead, we have the following three rules:

$$(\lambda x{:}\phi.\psi)M \to_\beta \psi[M/x], \quad (\lambda \alpha{:}K.\phi)\psi \to_\beta \phi[\psi/\alpha], \quad \text{and} \quad (\lambda x.M)N \to_\beta M[N/x].$$

Definition 1.2.4 *General type assignment system (TAS)* The rules of the general type assignment system (TAS) are used to derive *judgements* of the form $\Gamma \vdash A : B$, where Γ is a context and $A : B$ is a statement, and a statement is defined as in the typed case, using the syntax for untyped terms. The rules are:

i) The *common rules*, and the *constructor* and *kind rules* of TS, where $\vdash_t$ is replaced by $\vdash$, taking into account both the difference in syntax, and that the rule (*Conv*) now refers to the untyped reduction;

ii) The following *term rules*:

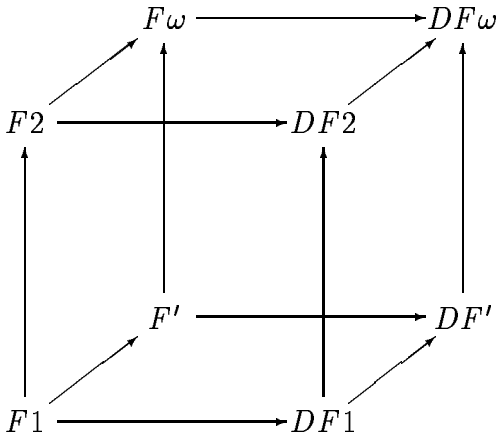
$$(I)\quad \frac{\Gamma, x{:}\phi \vdash M : \psi}{\Gamma \vdash \lambda x.M : \Pi x{:}\phi.\psi} \qquad (E)\quad \frac{\Gamma \vdash M : \Pi x{:}\phi.\psi \quad \Gamma \vdash N : \phi}{\Gamma \vdash MN : \psi[N/x]}$$

$$(I_K)\quad \frac{\Gamma, \alpha{:}K \vdash M : \phi}{\Gamma \vdash M : \Pi\alpha{:}K.\phi} \qquad (E_K)\quad \frac{\Gamma \vdash M : \Pi\alpha{:}K.\phi \quad \Gamma \vdash \psi : K}{\Gamma \vdash M : \phi[\psi/\alpha]}$$

The notion of derivation and subderivation for a judgement are the same as for TS and an analogue of Lemma 1.1.5 also holds. As before, a *type* is a constructor of kind $*$ (and again this is a context-dependent property). A λ-term M is *typable* if there are a context Γ, and a constructor ϕ such that $\Gamma \vdash M : \phi$. (We prove in Section 2 that then ϕ is a type.)

As in [10], we can distinguish eight different type assignment systems, defined using the same collection of rules given in Definition 1.1.6(i) for the TS cube. These systems can be represented as vertices of the following cube:

$$\begin{aligned} F1 &= \mathit{BaseRules} \\ F' &= F1 \cup \mathit{HigherOrder} \\ F2 &= F1 \cup \mathit{Polymorphism} \\ F\omega &= F2 \cup \mathit{HigherOrder} \\ DF1 &= F1 \cup \mathit{Dependencies} \\ DF' &= F' \cup \mathit{Dependencies} \\ DF2 &= F2 \cup \mathit{Dependencies} \\ DF\omega &= F\omega \cup \mathit{Dependencies} \end{aligned}$$



Let S denote one of the eight systems in this cube. Like for the TS we will write $\Gamma \vdash_S A : B$ to indicate that $\Gamma \vdash A : B$ can be derived using only the rules for S. Notice that in the left-hand side of the cube, both constructors and kinds coincide with the typed one, because there they cannot depend on terms. This is no longer true in the right-hand side: for example, we can build constructors like $(\lambda x{:}\phi.\psi)N$, where N is an untyped λ-term. The system $F1$ corresponds to the well-known Curry type assignment system, whereas $F2$ is the type assignment version of $\lambda 2$, which is essentially Girard's system F [11].

2 Basic properties of TAS

In this section, we will prove that all the systems in TAS cube have good computational properties; the subject reduction property, the Church-Rosser property and strong normal-

ization of typable terms will be shown. To prove these results we need more definitions and technical lemmas, stating properties of the systems, some of which are of interest in their own.

The following proposition states that every term, typable by $*$ or $\Box$, can not be typable by both, and guarantees consistency of the system.

Proposition 2.1 For every context Γ term A, and sorts s_1, s_2: if $\Gamma \vdash A : s_1$ and $\Gamma \vdash A : s_2$, then $s_1 \equiv s_2$. ■

Definition 2.2 We define the following relations on contexts:

i) $\Gamma \sqsubseteq \Gamma' \iff \Gamma$ is a prefix of Γ'.

ii) The relation $\stackrel{\cdot}{\sqsubseteq}$ is inductively defined as follows:

a) $<> \stackrel{\cdot}{\sqsubseteq} \Gamma$,

b) If $\Gamma \stackrel{\cdot}{\sqsubseteq} \Gamma'$, then $\Gamma, a{:}A \stackrel{\cdot}{\sqsubseteq} \Gamma', a{:}A$.

c) If $\Gamma \stackrel{\cdot}{\sqsubseteq} \Gamma'$, then $\Gamma \stackrel{\cdot}{\sqsubseteq} \Gamma', a{:}A$.

Theorem 2.3 *Church-Rosser.* If $A \twoheadrightarrow_\beta A'$ and $B \twoheadrightarrow_\beta B'$, then there exists C such that $A' \twoheadrightarrow_\beta C$ and $B' \twoheadrightarrow_\beta C$.

Proof: In the terminology of Klop [13], our beta reduction is a regular combinatory reduction system, and thus the Church-Rosser property follows from Theorem II.3.11 in [13]. ■

The following lemmas can be proved by easy induction on the structure of derivations.

Lemma 2.4 i) If $\Gamma \stackrel{\cdot}{\sqsubseteq} \Gamma'$, and $\Gamma \vdash A : B$, then $\mathrm{FV}(A) \cup \mathrm{FV}(B) \subseteq \mathcal{D}om(\Gamma)$, and $\Gamma' \vdash A : B$.

ii) Let $B \in \mathrm{ST}(A)$. If $\mathcal{D}$: $\Gamma \vdash A : C$, then there exist Γ', E and $\mathcal{D}' \subseteq \mathcal{D}$, such that $\mathcal{D}'$: $\Gamma' \vdash B : E$.

iii) If $\Gamma_1, c{:}C, \Gamma_2 \vdash A : B$, and $\Gamma_1 \vdash D : C$, then $\Gamma_1, \Gamma_2[D/c] \vdash A[D/c] : B[D/c]$. ■

The following lemma formulates a basic property of judgements: all predicates in derivable statements are typable.

Lemma 2.5 i) If $\Gamma \vdash E : F$, then $F \equiv \Box$ or $\Gamma \vdash F : s$.

ii) If $\Gamma \vdash M : \phi$ then $\Gamma \vdash \phi : *$, i.e. ϕ is a type with respect to the context Γ. ■

The following lemma is the key lemma for the proof of the subject reduction theorem. It states that contexts can be considered modulo β-conversion of predicates, and that a type for a term $\lambda x.M$ can always be obtained using a derivation that ends with the rule (I).

Lemma 2.6 i) Let $\Gamma_1, a{:}A, \Gamma_2 \vdash B : C$. Then $\Gamma_1, a{:}A', \Gamma_2 \vdash B : C$, for all A' such that $\Gamma_1 \vdash A' : s$ and $A =_\beta A'$.

ii) If $\Gamma \vdash \lambda x.M : \Pi x{:}\phi.\psi$, then $\Gamma, x{:}\phi \vdash M : \psi$.

Proof: i) By induction on the structure of the derivation.

ii) A judgement $\Gamma \vdash (\lambda x.M) : \theta$ can be provable only if $\theta =_\beta \prod_{i=1}^{k} \alpha_i{:}K_i.\Pi x{:}\phi'.\psi'$, for some $K_1, \ldots, K_k, \phi', \psi'$, such that $\Gamma, \alpha_1{:}K_1, \ldots, \alpha_k{:}K_k, x{:}\phi' \vdash M : \psi'$. (This can be proved by induction on derivations, using Lemma 2.4(iii).) Thus, $\Pi x{:}\phi.\psi =_\beta \prod_{i=1}^{k} \alpha_i{:}K_i.\Pi x{:}\phi'.\psi'$, and since these two expressions have a common reduct, it must be that $k = 0$ and that $\phi =_\beta \phi'$ and $\psi =_\beta \psi'$. So $\Gamma, x{:}\phi' \vdash M : \psi'$, and thus $\Gamma, x{:}\phi \vdash M : \psi$ follows from part (i) and rule (*Conv*). ■

Theorem 2.7 *Subject Reduction for Terms.* If $\Gamma \vdash M : \psi$ and $M \to_\beta N$ then $\Gamma \vdash N : \psi$.

Proof: By induction on the definition of $\to_\beta$. The main case is $M \equiv (\lambda x.P)Q$ and $N \equiv P[Q/x]$, the others follow by induction. Let $\mathcal{D}$ be a derivation for $\Gamma \vdash M : \psi$. It is not difficult to see that $\mathcal{D}$ has the following structure:

$$\mathcal{D}_1{:}\ \dfrac{\begin{array}{cc}\vdots & \vdots \\ \Gamma' \vdash (\lambda x.P) : \Pi x{:}\phi'.\psi' & \Gamma' \vdash Q : \phi'\end{array}}{\Gamma' \vdash (\lambda x.P)Q : \psi'[Q/x]}\,(E)$$

$$\vdots$$

$$\mathcal{D}{:}\ \dfrac{}{\Gamma \vdash (\lambda x.P)Q : \psi}$$

That is, there is a subderivation $\mathcal{D}_1$, ending with an application of rule (E), which is followed by a (possibly empty) sequence of applications of the not syntax-directed rules (*Proj*), (*Weak*), (*Conv*), (I_K) and (E_K). By Lemma 2.6(ii) we obtain: $\Gamma', x{:}\phi' \vdash P : \psi'$. Since also $\Gamma' \vdash Q : \phi'$, by Lemma 2.4(iii) we obtain $\Gamma' \vdash P[Q/x] : \psi'[Q/x]$. Apply the same rules as used to go from $\mathcal{D}_1$ to $\mathcal{D}$ to obtain $\Gamma \vdash P[Q/x] : \psi$. ■

An important property of the type assignment systems is strong normalization of typable terms; this is already known to hold for the systems $F\omega$, $F1$, $F2$, and F' (see [10]). Using this result, we will show that it also holds for the other four systems of the cube of type assignment systems. To achieve this, we use the function ED that 'erases dependencies' as defined in [10]. For the behaviour of the function ED on beta redexes, there are the following possibilities:

i) $ED\,((\lambda x.M)N) = (\lambda x.ED\,(M))(ED\,(N))$

ii) $ED\,(M[N/x]) = ED\,(M)[ED\,(N)/x]$;

iii) $ED\,((\lambda\alpha{:}K.\phi)\psi) = (\lambda\alpha{:}ED\,(K).ED\,(\phi))(ED\,(\psi))$

iv) $ED\,(\phi[\psi/\alpha]) = ED\,(\phi)[ED\,(\psi)/\alpha]$;

v) $ED\,((\lambda x{:}\phi.\psi)M) = ED\,(\psi)$;

vi) $ED\,(\psi[M/x]) = ED\,(\psi)$.

That is, $A \to_\beta B$ implies either $ED\,(A) \to_\beta ED\,(B)$ or $ED\,(A) \equiv ED\,(B)$.

Theorem 2.8 *Termination* If $\Gamma \vdash A : B$ then A is strongly normalizing.

Proof: In [10], Theorem 2.2.1 states that if $\Gamma \vdash A : B$ is a derived judgement in $DF\omega$ ($DF1$, $DF2$, DF'), then $ED(\Gamma) \vdash ED(A) : ED(B)$ is derivable in $F\omega$ ($F1$, $F2$, F'). Suppose now that $A \equiv A_0 \rightarrow_\beta A_1 \rightarrow_\beta A_2 \rightarrow_\beta \ldots$ is a sequence of beta reductions. By the property mentioned above, for every $i \geq 1$, either $ED(A_i) \rightarrow_\beta ED(A_{i+1})$, or $ED(A_i) \equiv ED(A_{i+1})$. Suppose the sequence $A_0 \rightarrow_\beta A_1 \rightarrow_\beta A_2 \rightarrow_\beta \ldots$ is infinite. Since beta reduction in $F\omega$ ($F1$, $F2$, F') is strongly normalizing, there is an n such that $ED(A_j) \equiv ED(A_{j+1})$, for every $j \geq n$. So from step n, every step in the infinite sequence $A_0 \rightarrow_\beta A_1 \rightarrow_\beta A_2 \rightarrow_\beta \ldots$ corresponds to a reduction of a 'bad' redex of the form $(\lambda x{:}\phi.\psi)M$. However, since M is an untyped term, such a reduction cannot create new 'bad' redexes. Thus the number of redexes must decrease after every step, and our reduction can not be infinite. ■

3 The relation between TS and TAS

In this section we will focus on the relation between Barendregt's cube and the cube of type assignment systems. First we introduce the notions of *consistency, similarity,* and *isomorphism* between typed systems and type assignment systems.

Definition 3.1 Let S_t and S_u be systems in corresponding vertices of TS and TAS cube.

i) S_t and S_u are *consistent* if $\Gamma_t \vdash_{S_t} A_t : B_t$ implies $E(\Gamma_t) \vdash_{S_u} E(A_t) : E(B_t)$.

ii) S_t and S_u are *similar* if they are consistent and, moreover, $\Gamma \vdash_{S_u} A : B$ implies that there exists Γ_t, A_t, and B_t satisfying $\Gamma_t \vdash_{S_t} A_t : B_t$ and $E(\Gamma_t) = \Gamma$, $E(A_t) \equiv A$, and $E(B_t) \equiv B$.

iii) Let $\mathcal{D}er_t$ and $\mathcal{D}er_u$ be the set of all the derivations in S_t and S_u. S_t and S_u are *isomorphic* if and only if there are : $\mathcal{F}$: $\mathcal{D}er_t \rightarrow \mathcal{D}er_u$ and $\mathcal{G}$: $\mathcal{D}er_u \rightarrow \mathcal{D}er_t$ such that:

   a) If $\mathcal{D}_t$: $\Gamma \vdash_{S_t} A : B$ then $\mathcal{F}(\mathcal{D}_t) : E(\Gamma) \vdash_{S_u} E(A) : E(B)$.

   b) $\mathcal{F} \circ \mathcal{G}$ and $\mathcal{G} \circ \mathcal{F}$ are the identity on $\mathcal{D}er_t$ and $\mathcal{D}er_u$ respectively.

   c) Both $\mathcal{F}$ and $\mathcal{G}$ preserve the structure of the derivations, (i.e. the tree obtained from the derivation by erasing all the judgements but not the names of the rules).

The definition of isomorphism between two systems was already given in [10], but in a less general way. Two systems are isomorphic according to the definition in [10], if they are isomorphic in the sense of the preceding Definition, and moreover, the function $\mathcal{F}$ is such that $\mathcal{F}(\mathcal{D}_t)$ is obtained from $\mathcal{D}_t$ by applying the erasing function to all terms in $\mathcal{D}_t$; by abuse of notation, we denote $\mathcal{F}(\mathcal{D}_t)$ by $E(\mathcal{D}_t)$. The following Proposition proves that the two notions of isomorphism coincide, in case of the TAS cube:

Proposition 3.2 Let S_t and S_u be systems in corresponding vertices of TS and TAS cube respectively, and suppose they are isomorphic through the functions $\mathcal{F}$ and $\mathcal{G}$. Then for every typed derivation $\mathcal{D}_t$, $\mathcal{F}(\mathcal{D}_t) = E(\mathcal{D}_t)$. ■

The following results are taken from [10]:

Theorem 3.3 Let S_t and S_u be systems in corresponding vertices of TS and TAS cube.

i) S_t and S_u are consistent.

ii) If S_t and S_u do not contain *Dependencies* as subset of their sets of rules, then S_t and S_u are isomorphic.

iii) If the assumption of (ii) is not satisfied, then S_t and S_u are not isomorphic.

Proof: See [10]. The proof uses the following properties of the erasing function:

i) $E(A[B/a]) \equiv E(A)[E(B)/a]$;

ii) If $A \twoheadrightarrow_\beta C$, then $E(A) \twoheadrightarrow_\beta E(C)$. ■

After the negative result of Theorem 3.3(iii), it is natural to ask if the corresponding systems in the TS and TAS cubes are at least similar. Such a conjecture was already stated in [10]. This property holds only for the systems without polymorphism, as will be shown in Theorem 3.7, namely, for $DF1$ versus λP, and for F' versus $\lambda\underline{\omega}$. Adding polymorphism makes a difference: the systems with both polymorphism and dependencies are not similar.

Theorem 3.4 Let S_t be either λP2 or λPω, and let S_u be respectively $DF2$ and $DF\omega$. Then S_t and S_u are not similar.

Proof: As a counterexample, we show a derivable judgement of $DF2$, that cannot be obtained as an erasure of any derivable judgement in λPω. In this proof, for reasons of readability, we will use the notation $A{\to}B$ for $\Pi a{:}A.B$, when a does not occur in B. Let Γ_0 denotes a context consisting of the following declarations:

(type variables)	$\alpha{:}*,\ \beta{:}*,\ \gamma{:}*,\ \delta{:}*,$
(constructor variable)	$\epsilon{:}(\beta{\to}*),$
(term variables)	$u{:}(\Pi\eta{:}*.((\eta{\to}\eta){\to}\alpha){\to}\beta),\ x{:}\alpha,\ y{:}\gamma,\ z{:}\delta,$

and let M, M_0, M_1 denote respectively the following untyped λ-terms:

$$M \equiv u(\lambda f.x), \quad M_0 \equiv u(\lambda f.\mathbf{K}x(fy)), \quad \text{and} \quad M_1 \equiv u(\lambda f.\mathbf{K}x(fz))$$

where the symbol $\mathbf{K}$ denotes the term $(\lambda xy.x)$. Clearly, both M_0 and M_1 beta-reduce to M, and all these terms can correctly be assigned the type β in the context Γ_0. Thus, one can derive:

$$\Gamma_0 \vdash \epsilon M_0{\to}\alpha : * \quad \text{and} \quad \Gamma_0 \vdash \epsilon M_1 : *$$

and this means that the context $\Gamma{=}\ \Gamma_0, p{:}\epsilon M_0{\to}\alpha, q{:}\epsilon M_1$ is legal. With help of rules (*Proj*) and (*Conv*), one can easily derive:

$$\Gamma \vdash pq : \alpha.$$

The above judgement cannot be obtained as an erasure of any judgement $\Gamma' \vdash N : \phi$ derivable in λP2 or λPω, (i.e. one cannot have $E(\Gamma') = \Gamma$, $E(N) \equiv pq$, and $E(\phi) \equiv \alpha$). Assume the opposite. First note that $\phi \equiv \alpha$, since no terms occur in α. (The erasing function can only modify types containing occurrences of terms, in which case the results must also contain

terms.) Similarly, Γ' may differ from Γ only in the declarations of p and q, which must be of the form:

$$p{:}\epsilon M_0' \to \alpha \quad \text{and} \quad q{:}\epsilon M_1'$$

where $E(M_0') \equiv M_0$ and $E(M_1') \equiv M_1$. Without loss of generality (see Theorem 2.8), we can assume that M_0' and M_1' are normal forms. We can also assume that N is of the form PQ, where $E(P) \equiv p$ and $E(Q) \equiv q$ (otherwise we consider an appropriate subterm of N instead). Since P is applied to Q, and the type of PQ is α, P must have a type of the form $\epsilon M_0'' \to \alpha$, where $E(M_0'') \equiv M_0$, and Q must have a type of the form $\epsilon M_1''$, where $E(M_1'') \equiv M_1$. In order to make the application well-typed (after a possible series of applications of rule $(Conv)$), it must be the case that $M_0'' =_\beta M_1''$.
It follows that we have beta-convertible terms M_0'', M_1'', which erase to M_0 and M_1, respectively, and both are of type β. Without loss of generality, we can assume that these terms have no beta-redexes involving polymorphic abstraction/application, and thus we may write:

$$M_0'' \equiv u\gamma(\lambda f{:}\gamma \to \gamma.K_0 x(fy)) \qquad M_1'' \equiv u\delta(\lambda f{:}\delta \to \delta.K_1 x(fz))$$

where K_0 and K_1 are such that $E(K_0) \equiv \mathbf{K}$ and $E(K_1) \equiv \mathbf{K}$. The types of f used in the above are forced by the applications fy and fz. Note that the type of f may not be externally quantified, because of the type of the polymorphic variable u. The normal forms of these terms are as follows: M_0'' reduces to $u\gamma(\lambda f{:}\gamma \to \gamma.x)$, while M_1'' reduces to $u\delta(\lambda f{:}\delta \to \delta.x)$. But these normal forms are different, and this contradicts the previous claim that $M_0'' =_\beta M_1''$. ■

The cause of the phenomenon demonstrated in the last proof, is the polymorphic variable. If polymorphism is not permitted, we can prove that the corresponding TS and TAS are similar. This requires a sequence of lemmas. In what follows, the symbol $\vdash$ denotes $\vdash_S$, for $S \in \{F1,\ F',\ DF1,\ DF'\}$, while $\vdash_t$ refers to the corresponding TS systems, i.e. we consider only systems without polymorphism.

Lemma 3.5 i) Suppose $\Gamma \vdash_t B_1 : A$ and $\Gamma \vdash_t B_2 : A$, and let both B_1 and B_2 be normal forms. If $E(B_1) \equiv E(B_2)$ then $B_1 \equiv B_2$.

ii) Let $\Gamma \vdash_t B_1 : A$ and $\Gamma \vdash_t B_2 : A$. If $E(B_1) =_\beta E(B_2)$, then $B_1 =_\beta B_2$.

Proof: i) By induction on the structure of B_1.

ii) Easy, using part (i). ■

Lemma 3.6 Suppose that $\Gamma \vdash A : B$. Then the following conditions hold:

i) There exists a typed context Γ_t, and typed terms A_t, B_t satisfying $E(\Gamma_t) = \Gamma$, $E(A_t) \equiv A$ and $E(B_t) \equiv B$, and such that $\Gamma_t \vdash_t A_t : B_t$.

ii) For every typed context Γ_t, and every typed term B_t satisfying $E(\Gamma_t) = \Gamma$, $E(B_t) \equiv B$ and $\Gamma_t \vdash_t B_t : s$, there exists a typed term A_t, such that $\Gamma_t \vdash_t A_t : B_t$, and $E(A_t) \equiv A$.

Proof: Parts (i) and (ii) can be proven by mutual induction on the structure of derivations. ■

Theorem 3.7 Let S_t be a TS system whose set of rules does not contain *Polymorphism* as subset, and let S_u be the corresponding TAS system. Then S_t and S_u are similar.

Proof: By Lemma 3.6. ■

4 How to obtain an isomorphism

In this section we show that it is possible to define another erasing function (which looks less natural), named E', that gives rise to a second type assignment cube TAS$'$ which is isomorphic to the TS cube. The main difference between E and E' is that, while E always erases type information in terms, E' is context dependent and erases type information from a term only if that term does not occur in a type; otherwise it leaves the term unchanged. So the difference between TAS$'$ and TAS is that dependent types of TAS$'$ contain occurrences of typed λ-terms rather than untyped λ-terms. The systems without *Dependencies* coincide exactly with the corresponding systems in the TAS cube. Also, either with *Dependencies* or without, the provable judgements are the same as long as their subjects are either constructors or kinds.

Definition 4.1 *The TAS$'$ Cube.* i) The untyped and typed terms, typed constructors and typed kinds are defined as before (Definitions 1.1.1 and 1.2.1). Let T'_u be the union of the sets $\Lambda, Cons_t$ and $Kind_t$.

ii) The new erasing function E': $T_t \to T'_u$ is defined as follows:

a) $E'(M) = E(M)$.
b) $E'(\phi) = \phi$.
c) $E'(K) = K$.

iii) Let M range over Λ, and A, B, and ϕ range over T_t. The general type assignment system induced by E'(TAS$'$) proves judgements of the following form:

$$\Gamma \vdash' M : \phi \quad \text{and} \quad \Gamma \vdash_t A : B, \text{ where } A \notin \Lambda_t.$$

iv) The type assignment rules are:

a) All the rules used for TS.
b) The rules (I), (I_K), and (E_K) of TAS (where $\vdash$ should be replaced by $\vdash'$).
c) The rules:

$$(Proj') \quad \frac{\Gamma \vdash_t \phi : * \quad x \notin \mathcal{D}om(\Gamma)}{\Gamma, x{:}\phi \vdash' x : \phi} \qquad (Weak') \quad \frac{\Gamma \vdash' M : \phi \quad \Gamma \vdash_t A : s \quad a \notin \mathcal{D}om(\Gamma)}{\Gamma, a{:}A \vdash' M : \phi}$$

$$(Conv') \quad \frac{\Gamma \vdash' M : \phi \quad \Gamma \vdash_t \psi : * \quad \phi =_\beta \psi}{\Gamma \vdash' M : \psi} \qquad (E') \quad \frac{\Gamma \vdash' M : \Pi x{:}\phi.\psi \quad \Gamma \vdash_t N : \phi}{\Gamma \vdash' M(E'(N)) : \psi[N/x]}$$

v) As in Definition 1.1.6(i), the rules can be grouped in sets. All the collections are as before, with the exception of (with abuse of notation): *Base Rules* $= \{(Axiom), (Proj), (Weak), (I), (E), (C{-}F_C), (Proj'), (Weak'), (E')\}$, Again eight type assignment systems

can be defined, whose relationships can be represented as before by drawing a cube. A system in this cube and one in the TS-cube are *corresponding* if the names for the set of rules allowed for these systems are exactly the same.

The main result on the relationship between the TS cube and the TAS′ cube is:

Theorem 4.2 Let S_t be any typed system in the TS cube, and let S_u be the corresponding system in the TAS′ cube. Then S_t and S_u are isomorphic.

Proof: The function $\mathcal{F}$: $\mathcal{D}er_t \to \mathcal{D}er_u$ can be defined by induction on the structure of $\mathcal{D} \in \mathcal{D}er_t$ in the following way:

i) If $\mathcal{D}_t$: $\Gamma \vdash_t A : B$ and $A \notin \Lambda_t$, then $\mathcal{F}(\mathcal{D}_t) = \mathcal{D}_t$.

ii) If the last rule of $\mathcal{D}_t$ is (E), i.e.:

$$\mathcal{D}_t: \frac{\mathcal{D}_1 : \Gamma \vdash_t M : \Pi x{:}\phi.\psi \quad \Gamma \vdash_t N : \phi}{\Gamma \vdash_t MN : \psi[N/x]}\ (E)$$

then $\mathcal{F}(\mathcal{D}_1)$: $E'(\Gamma) \vdash' E'(M) : E'(\Pi x{:}\phi.\psi)$. Since $E'(\Gamma) = \Gamma$ and $E'(\Pi x{:}\phi.\psi) \equiv \Pi x{:}\phi.\psi$, we can define:

$$\mathcal{F}(\mathcal{D}): \frac{\Gamma \vdash' E'(M) : \Pi x{:}\phi.\psi \quad \Gamma \vdash_t N : \phi}{\Gamma \vdash' E'(M)E'(N) : \psi[N/x]}\ (E')$$

iii) if the last rule is one of the other not mentioned, the definition of $\mathcal{F}$ is given by straightforward induction.

The definition of $\mathcal{G}$ is left to the reader. It is easy to verify that these two functions realize an isomorphism between the corresponding systems in the two cubes. ■

While the definition of the erasing function E' is (relatively) easy, the definition of the related cube is very involved. This is a consequence of the fact that, for systems with dependencies, the derivations are not compositional. Namely if $\mathcal{D}_t$ is a derivation and $\mathcal{D}'_t$ is a subderivation of $\mathcal{D}_t$ that ends with a judgement of the form $\Gamma \vdash_t M : \phi$, for $M \in \Lambda_t$, then $\mathcal{D}'_t$ need not be a valid derivation; this is because E' has a context dependent behaviour. This is the price we paid for reaching the isomorphism with the typed systems.

5 Conclusions

This paper, together with [10], can be seen as the first attempt to study type assignment systems with dependent types. In fact all the systems in the dependencies free part of the cubes TAS and TAS′ have been extensively studied in the literature. The only type assignment system with dependent types already defined in the literature is the system $\lambda\Pi$ of Dowek [7]. Strictly speaking, this is not a type assignment system in the usual sense. There are no derived judgements, instead, a valid judgement of $\lambda\Pi$ is defined as one of

the form $E'(\Gamma) \vdash' E'(A) : E'(B)$, where $\Gamma \vdash_t A : B$ is a valid judgement of λP. So Dowek's system is equivalent to the system corresponding to λP in the TAS$'$ cube. For this system, the type checking problem was shown to be undecidable in [7]. The method of proof of the undecidability is however applicable for all our systems with dependencies. We showed that all the systems with dependencies we defined enjoy good computational properties, and we focused our attention in particular on the relationship between typed and type assignment systems. A further step can be made by looking for a type assignment counterpart to the Generalised Type Systems, as defined in [1, 2, 3].

References

[1] Barendregt, H.P., Lambda Calculi with Types, *Handbook of Logic in Computer Science*, Abramsky, Gabbai, Maibaum eds., Oxford University Press, 1991.

[2] Barendregt, H.P., Introduction to Generalised Type Systems, *Journal of Functional Programming*, volume 1(2), 125–154, 1991.

[3] Berardi, S., *Towards a Mathematical Analysis of Type Dependence in Coquand–Huet Calculus of Constructions and the Other Systems in Barendregt's Cube*, Department of Computer Science, CMU, and Dipartimento di Matematica, Torino, 1988.

[4] Curry, H.B., Modified Basic Functionality in Combinatory Logic, *Dialectica*, 1969.

[5] Coquand, T., Metamathematical Investigations of a Calculus of Constructions, *Logic and Computer Science*, Odifreddi ed., Academic Press, 91–122, 1990.

[6] Coquand, T. and Huet, G., The Calculus of Constructions, *Information and Computation*, 76(2,3), 95–120, 1988.

[7] Dowek, G., The Undecidability of Typability in the Lambda-Pi-Calculus, *Proc. Typed Lambda Calculi and Applications*, *LNCS* 664, 139–145, 1993.

[8] Geuvers, H. and Nederhof, M., Modular Proof of Strong Normalization for the Calculus of Constructions, *Journal of Functional Programming*, 1(2), 155–189, 1991.

[9] Giannini, P. and Ronchi Della Rocca, S., Characterization of Typings in Polymorphic Type Discipline, *Proc. Logic in Computer Science*, IEEE, 61–70, 1988.

[10] Giannini, P., Honsell, F. and Ronchi Della Rocca, S., Type Inference: Some Results, Some Problems, *Fundamenta Informaticae*, 19(1,2), pp.87–126, 1993.

[11] Girard, J.Y., The System F of Variable Types, Fifteen Years Later, *Theoretical Computer Science*, 45, 159–192, 1987.

[12] Harper, B., Honsell, F. and Plotkin, G., A Framework for Defining Logics, *Journal of the ACM*, 40, 1993.

[13] Klop, J. W., *Combinatory Reduction Systems*, PhD-thesis, Rijksuniversiteit Utrecht, 1980.

[14] Leivant, D., Polymorphic Type Inference, In *Symposium on Principles of Programming Languages*, ACM, 88–98, 1983.

[15] Reynolds, J.C., Towards a Theory of Type Structures, *Proc. Paris Colloquium on Programming*, Springer Verlag, 408–425, 1974.