Comparing Cubes of Typed and Type Assignment Systems

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

Paweł Urzyczyn[2†]

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

[2] 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 [13], and confront it with Barendregt's typed λ-cube [4]. The first is obtained from the latter through applying a natural type erasing function E to derivation rules, that erases type information from terms. In particular, we address the question whether a judgement, derivable in a type assignment system, is always an erasure of a derivable judgement in a corresponding typed system; we show that this property holds only for the systems without polymorphism. The type assignment systems we consider satisfy the properties 'subject reduction' and 'strong normalization'. Moreover, we define a new type assignment cube that is isomorphic to the typed one.

Introduction

Types can be used as predicates for terms of λ-calculus in two different ways. A first approach is to define terms directly *decorated* with types; in this *fully typed* approach, every closed term comes with a unique, intrinsic type. A *typed system* consists of a set of (derivation) 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 that the term M_t has type ϕ_t under the context Γ_t, where Γ_t records 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. A *type assignment system* consists of a set of (derivation) rules for proving judgements of the shape $\Gamma \vdash M : \phi$, where M is a term of the untyped λ-calculus, ϕ is a type, and Γ is a context that assigns types to the free variables of M and ϕ. Such a judgement can be understood as that we can assign the type ϕ to the λ-term M, when types are assigned to the free variables of M and ϕ as specified in the context Γ. In this approach, types are viewed as *predicates*, or *properties*, of terms, and each closed term can be assigned either none or infinitely many types.

When we look at λ-calculus as a paradigmatic programming language, the first approach corresponds to explicitly typed languages, like for example Haskell, whereas the second corresponds to ML-like languages, where the user can write programs in a completely untyped language, and types are automatically inferred at compile time. The latter can be considered

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

†Partly supported by grants NSF CCR–9113196, KBN 2 P301 031 06, and by a grant from the Commission of The European Communities ERB–CIPA–CT92–2266(294).

to be the construction of an abstract interpretation of the program, that can be used as a correctness criterion.

In the typed approach, called *à la Church* by Barendregt, there exists several typed λ-calculi, where terms are decorated with types in various ways. Examples of typed λ-calculi are the simply typed λ-calculus ($\lambda\!\!\rightarrow$) of Church, the second order λ-calculus of Girard and Reynolds ($\lambda 2$) [15,20], and the calculus of constructions ($\lambda \mathrm{P}\omega$) [7,8]. Barendregt gave in [4] a compact and appealing presentation of a class of typed systems (TS), 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; the three dimensions of the cube represent the introduction of some new rules of type formation, namely *Polymorphism*, *Higher-Order* and *Dependencies* (see Definition *1.5*). This three-dimensional structure allows for a deep comparative analysis of different typed λ-calculi.

In [13], a type assignment version of Barendregt's cube (TAS) was defined trough an erasing function E that erases type information from terms. To be precise, if a typed system consists of a set R of derivation rules, the rules of the corresponding type assignment system can be obtained by applying E to every object occurring in the rules of R. The dependency-free plane of TAS contains some type assignment systems already known in the literature, that are convertible to certain typed systems: the Curry type assignment system ($F1$) [9] that corresponds to $\lambda\!\!\rightarrow$, the polymorphic type assignment system ($F2$) [17] that corresponds to $\lambda 2$, and the higher order type assignment system ($F\omega$) [14] that corresponds to $\lambda\omega$.

The fact that in [13] also systems that contain dependencies were considered, was a first attempt to study dependent types in a type assignment approach. In that paper was proved that the introduction of dependencies does not increase the expressiveness of a system, i.e., the terms typable in a type assignment system with dependencies are all nothing but those typable in the similar system, obtained from the first by erasing the dependencies.

In [13], it was observed that, perhaps surprisingly, in presence of dependencies there no longer exists an isomorphism between corresponding systems of typed and type assignment cubes, in the sense that not every derivation in TAS is the image under erasure of a derivation in TS. However, in that paper was conjectured that at least there exists an isomorphism between judgements rather than between derivations, i.e.: 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 $E(M_t) = M$, $E(\Gamma_t) = \Gamma$, and $E(\phi_t) = \phi$.

In this paper, where we focus closely on the differences and similarities between TS and TAS, we will disprove this conjecture, showing that it is true only for systems without polymorphism. The type assignment systems with polymorphism and dependencies ($DF2$ and $DF\omega$, that correspond respectively to $\lambda \mathrm{P}2$ and $\lambda \mathrm{P}\omega$) are in some sense more powerful than their typed versions. In fact, we prove that there are judgements, provable in one of these systems, that cannot be obtained as erasures of typed judgements. This implies that there are types, inhabited in these systems, that are not erasures of inhabited types in the corresponding typed systems, and, moreover, that a term M can be assigned more types than just those that can be obtained, through erasure, from types belonging to any typed version of M.

This result gives then rise to a new question, namely if it is possible to build a cube of type assignment systems that is isomorphic to Barendregt's cube, in the sense that typed and type assignment systems in the corresponding vertices are isomorphic. We solve this problem by defining a cube of type assignment systems TAS$'$ that enjoy this property. This cube is based on the definition of 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. This cube has the the (somewhat unelegant) property that some type assignment rules use explicitly typed rules of the corresponding typed system in Barendregt's cube. But this seems to be the price to pay for obtaining isomorphism.

As stated already in [13], the above mentioned erasing function E, at least for the dependency-free plane of TS and TAS, induces an isomorphism between derivations in corresponding systems. More precisely, if $\mathcal{D}$ is a derivation in a typed system, by applying E to every object (i.e. term, constructor, or kind) in $\mathcal{D}$, a valid derivation in the corresponding type assignment system is obtained. Vice-versa, again only for dependency-free systems, every type assignment derivation can be obtained by applying E to a typed one. Clearly, the fact that the classes of derivations for typed and a type assignment systems are isomorphic means that they have the same underlying logical system.

The relation with (intuitionistic) logic through the so-called Curry-Howard isomorphism, or '*formulae as types*' principle, has been profoundly studied for Barendregt's cube, and has been clearly established for the plane of the cube without dependencies. However, in the opposite plane, this relation is less clear, as demonstrated by Berardi in [6]. As mentioned above, in this paper, we show an example of a inhabited type in TAS, that cannot be obtained through erasure of an inhabited type in TS. This negative result of course implies that the logical sides of these two cubes are different; however, this difference only shows up in the plane of the cube with dependencies, where already TS has lost a clear connection with logic. Moreover, the underlined logics of the cube TAS$'$ are those of the typed cube of Barendregt.

Furthermore, it is also our opinion that there is more to types than just logic: studying types is not solely justifiable through the connection between types and logic, as is clearly shown by, for example, the type system developed for ML that models type-constants and recursion [18], and the intersection type discipline [2]. In our view, the main motivation for TAS comes from the ML-style of approaching types: to have type-free code with type assignment seen as a correctness criterion, or safety means, but always outside of programs rather than built in. Certainly, in order to be correctly applied in this way, a type assignment system must enjoy some fundamental properties, like the Church-Rosser property, the subject-reduction property and normalization. We prove these properties in this paper for all systems in TAS. So, TAS can make sense even if it does not fit the corresponding TS: it is just another way to select legitimate code. Studying type systems with dependencies can be of value from the point of view of abstract interpretation; such type assignment system could introduce a more refined notion of types in a programming language setting. For example, since the version of $F1$ with dependencies is decidable, and the core of the type system for ML is based on $F1$, designing a version of ML with dependent types seems feasible.

We would like to emphasize that the scope of this paper is to *compare* the systems TS and TAS, not to propagandize any of these.

This paper is organized as follows. Section *1* contains a presentation of Barendregt's cube in a stratified version, and of a cube of type assignment systems. In Section *2*, the properties of the type assignment systems belonging to the latter are studied; in particular, it contains the proofs of the subject reduction property, and of the strong normalization property. Section *3* is devoted to the study of the relation between the two cubes. In that section, we disprove the conjecture cited above. In Section *4*, a new erasing function, together with the induced new cube of type assignment systems is presented. In that section, we will show that these type assignment systems are isomorphic to the systems in Barendregt's cube.

A preliminary version of this paper was presented in [1].

Notational conventions: In this paper, a *term* will be either 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, μ, ... range over constructors; K ranges over kinds; s ranges over sorts: 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. The symbol Γ will range over contexts. All symbols can appear indexed. The symbol $\equiv$ denotes the syntactic identity of terms, and we will consider terms modulo α-conversion. The notation $\Pi_{i=1}^{n} a_i{:}A_i.B$ is an abbreviation of $\Pi a_1{:}A_1.\cdots\Pi a_n{:}A_n.B$.

1 Two Cubes

Barendregt's cube of typed systems, already defined in [4], is normally presented using a rather compact notation, using *rule schemes* rather than rules. Before coming to the definition of a cube of type assignment systems related to Barendregt's cube, in this section we will first present a 'stratified' version of the systems in that cube, by splitting the terms considered by Barendregt in three different classes, being those of λ-terms, constructors, and kinds. Starting from that stratified version, we will define an erasing function E and, using this function, obtain the related cube of type assignment systems. The same approach can be found in [13].

1.1 The Cube of Typed Systems

In this subsection we will give a short overview of Barendregt's cube. A number of formal notions and properties for this cube (like 'free variable', 'substitution', or 'context') are used in this paper; however, in view of the strong similarity with definitions given in Subsection *1.2*, we will skip those here. Here we will limit ourselves to the presentation of the formal syntax and derivation rules in our own denotation, since that differs from the one commonly used; this should enable the appreciation of the presentation of our cube of type assignment systems in the next subsection. For a complete development of Barendregt's cube, we refer to [4,11].

Definition 1.1 *i)* $\{*, \Box\}$ is the set of *sorts*.

ii) 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 λ-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 $Kind_t$.

Definition 1.2 (Typed reduction) β-reduction on typed terms (denoted as $\twoheadrightarrow_\beta$) is defined as usual, i.e., as the contextual, reflexive and transitive closure of the following one-step reduction rule:

$$(\lambda a{:}A.B)C \to_\beta B[C/a].$$

The symbol $=_\beta$ denotes β-conversion, i.e., the least equivalence relation generated by $\twoheadrightarrow_\beta$.

The introduction of three classes of 'terms' in Definition *1.1* induces a stratified version of the set derivation rules; each class comes with its own derivations rules. The names of the rules

are, to save space, restricted to a few characters. We have tried to use an orthogonal approach in baptizing the rules: in general a name for a rule is composed like $(X{-}Y_Z)$, meaning that:

- it is a rule that follows the syntax of objects in class X, where X is omitted for λ-terms, is C for constructors, and K for kinds,
- Y is either
  - I for an introduction rule, that are used to deal with the various λ-abstractions,
  - E for an elimination rule, that deal with applications,
  - F for a formation rule, that deal with the Π-abstraction,
- and Z is used (as X above) to indicate the class either of the bound variable (in case of an introduction or formation rule), or of the right-hand side term in an application (in case of a formation rule).

Definition 1.3 (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 derivation rules can be divided in four groups, depending of the subjects of the statements:

i) Common Rules

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

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

ii) Typed λ-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) Typed 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) Typed Kind Rules

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

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

If $\Gamma \vdash_t M : \phi$ for a typed λ-term M, then $\Gamma \vdash_t \phi : *$ (see [4]). In this case we say that ϕ is a *type* or, to be more precise, a type with respect to the context Γ.

In the next definition we present a notation for derivations, that is of use in the sequel.

Definition 1.4 *i)* We write $\mathcal{D} :: \Gamma \vdash_t A : B$ to express that $\mathcal{D}$ is a derivation for the judgement $\Gamma \vdash_t A : B$.

ii) We write $\mathcal{D}' \subseteq \mathcal{D}$ when $\mathcal{D}'$ is a subderivation of $\mathcal{D}$.

iii) We will use the notation

$$\mathcal{D} :: \frac{\mathcal{C}_1 \quad \cdots \quad \mathcal{C}_n}{\mathcal{C}} \, (R)$$

to denote the derivation $\mathcal{D}$, proving the judgement $\mathcal{C}$, that is obtained by applying the rule (R) to the premises $\mathcal{C}_1, \ldots, \mathcal{C}_n$, which are conclusions of some derivations.

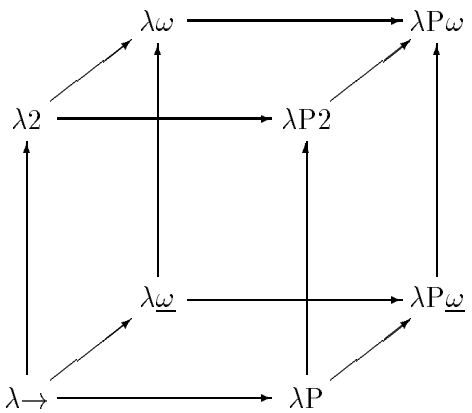
Definition 1.5 *i)* Let the following sets of rules be defined by:

$$\begin{aligned}
\mathit{Base\text{-}Rules} &= \{(\mathit{Axiom}), (\mathit{Proj}), (\mathit{Weak}), (I), (E), (C\text{–}F_C)\}, \\
\mathit{Polymorphism} &= \{(I_K), (E_K), (C\text{–}F_K)\}, \\
\mathit{Dependencies} &= \{(C\text{–}I_C), (C\text{–}E_C), (K\text{–}F_C), (\mathit{Conv})\}, \\
\mathit{Higher\text{-}Order} &= \{(C\text{–}I_K), (C\text{–}E_K), (K\text{–}F_K), (\mathit{Conv})\}.
\end{aligned}$$

ii) The eight typed systems in Barendregt's cube can be represented by the set of derivation rules used in each system.

$$\begin{aligned}
\lambda{\to} &= \mathit{Base-Rules}, \\
\lambda\underline{\omega} &= \lambda{\to} \cup \mathit{Higher-Order}, \\
\lambda 2 &= \lambda{\to} \cup \mathit{Polymorphism}, \\
\lambda\omega &= \lambda{\to} \cup \mathit{Higher-Order} \cup \mathit{Polymorphism}, \\
\lambda\mathrm{P} &= \lambda{\to} \cup \mathit{Dependencies}, \\
\lambda\mathrm{P}\underline{\omega} &= \lambda{\to} \cup \mathit{Dependencies} \cup \mathit{Higher-Order}, \\
\lambda\mathrm{P}2 &= \lambda{\to} \cup \mathit{Dependencies} \cup \mathit{Polymorphism}, \\
\lambda\mathrm{P}\omega &= \lambda{\to} \cup \mathit{Dependencies} \cup \mathit{Higher-Order} \cup \mathit{Polymorphism}.
\end{aligned}$$

For each set of rules S, we write $\Gamma \vdash_S A : B$ to indicate that $\Gamma \vdash_t A : B$ can be derived using only the rules in S. The expression 'system S' refers to the typed system obtained by restricting the full system to allow only the rules in S. Then the eight typed systems can be arranged as vertices of the following cube (*Barendregt's cube*):



We list a few of the properties of this cube, being those that are explicitly used in this paper.

Property 1.6 $A[B/b][C/c] \equiv A[C/c][B[C/c]/b]$, *provided* $b \notin \mathrm{FV}(C)$. ■

Property 1.7 (Church-Rosser for TS) $A =_\beta B \ \& \ \Gamma \vdash_t A : C \ \& \ \Gamma \vdash_t B : C \ \Rightarrow \ \exists D \ [A \twoheadrightarrow_\beta D \ \& \ B \twoheadrightarrow_\beta D]$. ■

Property 1.8 *Barendregt's general typed system derives judgements of the following shapes:*

$$\Gamma \vdash_t M : \phi, \quad \Gamma \vdash_t \phi : K, \quad \textit{or} \quad \Gamma \vdash_t K : \Box.$$ ■

Property 1.9 $\mathcal{D} :: \Gamma, c{:}C \vdash_t A : B \ \Rightarrow \exists \mathcal{D}' \subseteq \mathcal{D} \ [\mathcal{D}' :: \Gamma \vdash_t C : s]$. ■

Property 1.10 (Typed Generation Lemma) *i)* $\Gamma \vdash_t a : A \ \Rightarrow \ \exists s, B \ [\Gamma \vdash_t B : s \ \& \ a{:}B \in \Gamma \ \& \ A =_\beta B]$.
ii) $\Gamma \vdash_t \lambda a{:}A.B : C \ \Rightarrow \ \exists s, D \ [\Gamma \vdash_t \Pi a{:}A.D : s \ \& \ \Gamma, a{:}A \vdash_t B : D \ \& \ C =_\beta \Pi a{:}A.D]$.
iii) $\Gamma \vdash_t AB : C \ \Rightarrow \ \exists D, E \ [\Gamma \vdash_t A : \Pi d{:}D.E \ \& \ \Gamma \vdash_t B : D \ \& \ C =_\beta E[B/d]]$.
iv) $\Gamma \vdash_t \Pi a{:}A.B : C \ \Rightarrow \ \exists s_1, s_2, s_3 \ [\Gamma \vdash_t A : s_1 \ \& \ \Gamma, a{:}A \vdash_t B : s_2 \ \& \ C =_\beta s_3]$. ■

Property 1.11 $\Gamma \vdash_t A : B \ \Rightarrow \ B \equiv \Box \vee \Gamma_t \vdash_t B : s$. ■

Property 1.12 (Termination for typed terms) *If* $\Gamma \vdash_t A : B$, *then* A *and* B *are both strongly normalizing.* ■

1.2 The Cube of Type Assignment Systems

In this subsection, we will present the cube of type assignment system as was first introduced in [13]. The definition of the type assignment cube is based on the definition of the type-erasing function E. In fact, both the syntax of terms, and the rules of the type assignment systems in the cube are obtained directly from the corresponding syntax and rules of the typed systems in Barendregt's cube, by applying E.

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.

As we already mentioned in the introduction, for the plane of the TScube without dependencies, there exists a function that, erasing type information from typed λ-terms, allows to switch from a typed system to a corresponding type assignment system. To be precise, it erases type information from λ-bindings occurring in λ-terms, while leaving all type information that decorates bindings in constructors and kinds intact. In [13], a more general function E was defined, by extending the domain of the above function to terms with dependencies in a natural way, as shown in the next definition.

Definition 1.13 *i)* $\{*, \Box\}$ is the set of *sorts*.
ii) The sets of λ-*terms* (Λ), *constructors* (*Cons*), and *kinds* (*Kind*) are mutually defined by the following grammar, where M, ϕ and K, are metavariables for λ-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 *terms* is the union of the sets Λ, *Cons* and *Kind*.

iii) The *erasing function* $E : T_t \to T_u$ is inductively defined as follows:

a) On Λ_t.

$$\begin{aligned} E(x) &= x, \\ E(MN) &= E(M)E(N), \\ E(M\phi) &= E(M), \\ E(\lambda x{:}\phi.M) &= \lambda x.E(M), \\ E(\lambda\alpha{:}K.M) &= E(M). \end{aligned}$$

b) On $Cons_t$.

$$\begin{aligned} E(\alpha) &= \alpha, \\ E(\Pi x{:}\phi.\psi) &= \Pi x{:}E(\phi).E(\psi), \\ E(\Pi\alpha{:}K.\psi) &= \Pi\alpha{:}E(K).E(\psi), \\ E(\lambda x{:}\phi.\psi) &= \lambda x{:}E(\phi).E(\psi), \\ E(\lambda\alpha{:}K.\psi) &= \lambda\alpha{:}E(K).E(\psi), \\ E(\phi\psi) &= E(\phi)E(\psi), \\ E(\phi M) &= E(\phi)E(M). \end{aligned}$$

c) On $Kind_t$.

$$\begin{aligned} E(*) &= *, \\ E(\Pi x{:}\phi.K) &= \Pi x{:}E(\phi).E(K), \\ E(\Pi\alpha{:}K_1.K_2) &= \Pi\alpha{:}E(K_1).E(K_2). \end{aligned}$$

The erasing function is extended to contexts in the obvious way and we use the notation $E(\Gamma)$. Note that the behaviour of E is such that, in the image of E, λ-terms are completely untyped, while constructors and kinds are 'partially' typed.

The notions of free variables, subterms and β-reduction, to be defined below, are similar to their 'fully typed' counterparts as can be found in [4,11], but slightly modified, according to the untyped term syntax.

Definition 1.14 $\mathrm{FV}(A)$, the set of *free variables of* A, and $\mathrm{ST}(A)$, the set of *subterms* of A, are inductively defined by:

$$\begin{aligned} \mathrm{FV}(*) &= \emptyset, & \mathrm{ST}(*) &= \{*\}, \\ \mathrm{FV}(a) &= \{a\}, & \mathrm{ST}(a) &= \{a\}, \\ \mathrm{FV}(BC) &= \mathrm{FV}(B) \cup \mathrm{FV}(C), & \mathrm{ST}(BC) &= \{BC\} \cup \mathrm{ST}(B) \cup \mathrm{ST}(C), \\ \mathrm{FV}(\Pi a{:}B.C) &= \mathrm{FV}(B) \cup (\mathrm{FV}(C) \setminus \{a\}), & \mathrm{ST}(\Pi a{:}B.C) &= \{\Pi a{:}B.C\} \cup \mathrm{ST}(B) \cup \mathrm{ST}(C), \\ \mathrm{FV}(\lambda a{:}B.C) &= \mathrm{FV}(B) \cup (\mathrm{FV}(C) \setminus \{a\}), & \mathrm{ST}(\lambda a{:}B.C) &= \{\lambda a{:}B.C\} \cup \mathrm{ST}(B) \cup \mathrm{ST}(C), \\ \mathrm{FV}(\lambda x.M) &= \mathrm{FV}(M) \setminus \{x\}, & \mathrm{ST}(\lambda x.M) &= \{\lambda x.M\} \cup \mathrm{ST}(M). \end{aligned}$$

Definition 1.15 The result of a simultaneous substitution $S = [A_1/a_1, \ldots, A_n/a_n]$, applied to a term D, is denoted either by $D[A_1/a_1, \ldots, A_n/a_n]$, or by D^S. 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. Formally, the substitution on terms is inductively defined by:

$$\begin{aligned} a_i^S &\equiv A_i, \text{ for } 1 \leq i \leq n, \\ b^S &\equiv b, \text{ for every } b \notin \{a_1, \ldots, a_n\}, \\ (BC)^S &\equiv B^S C^S, \\ (\Pi b{:}B.C)^S &\equiv \Pi b{:}B^S.C^S, \\ (\lambda b{:}B.C)^S &\equiv \lambda b{:}B^S.C^S, \\ (\lambda x.M)^S &\equiv \lambda x.M^S. \end{aligned}$$

The 'untyped variant' of Property *1.6* also holds:

Lemma 1.16 $A[B/b][C/c] \equiv A[C/c][B[C/c]/b]$, *provided* $b \notin \mathrm{FV}(C)$.

Proof: By easy induction on the definition of substitution. ■

Definition 1.17 β-reduction on terms can no longer be presented through a single generic rule as in Definition *1.2*. Instead, we have the following three rules:

$$\begin{aligned} (\lambda x{:}\phi.\psi)M &\to_\beta \psi[M/x], \\ (\lambda \alpha{:}K.\phi)\,\psi &\to_\beta \phi[\psi/\alpha], \\ (\lambda x.M)N &\to_\beta M[N/x]. \end{aligned}$$

The relations $\twoheadrightarrow_\beta$ and $=_\beta$ are defined as usual, starting from the reduction rules defined above.

Lemma 1.18 *If* $B =_\beta C$, *then* $B[D/a] =_\beta C[D/a]$.

Proof: By easy induction on the definition of $=_\beta$, using Lemma *1.16*. ■

Definition 1.19 *i)* A *statement* is an expression of one of the forms:

$$M : \phi, \quad \phi : K, \quad \text{or} \quad K : \Box,$$

where M is a λ-term, ϕ is a constructor and K is a kind. The left part of the statement is called the *subject*, the right part is called the *predicate*.

ii) A *declaration* is a statement whose subject is a variable.

Definition 1.20 *i)* A *context* is a sequence of declarations, whose subjects are distinct. The empty context is denoted by $<>$.

ii) Equality on contexts is defined by:

a) $<> = <>$;

b) $\Gamma, a{:}A = \Gamma', b{:}B$, if $\Gamma = \Gamma'$, $a \equiv b$, and $A \equiv B$.

iii) We write $a{:}A \in \Gamma$, if the declaration $a{:}A$ occurs in Γ.

iv) The *domain* of Γ, denoted by $\mathcal{D}om(\Gamma)$, is the set $\{a \mid \exists A\ [a{:}A \in \Gamma]\}$.

v) 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.

vi) $\mathrm{FV}(\Gamma) = \bigcup\{\mathrm{FV}(A) \mid \exists a\ [a{:}A \in \Gamma]\}$.

vii) We extend the notion of substitution to contexts by: $<>^S = <>$, and $(\Gamma, b{:}B)^S = \Gamma^S, b{:}B^S$.

Given the difference in syntax, the type assignment rules as presented in Definition *1.21* are only in appearance similar to those of Definition *1.3*. Note that the denotation of a rule is only different for the rules (I), (I_K) and (E_K). We will, therefore, take the liberty of using the same notation and names for rules; note, however, that the similarity is only superficial.

Definition 1.21 (General type assignment system) The following rules are used to derive *judgements* of the form $\Gamma \vdash A : B$, where Γ is a context and $A : B$ is a statement.

i) Common Rules

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

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

ii) λ-Term Rules

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

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

iii) Constructor Rules

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

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

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

iv) Kind Rules

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

$$(K{-}F_K) \quad \frac{\Gamma, \alpha{:}K_1 \vdash K_2 : \Box}{\Gamma \vdash \Pi\alpha{:}K_1.K_2 : \Box}$$

Notice that, unlike for the derivation rules of Definition *1.3*, the subject does not change in the type assignment rules (I_K) and (E_K). These two, together with the rules $(Weak)$ and $(Conv)$, are called the *not syntax-directed rules.*

The notion of derivation and subderivation for a judgement are the same as in Definition *1.4*, and an analogue of Property *1.8* also holds:

Lemma 1.22 The general type assignment system derives judgements of the following shapes:

$$\Gamma \vdash M : \phi, \quad \Gamma \vdash \phi : K, \quad \textit{or} \quad \Gamma \vdash K : \Box.$$

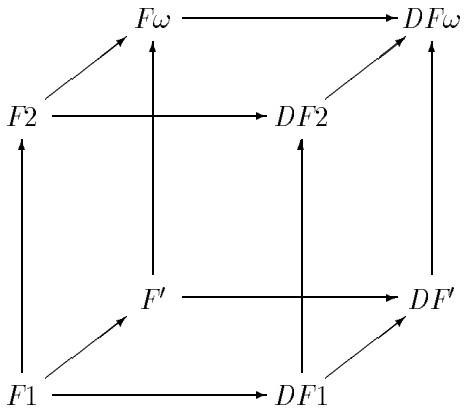
Proof: Easy, by looking at the rules and by observing that the sets *Cons* and *Kind* are closed for the substitution of λ-term-variables by λ-terms and constructor-variables by constructors. ■

As before, a *type* is a constructor of kind $*$ (and this is again 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 ϕ must be a type).

As in [13], we can distinguish eight different type assignment systems, defined using the same collection of rules given in Definition *1.5 (i)* for the TS-cube.

$$\begin{aligned}
F1 &= Base\text{-}Rules,\\
F' &= F1 \cup Higher\text{-}Order,\\
F2 &= F1 \cup Polymorphism,\\
F\omega &= F1 \cup Higher\text{-}Order \cup Polymorphism,\\
DF1 &= F1 \cup Dependencies,\\
DF' &= F1 \cup Dependencies \cup Higher\text{-}Order,\\
DF2 &= F1 \cup Dependencies \cup Polymorphism,\\
DF\omega &= F1 \cup Dependencies \cup Higher\text{-}Order \cup Polymorphism.
\end{aligned}$$

Like for TS we will use, for each set of rules S, the expression $\Gamma \vdash_S A : B$ to indicate that $\Gamma \vdash A : B$ can be derived using only the rules in S. These systems can be arranged as vertices of the following cube:



Notice that, in the left-hand plane of the cube, the constructors coincide with the typed ones, because there they cannot depend on λ-terms. This no longer holds in the right-hand plane: here we can build constructors like $(\lambda x{:}\phi.\psi)N$, where N is a pure, untyped λ-term.

The system $F1$ corresponds to the well-known Curry type assignment system, whereas $F2$ is the type assignment version of the second order λ-calculus. The three dimensions in this cube of type assignment systems correspond, as for Barendregt's cube, to the introduction of *polymorphic types*, *higher-order types* and *dependent types*.

2 Properties of the Cube of Type Assignment Systems

In this section, we will prove that all systems in the TAS-cube satisfy good computational properties, like subject reduction, the Church-Rosser property, and strong normalization of

typable terms. To prove these results, we need more definitions and technical lemmas, stating properties of the systems that are also of independent interest.

2.1 Basic properties

In this subsection, we will focus on some of the basic properties that hold for the cube of Type Assignment Systems. They are those that can be expected, and that also hold (in their typed variants) for Barendregt's cube; of course the results of section *3* show that those results cannot be used for the proofs needed here.

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

Lemma 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* $s_1 \equiv s_2$.

Proof: This is an obvious consequence of Lemma *1.22*. ■

Definition 2.2 A *legal context* is inductively defined as follows:

i) The empty context $<>$ is legal;

ii) If Γ is legal, and there exists s, such that $\Gamma \vdash A : s$, then, for every $a \notin \mathcal{D}om\,(\Gamma)$, also $\Gamma, a{:}A$ is legal.

Lemma 2.3 *If* $\Gamma \vdash A : B$, *then* Γ *is legal.*

Proof: By easy induction on the structure of derivations, using Definition *2.2*. ■

From now on, to avoid unnecessary complications in proofs and definitions, every context is assumed to be legal.

We define the following relations on contexts:

Definition 2.4 *i)* $\Gamma \sqsubseteq \Gamma' \iff \Gamma$ is a prefix of Γ'.

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

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 $\Gamma \stackrel{\cdot}{\sqsubseteq} \Gamma', a{:}A$.

For these relations, the following lemma holds.

Lemma 2.5 *i)* *If* $\Gamma_1 \sqsubseteq \Gamma_2$, *then* $\Gamma_1 \stackrel{\cdot}{\sqsubseteq} \Gamma_2$.

ii) *If* $\Gamma_1, a{:}A, \Gamma_2 \stackrel{\cdot}{\sqsubseteq} \Gamma'$, *then* $\Gamma' = \Gamma'_1, a{:}A, \Gamma'_2$, *with* $\Gamma_1 \stackrel{\cdot}{\sqsubseteq} \Gamma'_1$, *and* Γ'_2 *contains at least all statements of* Γ_2.

Proof: Easy. ■

Notice that, in part *(ii)*, in general the subcontexts Γ_2 and Γ'_2 are not legal contexts.

Lemma 2.6 *If* $\Gamma \stackrel{\cdot}{\sqsubseteq} \Gamma'$ *and* $\Gamma \vdash A : B$, *then*

i) (Free Variable Lemma) $\mathrm{FV}\,(A) \cup \mathrm{FV}\,(B) \subseteq \mathcal{D}om\,(\Gamma)$.

ii) (Thinning Lemma) $\Gamma' \vdash A : B$.

Proof: *i)* By induction on the structure of derivations. The only interesting cases are the

elimination rules; take for instance (E):

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

By induction, if $c \in \mathrm{FV}(M) \cup \mathrm{FV}(\Pi x{:}\phi.\psi)$, then $c \in \mathcal{D}om(\Gamma)$, and if $c \in \mathrm{FV}(N) \cup \mathrm{FV}(\phi)$, then $c \in \mathcal{D}om(\Gamma)$. Observe that $\mathrm{FV}(\psi[N/x]) = \mathrm{FV}(\psi)\backslash\{x\} \cup \mathrm{FV}(N)$.

ii) By induction on the structure of derivations. The only interesting cases are (*Proj*) and (*Weak*).

(*Proj*): Then $\Gamma = \Gamma_1, a{:}A$, for some Γ_1, such that $\Gamma_1 \vdash A : s$ and $\Gamma_1, a{:}A \boxdot \Gamma'$. Then, by Lemma *2.5 (ii)*, $\Gamma' = \Gamma'_1, a{:}A, \Gamma'_2$, with $\Gamma_1 \boxdot \Gamma'_1$, and, by induction, $\Gamma'_1 \vdash A : s$. To derive $\Gamma' \vdash a : A$, apply (*Proj*) once, and then (*Weak*) a suitable number of times.

(*Weak*): Then $\Gamma = \Gamma_1, c{:}C$. Since $\Gamma_1, c{:}C \boxdot \Gamma'$, by Lemma *2.5 (ii)*, we have that $\Gamma' = \Gamma'_1, c{:}C, \Gamma'_2$ and $\Gamma_1 \boxdot \Gamma'_1$. By induction, $\Gamma'_1 \vdash A : B$, and, by applying a series of (*Weak*), we derive $\Gamma' \vdash A : B$. ■

The following relation is introduced to abbreviate a sequence of derivation rules (I_K) and (E_K), that together correspond to polymorphism, and also takes the presence of rule (*Conv*) into account. It will be of use in Lemma *2.8* and in Theorem *2.18*.

Definition 2.7 We define the relation $\preceq$ on constructors inductively by:

$$\begin{array}{rcl} \phi =_\beta \psi & \Rightarrow & \phi \preceq \psi, \\ \phi & \preceq & \Pi\alpha{:}K.\phi, \\ \Pi\alpha{:}K.\phi & \preceq & \phi[\xi/\alpha], \\ \phi \preceq \psi \preceq \xi & \Rightarrow & \phi \preceq \xi. \end{array}$$

The four cases in Definition *2.7* reflect, respectively, an application of rule (*Conv*), (I_K), or (E_K), and a sequence of not syntax directed rules.

Worth noticing is the second case of Definition *2.7*, because it illustrates an important difference between the original presentation of the polymorphic type assignment system [17], and our presentation as a system in the topology of the TAS-cube. The equivalent rule for (I_K) in the polymorphic type assignment system is:

$$(\forall \mathrm{I}) \qquad \frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : \forall\alpha.\sigma}\,\alpha \notin \mathrm{FV}(\Gamma)$$

The type $\forall\alpha.\sigma$ is essentially the constructor $\Pi\alpha{:}*.\sigma$, and $\alpha \notin \mathrm{FV}(\Gamma)$ is a side-condition, indicating that binding of the type variable α is only allowed when α does not occur free in any predicate belonging to the context. The polymorphic type assignment system needs this side-condition to avoid to assign, for example,

$$x{:}\alpha \vdash x : \forall\alpha.\alpha.$$

The TAS presentation of this system does not require this extra condition on the derivation rule (I_K): in fact, types are generated by the system itself, using only legal contexts, which are essentially linear ordered sets of declarations in the derivations. In these systems, it is impossible to apply a (I_K) rule to the derivation for

$$\alpha{:}*, x{:}\alpha \vdash x : \alpha,$$

because $\alpha{:}*$ is not the right-most declaration in the context; when $\Gamma, \alpha{:}* \vdash M : \alpha$, then, by legality of the context, α does not occur in Γ, so especially does not occur free in Γ. We can say that the extra condition on (I_K) is *hidden* in the definition of legal context.

For TAS the following properties hold:

Lemma 2.8 (Generation Lemma for λ-terms) *i) If* $\Gamma \vdash x : \xi$, *then there is* ξ', *such that* $x{:}\xi' \in \Gamma$ *and* $\xi' \preceq \xi$.

ii) If $\mathcal{D} :: \Gamma \vdash MN : \xi$, *then there are* Γ', ϕ, ψ, *and* $\mathcal{D}' \subseteq \mathcal{D}$, *such that* $\psi[N/x] \preceq \xi$ *and*

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

iii) If $\mathcal{D} :: \Gamma \vdash \lambda x.M : \xi$, *then there are* Γ', ϕ, ψ, *and* $\mathcal{D}' \subseteq \mathcal{D}$, *such that* $\Pi x{:}\phi.\psi \preceq \xi$ *and*

$$\mathcal{D}' :: \frac{\Gamma', x{:}\phi \vdash M : \psi}{\Gamma' \vdash \lambda x.M : \Pi x{:}\phi.\psi}\ (I).$$

Proof: By induction on the structure of derivations, using Definition *2.7*. ■

Notice that this lemma states more than, for example, Property *1.10*, since it explicitly states the existence of a subderivation. This will be convenient in the proof of Theorem *2.18*. Also the following properties hold.

Lemma 2.9 (Generation Lemma for constructors and kinds) *i) If* $\Gamma \vdash \alpha : K$, *then there is* K', *such that* $\alpha{:}K' \in \Gamma$ *and* $K' =_\beta K$.

ii) If $\Gamma \vdash \lambda a{:}A.B : C$, *then there are* $\Gamma' \sqsubseteq \Gamma$, *and* D, *such that* $\Gamma', a{:}A \vdash B : D$ *and* $\Pi a{:}A.D =_\beta C$.

iii) If $\Gamma \vdash AB : C$, *then there are* $\Gamma' \sqsubseteq \Gamma$, D, *and* E, *such that* $\Gamma' \vdash A : \Pi d{:}D.E$, $\Gamma' \vdash B : D$ *and* $E[B/d] =_\beta C$.

iv) If $\Gamma \vdash \Pi a{:}A.B : s$, *then there is* $\Gamma' \sqsubseteq \Gamma$, *such that* $\Gamma', a{:}A \vdash B : s$.

Proof: By easy induction on the structure of derivations. ■

2.2 Typability

In this subsection, we will focus on a number of more evolved properties of the cube of Type Assignment Systems. First we prove that the notion of reduction as presented in Definition *1.17* satisfies the following property.

Property 2.10 (Church-Rosser for TAS) *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 [16], our β-reduction is a regular combinatory reduction system, and thus the Church-Rosser property follows from Theorem 3.11 in [16]. ■

The following lemma shows that all subterms of typable terms are typable.

Lemma 2.11 *Let* $B \in \mathrm{ST}(A)$. *If* $\Gamma \vdash A : C$, *then there exist* Γ', E, *such that* $\Gamma' \vdash B : E$.

Proof: By induction on the structure of derivations. The only interesting cases are the introduction rules (C–I_C) and (C–I_K); the others follow by easy induction. Take for instance (C–I_C):

$$\frac{\Gamma, x{:}\phi \vdash \psi : K}{\Gamma \vdash \lambda x{:}\phi.\psi : \Pi x{:}\phi.K}\ (C\text{–}I_C).$$

Recall that $\mathrm{ST}(\lambda x{:}\phi.\psi) = \{\lambda x{:}\phi.\psi\} \cup \mathrm{ST}(\phi) \cup \mathrm{ST}(\psi)$. The result follows directly for $B \equiv \lambda x{:}\phi.\psi$, by induction for $B \in \mathrm{ST}(\psi)$, and by induction and Lemma *2.3* for $B \in \mathrm{ST}(\phi)$. ■

The next lemma formulates that the class of derivable statements is closed for substitution on terms.

Lemma 2.12 (Substitution Lemma) *If* $\Gamma_1, c{:}C, \Gamma_2 \vdash A : B$, *and* $\Gamma_1 \vdash D : C$, *then* $\Gamma_1, \Gamma_2[D/c] \vdash A[D/c] : B[D/c]$.

Proof: By induction on the structure of derivations. The most interesting cases are when the last rule is (*Proj*), (*Weak*), or (*Conv*).

(*Proj*): If this rule is applied to the variable c, then we have

$$\frac{\Gamma_1 \vdash C : s \quad c \notin \mathcal{D}om(\Gamma_1)}{\Gamma_1, c{:}C \vdash c : C}\ (Proj),$$

and the result follows immediately from the assumption $\Gamma_1 \vdash D : C$.
Otherwise, (*Proj*) is applied to a variable different from c, i.e., $\Gamma_2 = \Gamma_2', b{:}B$ and

$$\frac{\Gamma_1, c{:}C, \Gamma_2' \vdash B : s \quad b \notin \mathcal{D}om(\Gamma_1, c{:}C, \Gamma_2)}{\Gamma_1, c{:}C, \Gamma_2', b{:}B \vdash b : B}\ (Proj).$$

Then, by induction, we obtain $\Gamma_1, \Gamma_2[D/c] \vdash B[D/c] : s$. Notice that, from the assumption $\Gamma_1 \vdash D : C$ and Lemma *2.6 (i)*, we know that $\mathrm{FV}(D) \subseteq \mathcal{D}om(\Gamma_1)$. Then, since $b \notin \mathcal{D}om(\Gamma_1, c{:}C, \Gamma_2)$, also $b \notin \mathcal{D}om(\Gamma_1, \Gamma_2[D/c])$. Then $\Gamma_1, \Gamma_2[D/c], b{:}B[D/c] \vdash b : B[D/c]$, as desired.

(*Weak*): Like for (*Proj*), we have to consider two subcases, of which the first is of the form

$$\frac{\Gamma_1 \vdash A : B \quad \Gamma_1 \vdash C : s \quad c \notin \mathcal{D}om(\Gamma_1)}{\Gamma_1, c{:}C \vdash A : B}\ (Weak).$$

The result follows directly from the assumption $\Gamma_1 \vdash A : B$ and the observation that, since $c \notin \mathcal{D}om(\Gamma_1) \supseteq \mathrm{FV}(A) \cup \mathrm{FV}(B)$, by Lemma *2.6 (i)*, also $A[D/c] \equiv A$ and $B[D/c] \equiv B$.
The second case follows by straightforward induction.

(*Conv*): By induction and Lemma *1.18*. ■

Now we are able to prove the property that all predicates in derivable statements are typable.

Lemma 2.13 *If* $\Gamma \vdash E : F$, *then either* $F \equiv \Box$, *or* $\Gamma \vdash F : s$.

Proof: By induction on the structure of derivations.

(*Axiom*), (*Conv*), (*Proj*): Immediate.

(*Weak*): By induction.

Elimination Rules: Then $F \equiv B[D/a]$, $\Gamma \vdash C : \Pi a{:}A.B$ and $\Gamma \vdash D : A$. By induction, either $\Pi a{:}A.B \equiv \Box$, or $\Gamma \vdash \Pi a{:}A.B : s$, for some s. The first is impossible; for the second, by the Generation Lemma (*2.9 (iv)*), there is a Γ', such that $\Gamma', a{:}A \vdash B : s$ and $\Gamma' \sqsubseteq \Gamma$. Then $\Gamma, a{:}A \vdash B : s$ follows from Lemmas *2.5 (i)* and *2.6 (ii)*. By applying the Substitution Lemma (*2.12*), we obtain $\Gamma \vdash B[D/a] : s$.

Introduction Rules: Then $F \equiv \Pi a{:}A.B$, and $\Gamma, a{:}A \vdash E : B$. By induction, either $B \equiv \Box$, or $\Gamma, a{:}A \vdash B : s$, for some s. The first case is impossible; for the second, apply a formation rule to obtain $\Gamma \vdash \Pi a{:}A.B : s$.

Formation Rules: Then $F \equiv s$, and $\Gamma, a{:}A \vdash B : s$. Clearly $F \equiv \Box$, or $F \equiv *$. The first case is immediate; in the second, observe that $\Gamma' \vdash * : \Box$ is derivable for all legal contexts. ■

Lemma 2.14 *If $\Gamma \vdash M : \phi$, then $\Gamma \vdash \phi : *$, i.e., ϕ is a type with respect to Γ.*

Proof: By Lemma *2.13*, either $\phi \equiv \Box$, or $\Gamma \vdash \phi : s$. But, by Lemma *1.22*, $\phi \not\equiv \Box$, since $M \in \Lambda$. We finish the proof by showing that $\Gamma \vdash \phi : s$ implies $s \equiv *$, by induction on the structure of derivations. The only important cases are the elimination rules, of which we show one example, and the (*Proj*) rule for constructors; the others follow by easy induction.

(C–E_C): Then $\Gamma \vdash \phi : K[N/x]$ and $K[N/x]$ is a sort. By the syntax of kinds we have that $K[N/x] \equiv *$.

(*Proj*): Then ϕ is a constructor variable, say α. Then $\Gamma = \Gamma', \alpha{:}s$ and, by legality of contexts, also $\Gamma' \vdash s : s'$. But this is possible only if $s \equiv *$. ■

2.3 Subject reduction

We now come to the proof that the here defined notion of type assignment is closed for subject reduction on typable λ-terms, i.e., if $\Gamma \vdash M : \psi$ and $M \twoheadrightarrow_\beta N$, then also $\Gamma \vdash N : \psi$. The proof of this result is not immediate, because of the presence of the derivations rules that are not syntax directed. It requires a sequence of lemmas; to start with, the next lemma states that contexts can be considered modulo β-conversion of predicates.

Lemma 2.15 *If $\Gamma_1, a{:}A, \Gamma_2 \vdash B : C$, then $\Gamma_1, a{:}A', \Gamma_2 \vdash B : C$, for all A' such that $\Gamma_1, a{:}A'$ is legal and $A' =_\beta A$.*

Proof: By induction on the structure of derivations. Most cases are dealt with by straightforward induction, except for:

(*Proj*): If $B \equiv a$, also $C \equiv A$ and $\Gamma_2 = <>$. Then $\Gamma_1 \vdash A : s$ and $a \notin \mathcal{D}om(\Gamma_1)$. The result is obtained from the following derivation (which exists, since $\Gamma_1, a{:}A'$ is legal):

$$\dfrac{\dfrac{\Gamma_1 \vdash A' : s \quad a \notin \mathcal{D}om(\Gamma_1)}{\Gamma_1, a{:}A' \vdash a : A'}\,(Proj) \qquad \dfrac{\Gamma_1 \vdash A : s \quad \Gamma_1 \vdash A' : s \quad a \notin \mathcal{D}om(\Gamma_1)}{\Gamma_1, a{:}A' \vdash A : s \qquad A =_\beta A'}\,(Weak)}{\Gamma_1, a{:}A' \vdash a : A.}\,(Conv)$$

If B is a variable different from a, the result follows by induction.

(*Weak*): If this rule is applied to the variable a, then $\Gamma_1 \vdash B : C$ and $a \notin \mathcal{D}om(\Gamma_1)$. Then we can also derive (again, by assumption, $\Gamma_1, a{:}A'$ is legal):

$$\frac{\Gamma_1 \vdash B : C \quad \Gamma_1 \vdash A' : s \quad a \notin \mathcal{D}om(\Gamma_1)}{\Gamma_1, a{:}A' \vdash B : C}\ (\textit{Weak}).$$

Again, if this rule is applied to a variable c different from a, the result follows by induction. ■

The following two lemmas together prove that if $\Pi x{:}\phi.\psi$ is derivable for $\lambda x.M$ from the context Γ, then ψ is derivable for M from the context $\Gamma, x{:}\phi$.

Lemma 2.16 *If* $\Gamma \vdash \lambda x.M : \xi$, *then there are* $K_1, \ldots, K_k$, ϕ *and* ψ, *such that* $\Pi_{i=1}^k \alpha_i{:}K_i.\Pi x{:}\phi.\psi =_\beta \xi$, *and* $\Gamma, \alpha_1{:}K_1, \ldots, \alpha_k{:}K_k, x{:}\phi \vdash M : \psi$.

Proof: By induction on the structure of derivations.

(I): Immediate, with $k = 0$.

(*Weak*): Let Γ be $\Gamma', c{:}C$, then $\Gamma' \vdash \lambda x.M : \xi$ and $\Gamma' \vdash C : s$. By induction,

$$\xi =_\beta \Pi_{i=1}^k \alpha_i{:}K_i.\Pi x{:}\phi.\psi \text{ and } \Gamma', \alpha_1{:}K_1, \ldots, \alpha_k{:}K_k, x{:}\phi \vdash M : \psi.$$

Since

$$\Gamma', \alpha_1{:}K_1, \ldots, \alpha_k{:}K_k, x{:}\phi \boxminus \Gamma', c{:}C, \alpha_1{:}K_1, \ldots, \alpha_k{:}K_k, x{:}\phi,$$

we can apply Lemma *2.6 (ii)* to obtain $\Gamma', c{:}C, \alpha_1{:}K_1, \ldots, \alpha_k{:}K_k, x{:}\phi \vdash M : \psi$.

(*Conv*), (I_K): Easy.

(E_K): Let $\xi \equiv \xi'[\mu/\alpha]$, then there exists K, such that $\Gamma \vdash \lambda x.M : \Pi\alpha{:}K.\xi'$ and $\Gamma \vdash \mu : K$. By induction, we have

$$\Pi\alpha{:}K.\xi' =_\beta \Pi_{i=1}^k \alpha_i{:}K_i.\Pi x{:}\phi.\psi \text{ and } \Gamma, \alpha_1{:}K_1, \alpha_2{:}K_2, \ldots, \alpha_k{:}K_k, x{:}\phi \vdash M : \psi.$$

By the Church-Rosser Property (*2.10*), β-convertible terms have a common reduct, so it must be that $k \geq 1$, $K =_\beta K_1$ (assuming by α-conversion that α is α_1) and $\xi' =_\beta \Pi_{i=2}^k \alpha_i{:}K_i.\Pi x{:}\phi.\psi$. By Lemma *1.18*, we have $\xi \equiv \xi'^S =_\beta \Pi_{i=2}^k \alpha_i{:}K_i^S.\Pi x{:}\phi^S.\psi^S$, where S stands for the substitution $[\mu/\alpha]$. By Lemma *2.15*, we have

$$\Gamma, \alpha{:}K, \alpha_2{:}K_2, \ldots, \alpha_k{:}K_k, x{:}\phi \vdash M : \psi,$$

and by the Substitution Lemma (*2.12*), we obtain

$$\Gamma, \alpha_2{:}K_2^S, \ldots, \alpha_k{:}K_k^S, x{:}\phi^S \vdash M : \psi^S.$$

The other cases are impossible. ■

Lemma 2.17 (Term Abstraction Lemma) *If* $\Gamma \vdash \lambda x.M : \Pi x{:}\phi.\psi$, *then* $\Gamma, x{:}\phi \vdash M : \psi$.

Proof: By Lemma *2.16*, we have $\Pi x{:}\phi.\psi =_\beta \Pi_{i=1}^k \alpha_i{:}K_i.\Pi x{:}\phi'.\psi'$, and since these two expressions have a common reduct, it must be that $k = 0$, $\phi =_\beta \phi'$ and $\psi =_\beta \psi'$. Also by Lemma *2.16*, we know that $\Gamma, x{:}\phi' \vdash M : \psi'$. Since $\Gamma \vdash \lambda x.M : \Pi x{:}\phi.\psi$, by Lemma *2.13*, also $\Gamma \vdash \Pi x{:}\phi.\psi : *$. By Lemma *2.9 (iv)*, there exists $\Gamma' \sqsubseteq \Gamma$, such that $\Gamma', x{:}\phi \vdash \psi : *$, so by the Thinning Lemma *2.6 (ii)*, also $\Gamma, x{:}\phi \vdash \psi : *$ and, by Lemma *2.3*, $\Gamma, x{:}\phi$ is legal. Since $\phi =_\beta \phi'$, by Lemma *2.15*, $\Gamma, x{:}\phi \vdash M : \psi'$. Moreover, since $\Gamma, x{:}\phi \vdash \psi : *$ and $\psi =_\beta \psi'$, we can apply rule (*Conv*) to this last derivation and obtain $\Gamma, x{:}\phi \vdash M : \psi$. ■

Using this last lemma, is becomes easy to prove that the notion of type assignment we consider in this paper is closed for subject reduction on terms.

Theorem 2.18 (Subject Reduction for Terms) *If* $\Gamma \vdash M : \psi$ *and* $M \twoheadrightarrow_\beta N$, *then* $\Gamma \vdash N : \psi$.

Proof: By induction on the number of β-reduction steps in $M \twoheadrightarrow_\beta N$. We just consider the base case, the inductive step is straightforward. The base case is proved by structural induction on the context in which the redex occurs: we only consider the case $M \equiv (\lambda x.P)Q$ and $N \equiv P[Q/x]$. Let $\mathcal{D}$ be a derivation for $\Gamma \vdash (\lambda x.P)Q : \psi$. By the Generation Lemma (*2.8 (ii)*), $\mathcal{D}$ has the following structure:

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

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

with $\psi'[Q/x] \preceq \psi$. That is, there is a subderivation $\mathcal{D}_1$, ending with an application of rule (E), that is followed by a (possibly empty) sequence of applications of the not syntax-directed rules (*Weak*), (*Conv*), (I_K) and (E_K). By the Term Abstraction Lemma (*2.17*), we obtain

$$\Gamma', x{:}\phi' \vdash P : \psi'.$$

Since $\Gamma' \vdash Q : \phi'$, by the Substitution Lemma (*2.12*), 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

$$\Gamma \vdash P[Q/x] : \psi,$$

as desired. ■

2.4 Normalization

An important property of type assignment systems is the strong normalization of typable terms; this is already know to hold for the systems $F\omega$, $F1$, $F2$ and F' [13]. Using this result, we will show that it also holds for the other four systems of the cube of type assignment systems.

For this, we use the function ED that 'erases dependencies', i.e., removes the λ-term information in dependent types, as defined in [13], that is based on a similar definition given in [19]. A similar function, erasing term-dependencies in the Theory of Generalized Functionality of [21], can also be found in [5].

Definition 2.19 The function $ED : T_u \to T_u$ is defined as follows:

i) On Λ.

$$ED\,(M) = M.$$

ii) On *Cons*.

$$\begin{aligned} ED\,(\alpha) &= \alpha, \\ ED\,(\Pi x{:}\phi.\psi) &= \Pi x{:}ED\,(\phi).ED\,(\psi), \\ ED\,(\Pi\alpha{:}K.\psi) &= \Pi\alpha{:}ED\,(K).ED\,(\psi), \\ ED\,(\lambda x{:}\phi.\psi) &= ED\,(\psi), \\ ED\,(\lambda\alpha{:}K.\psi) &= \lambda\alpha{:}ED\,(K).ED\,(\psi), \\ ED\,(\phi\psi) &= ED\,(\phi)ED\,(\psi), \\ ED\,(\phi M) &= ED\,(\phi). \end{aligned}$$

iii) On *Kind*.

$$\begin{aligned} ED\,(*) &= *, \\ ED\,(\Pi x{:}\phi.K) &= ED\,(K), \\ ED\,(\Pi\alpha{:}K_1.K_2) &= \Pi\alpha{:}ED\,(K_1).ED\,(K_2). \end{aligned}$$

Lemma 2.20 For ED, the following properties hold:

$$\begin{aligned} ED\,((\lambda x.M)N) &= (\lambda x.ED\,(M))(ED\,(N)), \\ ED\,(M[N/x]) &= ED\,(M)[ED\,(N)/x], \\ ED\,((\lambda\alpha{:}K.\phi)\,\psi) &= (\lambda\alpha{:}ED\,(K).ED\,(\phi))(ED\,(\psi)), \\ ED\,(\phi[\psi/\alpha]) &= ED\,(\phi)[ED\,(\psi)/\alpha], \\ ED\,((\lambda x{:}\phi.\psi)M) &= ED\,(\psi), \\ ED\,(\psi[M/x]) &= ED\,(\psi). \end{aligned}$$

Let $\rightarrow_\beta$ denote the one-step β-reduction rule. Then $A \rightarrow_\beta B$ implies either $ED\,(A) \rightarrow_\beta ED\,(B)$, or $ED\,(A) \equiv ED\,(B)$.

Proof: Easy. ■

Using this dependency-erasing function, we can relate the strong normalization problem for the full cube to that of the plane without dependencies, as done in the following theorem.

Theorem 2.21 (Termination for terms) *If $\Gamma \vdash A : B$, then A is strongly normalizing.*

Proof: In [13], 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 β-reductions. By Lemma *2.20*, 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 β-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 redex of the form $(\lambda x{:}\phi.\psi)M$. However, since M is a λ-term, such a reduction cannot create new abstractions of the form $\lambda x{:}\phi.\psi$. Therefore, the number of such abstractions must decrease after every step, and our reduction cannot be infinite. ■

Corollary 2.22 If $\Gamma \vdash A : B$, then:
i) B is strongly normalizing.
ii) all predicates in Γ are strongly normalizing.

Proof: Immediate, using Lemmas *2.13* and *2.3*. ■

3 The relation between the cubes of Typed and Type Assignment Systems

In this section we will focus on the relation between Barendregt's cube and the cube of Type Assignment Systems.

3.1 Consistency, similarity, and isomorphism between systems

In this subsection, we first introduce the notions of *consistency*, *similarity*, and *isomorphism* between typed and type assignment systems. Note that these notions depend on the choice of an erasing function $\mathcal{E}$.

Definition 3.1 Let S_t and S_u be, respectively, a typed and type assignment system, and let $\mathcal{E}$ be an erasing function from terms in S_t to terms in S_u.

i) We say that S_t and S_u are *consistent* (via $\mathcal{E}$) if $\mathcal{E}$ is sound with respect to provable judgements, i.e. $\Gamma_t \vdash_{S_t} A_t : B_t$ implies $\mathcal{E}(\Gamma_t) \vdash_{S_u} \mathcal{E}(A_t) : \mathcal{E}(B_t)$.

ii) Systems S_t and S_u are *similar* (via $\mathcal{E}$) if they are consistent and, moreover, $\mathcal{E}$ is complete with respect to provable judgements, i.e. $\Gamma \vdash_{S_u} A : B$ implies that there exists Γ_t, A_t and B_t, such that $\Gamma_t \vdash_{S_t} A_t : B_t$ and $\mathcal{E}(\Gamma_t) = \Gamma$, $\mathcal{E}(A_t) \equiv A$ and $\mathcal{E}(B_t) \equiv B$.

iii) Let $\mathcal{D}er_t$ and $\mathcal{D}er_u$ be the sets of all derivations in S_t and S_u. Systems S_t and S_u are *isomorphic* (via $\mathcal{E}$) if and only if there are $\mathcal{F}$: $\mathcal{D}er_t \to \mathcal{D}er_u$ and $\mathcal{G}$: $\mathcal{D}er_u \to \mathcal{D}er_t$, such that:

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

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

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

Notice that the definition of isomorphism expresses more than just soundness and completeness of $\mathcal{E}$. Notice, moreover, that $\mathcal{F}$ is not defined by induction on derivations, a detail that will be of importance in Section *4*. Finally, notice that, in the previous definition, S_u is not assumed to be obtained from S_t through the application of $\mathcal{E}$ to the rules of S_t.

The definition of isomorphism between two systems was already given in [13], but in a less general way. We have defined isomorphism with respect to an erasing function $\mathcal{E}$; the definition of isomorphism in [13] used a *fixed* function. To be more precise, two systems are isomorphic according to the definition in [13], if they are isomorphic in the sense of Definition *3.1* with respect to the function $\mathcal{F}$ that is defined as follows: $\mathcal{F}(\mathcal{D})$ is obtained from $\mathcal{D}$ by applying the erasing function E to all terms in $\mathcal{D}$; by abuse of notation, we denote $\mathcal{F}(\mathcal{D})$ by $E(\mathcal{D})$.

The following lemma states that, for the TS and TAS-cubes, the two notions of isomorphism coincide.

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

Proof: By easy induction. ∎

To show consistency of our systems we need the following lemma that shows that type erasure does not affect β-reduction.

Lemma 3.3 i) $E(A[B/b]) \equiv E(A)[E(B)/b]$.
ii) If $A \twoheadrightarrow_\beta B$, then $E(A) \twoheadrightarrow_\beta E(B)$.

Proof: i) By easy induction on the definition of substitution.
ii) By induction on the definition of $\twoheadrightarrow_\beta$, using part *(i)*. ∎

A similar result for β-conversion follows easily.

In [13], some results about the relation between TS and TAS have been proved. They are summarized in the following proposition.

Proposition 3.4 ([13]) *Let S_t and S_u be systems in corresponding vertices of the* TS *and* TAS-*cube, respectively.*

i) Systems S_t and S_u are consistent.

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

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

Proof: See [13]. The proof of parts *(i)* and *(ii)* uses Lemma *3.3*. ■

So, all typed systems S_t are consistents with respect to the corresponding untyped systems S_u. In addition, applying the erasing function E to all judgements used in a derivation in S_t yields a correct derivation in S_u. This implies that S_t and S_u are similar. Unfortunately, systems with dependencies need not be isomorphic, as we will show below.

Although a counterexample for proving Proposition *3.4 (iii)* can be found in [13], we will give here another, both for the convenience of the reader, and because it is an easier example than that in [13] (it does not make use of the (*Conv*) rule).

Example 3.5 Consider the following derivation in $DF1$ (for reasons of readability, we use the notation $A{\to}B$ for $\Pi a{:}A.B$, when a does not occur in B).

Let $\mathbf{O}$ stand for the λ-term $(\lambda xy.y)$ and let $\mathbf{I}$ denote the identity $(\lambda x.x)$. Let Γ be a context consisting of the following declarations:

$$\alpha{:}*, \beta{:}*, \gamma{:}*, a{:}(\gamma{\to}\gamma){\to}* .$$

Clearly, we can derive both $\Gamma \vdash \mathbf{I} : \alpha{\to}\alpha$ and $\Gamma \vdash \mathbf{O} : (\alpha{\to}\alpha){\to}\gamma{\to}\gamma$; combining these gives $\Gamma \vdash \mathbf{OI} : \gamma{\to}\gamma$, with which we can derive $\Gamma \vdash a(\mathbf{OI}) : *$; let $\mathcal{D}_i$ be the derivation for this result:

$$\mathcal{D}_1 :: \Gamma \vdash a(\mathbf{OI}) : *.$$

By applying rules (*Weak*) and (C–F_C), we get $\Gamma \vdash a(\mathbf{OI}){\to}\gamma : *$. Applying rule (*Proj*) gives $\Gamma, u{:}a(\mathbf{OI}){\to}\gamma \vdash u : a(\mathbf{OI}){\to}\gamma$, and by applying a (*Weak*) (using again derivation $\mathcal{D}_1$), we get

$$\mathcal{D}_2 :: \Gamma, u{:}a(\mathbf{OI}){\to}\gamma, v{:}a(\mathbf{OI}) \vdash u : a(\mathbf{OI}){\to}\gamma.$$

On the other hand, we can also derive $\Gamma \vdash \mathbf{I} : \beta{\to}\beta$ and $\Gamma \vdash \mathbf{O} : (\beta{\to}\beta){\to}\gamma{\to}\gamma$, which can be used, as above, to obtain $\Gamma \vdash a(\mathbf{OI}){\to}\gamma : *$, and by applying rules (*Weak*), (C–F_C) and (*Proj*), we obtain

$$\mathcal{D}_3 :: \Gamma, u{:}a(\mathbf{OI}){\to}\gamma, v{:}a(\mathbf{OI}) \vdash v : a(\mathbf{OI}).$$

Thus, using derivations $\mathcal{D}_2$ and $\mathcal{D}_3$, and applying rule (E), we may conclude with

$$\mathcal{D}_4 :: \Gamma, u{:}a(\mathbf{OI}){\to}\gamma, v{:}a(\mathbf{OI}) \vdash uv : \gamma.$$

The above described derivation, although legal in $DF1$, is not an erasure of any derivation in the fully typed system λP. To see this, note that we used two different types for different occurrences of the λ-term $\mathbf{I}$ (and thus also for different occurrences of $\mathbf{O}$) to obtain the two derivations $\mathcal{D}_2$ and $\mathcal{D}_3$. The expression $\mathbf{OI}$, however, is free of types, so the final typing for the application uv is correct. But if we want to obtain a correct derivation in λP of which $\mathcal{D}_4$ is the erasure, we have to assign *the same type* to the occurrences of $\mathbf{I}$ in the types of u and v.

More precisely, assume that $\mathcal{D}_4$ is obtained by erasure of a typed derivation $\mathcal{D}'_4$, such that

$$\mathcal{D}'_4 :: \Gamma_t, u{:}aM_t{\to}\gamma, v{:}aM_t \vdash_t uv : \gamma.$$

where M_t is a typed λ-term of type $\gamma{\to}\gamma$, such that $E\,(M_t) \equiv \mathbf{OI}$. The latter implies that

$$M_t \equiv (\lambda\alpha_1{:}K_1 \ldots \lambda\alpha_n{:}K_n.\mathbf{O}_t\mathbf{I}_t)\,\phi_1 \cdots \phi_m,$$

for some $n, m \geq 0$, where $E\,(\mathbf{O}_t) \equiv \mathbf{O}$ and $E\,(\mathbf{I}_t) \equiv \mathbf{I}$. But the fact that M_t must have type $\gamma{\to}\gamma$ implies that $n = m = 0$, and so $M_t \equiv \mathbf{O}_t\mathbf{I}_t$.

Since $\mathcal{D}_4$ is obtained from $\mathcal{D}'_4$ by erasure, there must be two subderivations of $\mathcal{D}'_4$ proving, respectively,

$$\Gamma_t \vdash_t \mathbf{O}_t : (\alpha{\to}\alpha){\to}\gamma{\to}\gamma, \text{ and } \Gamma_t \vdash_t \mathbf{O}_t : (\beta{\to}\beta){\to}\gamma{\to}\gamma,$$

such that $E\,(\mathbf{O}_t) \equiv \mathbf{O}$. But this is not possible, since this implies that

$$\mathbf{O}_t \equiv \lambda x{:}\alpha{\to}\alpha.\lambda y{:}\gamma.y, \text{ and } \mathbf{O}_t \equiv \lambda x{:}\beta{\to}\beta.\lambda y{:}\gamma.y,$$

at the same time, while α and β are different constructor variables.

Notice however, that we *can* derive $\Gamma_t, u : a(\mathbf{O}_t\mathbf{I}_t){\to}\gamma, v{:}a(\mathbf{O}_t\mathbf{I}_t) \vdash_t uv : \gamma$, because in constructing a derivation we are not forced to construct different types $\alpha{\to}\alpha$ and $\beta{\to}\beta$ for $\mathbf{I}_t$, but are free to choose $\beta \equiv \alpha$; therefore, this example is not a counterexample against similarity.

After the negative result of Proposition *3.4 (iii)*, it is natural to ask if the corresponding systems in the TS and TAS-cubes are at least similar, like was stated as conjecture in [13]. This property will be shown to hold in Theorem *3.12*, but only for those systems with dependencies that are without polymorphism, namely, for $DF1$ versus λP, and for DF' versus $\lambda\mathrm{P}\underline{\omega}$. Unfortunately, adding polymorphism makes a difference: the systems with both polymorphism and dependencies are not similar.

Theorem 3.6 Let S_t be either λP2, or $\lambda\mathrm{P}\omega$, and let S_u be, respectively, DF2 and DFω. 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 λP2.

Let Γ_0 be a context consisting of the following declarations:

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

Let $\mathbf{K}$ denote the λ-term $(\lambda xy.x)$, and let the untyped λ-terms M, M^0 and M^1 be defined by:

$$M \equiv u(\lambda f.x), \quad M^0 \equiv u(\lambda f.\mathbf{K}x(fy)), \quad \text{and} \quad M^1 \equiv u(\lambda f.\mathbf{K}x(fz)).$$

Clearly, both M^0 and M^1 β-reduce to M, and all these terms can correctly be assigned the type β in the context Γ_0. Thus, we can assert

$$\Gamma_0 \vdash \epsilon M^0{\to}\alpha : *, \quad \text{and} \quad \Gamma_0 \vdash \epsilon M^1 : *,$$

and this means that the context $\Gamma = \Gamma_0, p{:}\epsilon M^0{\to}\alpha, q{:}\epsilon M^1$ is legal. With help of the rules $(Proj)$, $(Conv)$ and (E), we can easily derive

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

We claim that the above judgement cannot be obtained as an erasure of any judgement $\Gamma_t \vdash_t N_t : \phi$ derivable in λP2, i.e., that we cannot have $E\,(\Gamma_t) = \Gamma$, $E\,(N_t) \equiv pq$ and $E\,(\phi) \equiv \alpha$.

To justify our claim, let us 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 result must also contain terms). Similarly, Γ_t may differ from Γ only in the declarations of p and q, which must be of the form:

$$p{:}\epsilon M_t^0 \to \alpha, \quad \text{and} \quad q{:}\epsilon M_t^1,$$

where $E\,(M_t^0) \equiv M^0$ and $E\,(M_t^1) \equiv M^1$.

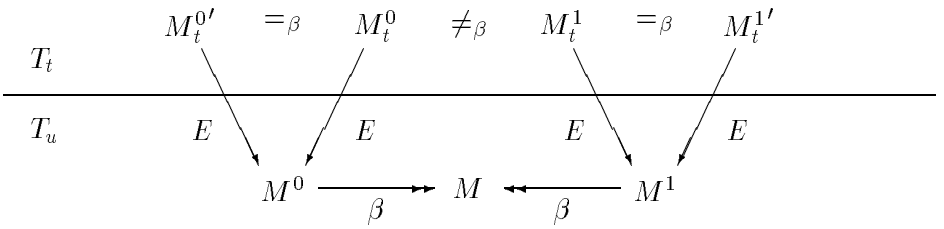
We can assume that N_t is of the form $P_t Q_t$, where $E\,(P_t) \equiv p$ and $E\,(Q_t) \equiv q$ (otherwise we consider an appropriate subterm of N_t instead). Since P_t is applied to Q_t, and the type of $P_t Q_t$ is α, it follows that P_t has a type of the form $\epsilon M_t^{0\prime} \to \alpha$, where $E\,(M_t^{0\prime}) \equiv M^0$. Similarly, Q_t has a type of the form $\epsilon M_t^{1\prime}$, where $E\,(M_t^{1\prime}) \equiv M^1$. In order to make the application well-typed (after a possible series of applications of rule $(Conv)$), it must be that $M_t^{0\prime} =_\beta M_t^{1\prime}$.

So we have β-convertible λ-terms $M_t^{0\prime}$ and $M_t^{1\prime}$, that erase to M^0 and M^1, respectively, and both are of type β. Without loss of generality, we can assume that these λ-terms have no β-redexes involving polymorphic abstraction and/or application, and thus we may write:

$$M_t^{0\prime} \equiv u\gamma(\lambda f{:}\gamma\to\gamma.\mathbf{K}_t x(fy)), \quad \text{and} \quad M_t^{1\prime} \equiv u\delta(\lambda f{:}\delta\to\delta.\mathbf{K}'_t x(fz)),$$

where $\mathbf{K}_t$ and $\mathbf{K}'_t$ both erase to $\mathbf{K}$. 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: if the polymorphic variable u is applied to a type μ, then f must be of type $\mu\to\mu$, and no constructor application $f\phi$ is possible. The β-normal forms of these terms are as follows: $M_t^{0\prime}$ reduces to $u\gamma(\lambda f{:}\gamma\to\gamma.x)$, while $M_t^{1\prime}$ reduces to $u\delta(\lambda f{:}\delta\to\delta.x)$. But these β-normal forms are different, and this contradicts the previous claim that $M_t^{0\prime} =_\beta M_t^{1\prime}$.

The following picture graphically summarizes this proof.



■

In the above system, we have shown the existence of two typed λ-terms, namely $M_t^{0\prime}$ and $M_t^{1\prime}$, and of a context Γ and type β, such that:

- $M_t^{0\prime} \neq_\beta M_t^{1\prime}$;
- $E\,(M_t^{0\prime}) =_\beta E\,(M_t^{1\prime})$;
- $\Gamma \vdash_t M_t^{0\prime} : \beta$ and $\Gamma \vdash_t M_t^{1\prime} : \beta$.

This allows the construction of the counterexample to the similarity.

The heart of the counterexample lies in both the polymorphic rules, and the fact that it is possible to abstract with respect to variables not occurring in the body. In fact, in the proof above, the polymorphic behaviour of the variable u makes that this term can be applied to both the terms $\lambda f{:}\gamma\to\gamma.\mathbf{K}_t x(fy)$ and $\lambda f{:}\delta\to\delta.\mathbf{K}_t x(fy)$. Also the use of the λ-term $\mathbf{K}$ is essential in order to obtain the correct final typing; because $\mathbf{K}$ is a cancelling term, the type assumed for the variable f has no effect on the type of the full terms $M_t^{0\prime}$ and $M_t^{1\prime}$.

It is natural to ask if this result allows some comparison between the power (with respect to typability and inhabitation) of the corresponding systems, respectively $DF2$ and λP2, $DF\omega$ and λPω. Recall that a (closed) type ϕ is *inhabited* in a system S, if and only if there is a

(closed) term M such that $<> \vdash M : \phi$.

The following corollary states that the set of types inhabited in S_u includes properly those types that are obtained through E from inhabited types in S_t, and states that also the set of types assignable to a term in S_u is larger than its corresponding set in S_t.

Corollary 3.7 Let S_t be either $\lambda P2$ or $\lambda P\omega$, and S_u be, respectively, DF2 or DFω.

i) $\{\phi \mid \exists M\ [<> \vdash_{S_u} M : \phi]\} \supsetneq \{\phi \mid \exists \phi_t, M_t\ [<> \vdash_{S_t} M_t : \phi_t\ \&\ E(\phi_t) = \phi]\}$.

ii) Let M be a closed term. Then $\{\phi \mid <> \vdash_{S_u} M : \phi\} \supsetneq$
$\{\phi \mid \exists \phi_t, M_t\ [<> \vdash_{S_t} M_t : \phi_t\ \&\ E(M_t) = M\ \&\ E(\phi_t) = \phi]\}$.

Proof: *i)* The inclusion follows immediately from the fact that S_t and S_u are consistent vie E. To prove that the inclusion is proper, take the derivation for $\Gamma \vdash pq : \alpha$ as constructed in the proof of Theorem *3.6*. Clearly, $<> \vdash \lambda xyzpq.pq : \phi$, where ϕ is the closure of α with respect to the context Γ. Then there is a derivation proving $\xi{:}\phi{\to}* \vdash \lambda xyzpq.pq : *$, and, therefore, $<> \vdash \Pi\xi{:}\phi{\to}*\lambda xyzpq.pq : *$. Let ψ be short for $\Pi\xi{:}\phi{\to}*\lambda xyzpq.pq$, then obviously $\psi{\to}\psi$ is a type inhabited in S_u by the term $\lambda x.x$, while it is not the erasure of an inhabited type in S_t.

ii) Also in this case, the inclusion follows from the consistency via E between S_u and S_t. To prove that the inclusion is proper, it is sufficient to observe that, for every closed term M typeable in S_u, there is a typing for M of the shape $\alpha{:}* \vdash M : \xi[\alpha]$. Then $<> \vdash M : \Pi\alpha{:}*.\xi[\alpha]$, and, by rule (E_K), $<> \vdash M : \xi[\psi]$, where ψ is defined as in part *(i)*. Clearly this type cannot be the erasure of an inhabited type in S_t. ■

3.2 Systems without polymorphism

In case polymorphism is not permitted, we can prove that the corresponding TS and TAS are similar. In what follows, the symbol $\vdash$ denotes $\vdash_S$, for $S \in \{F1, F', DF1, DF'\}$, while $\vdash_t$ refers to the corresponding TS. That is, we consider only systems without polymorphism. It is important to point out that, restricting the systems in this way, the derivation rules (I_K), (E_K) and $(C{-}F_K)$ are eliminated. Moreover, the syntax of terms is limited by no longer allowing for terms of the shape $M\phi$, $\lambda\alpha{:}K.\phi$ and $\Pi\alpha{:}K.\phi$.

Before we come to the main proof, we need some preliminary lemmas.

Lemma 3.8 i) If $\Gamma \vdash_t A : B$ and A is in β-normal form, then so is $E(A)$.

ii) If $E(A)$ is of the form $*$, variable, application, abstraction, or product, then so is A.

Proof: Easy. ■

The following lemma formulates that, in the absence of polymorphism, the erasing function E is injective on terms in normal form that can be assigned the same predicate.

Lemma 3.9 Suppose $\Gamma \vdash_t B_1 : A$ and $\Gamma \vdash_t B_2 : A$, and let both B_1 and B_2 be β-normal forms. If $E(B_1) \equiv E(B_2)$, then $B_1 \equiv B_2$.

Proof: By induction on the structure of terms.

Variable or sort constant: This case is immediate.

Abstractions: We only consider the case that B_1 is a λ-term, the other are essentially the same. Let $B_1 \equiv \lambda x{:}\phi_1.M_1$, with ϕ_1 and M_1 in β-normal form. By Property *1.10 (ii)*, we have $A =_\beta \Pi x{:}\phi_1.\psi_1$, for some ψ_1, such that $\Gamma, x{:}\phi_1 \vdash_t M_1 : \psi_1$. Since $E(B_2) \equiv E(B_1) \equiv$

$\lambda x.E\,(M_1)$, it must be that $B_2 \equiv \lambda x{:}\phi_2.M_2$, for some ϕ_2 and M_2 in β-normal form. Furthermore, $\Gamma \vdash_t \lambda x{:}\phi_2.M_2 : A$ and, by $(Conv)$, we have $\Gamma \vdash_t \lambda x{:}\phi_2.M_2 : \Pi x{:}\phi_1.\psi_1$. By Property *1.10 (ii)*, we have that $\Gamma, x{:}\phi_2 \vdash_t M_2 : \psi_2$, for some ψ_2, such that $\Pi x{:}\phi_1.\psi_1 =_\beta A =_\beta \Pi x{:}\phi_2.\psi_2$. By the Church-Rosser Property for TS (*1.7*), this implies $\phi_1 =_\beta \phi_2$ and $\psi_1 =_\beta \psi_2$. Since ϕ_1 and ϕ_2 are β-normal forms, they must be identical. Thus, we have in fact $\Gamma, x{:}\phi_1 \vdash_t M_2 : \psi_1$ (with help of $(Conv)$) and, by induction, we get $M_1 \equiv M_2$, so $B_1 \equiv B_2$.

Applications: If B_1 is an application in β-normal form, then $B_1 \equiv aC_1 \cdots C_n$, where a is a variable. Assume that $a{:}D \in \Gamma$. By Property *1.10 (iii)*, there are $F_1, \ldots, F_n$ and G, such that:

a) $D =_\beta \Pi_{i=1}^n b_i{:}F_i.G$;
b) $A =_\beta G[C_1/b_1]\cdots[C_n/b_n]$;
c) $\Gamma \vdash_t C_i : F_i[C_1/b_1]\cdots[C_{i-1}/b_{i-1}]$, for all $1 \le i \le n$.

Since $E\,(B_1) \equiv E\,(B_2)$, we have $B_2 \equiv aC_1' \cdots C_n'$, with $E\,(C_i) \equiv E\,(C_i')$. Repeating for B_2 the same argument as above, we get

d) $D =_\beta \Pi_{i=1}^n b_i{:}F_i'.G'$;
e) $A =_\beta G'[C_1'/b_1]\cdots[C_n'/b_n]$;
f) $\Gamma \vdash_t C_i' : F_i'[C_1'/b_1]\cdots[C_{i-1}'/b_{i-1}]$, for all $1 \le i \le n$.

We have $\Pi_{i=1}^n b_i{:}F_i.G =_\beta D =_\beta \Pi_{i=1}^n b_i{:}F_i'.G'$. Using the Church-Rosser Property for TS (*1.7*), we easily get $G =_\beta G'$ and $F_i =_\beta F_i'$, for all $1 \le i \le n$. It remains to show that $C_i \equiv C_i'$, for all i. Note that, by induction, and by the above properties *(c)* and *(f)*, the terms C_i and C_i' have the same type (the same kind) and are in β-normal form. Since C_i is a subterm of B_1, by induction, we obtain $C_i \equiv C_i'$.

Products: Let $B_1 \equiv \Pi c{:}C_1.D_1$. As $E\,(B_2) \equiv E\,(B_1)$, we have $B_2 \equiv \Pi c{:}C_2.D_2$. By Property *1.10 (iv)*, we have $\Gamma, c{:}C_1 \vdash_t D_1 : s$ and $\Gamma, c{:}C_2 \vdash_t D_2 : s$. Since the contexts are legal, we have also $\Gamma \vdash_t C_1 : s$ and $\Gamma \vdash_t C_2 : s$. By induction, we obtain $C_1 \equiv C_2$. Thus $\Gamma, c{:}C_1 \vdash_t D_2 : s$ and, once more by induction, we get $D_1 \equiv D_2$. ■

Using this result, in the following lemma we will prove that, in the absence of polymorphism, the erasing function E is injective on terms, modulo β-equality, that can be assigned the same predicate.

Lemma 3.10 *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: Let B_1' be the β-normal form of B_1 and let B_2' be the β-normal form of B_2 (by Theorem *1.12*, both terms are strong normalizable). Since B_1 reduces to B_1', we have $E\,(B_1) =_\beta E\,(B_1')$, by Lemma *3.3 (ii)*, and similarly for B_2. Thus, by Lemma *3.8 (i)*, we have $E\,(B_1') \equiv E\,(B_2')$ as they are β-normal forms. Finally, by Lemma *3.9*, we have $B_1' \equiv B_2'$, and thus $B_1 =_\beta B_2$. ■

Lemma 3.11 *Suppose that $\Gamma \vdash A : B$. Then the following conditions hold:*

i) There exists a typed legal context Γ_t, and typed terms A_t, B_t, satisfying $E\,(\Gamma_t) = \Gamma$, $E\,(A_t) \equiv A$ and $E\,(B_t) \equiv B$, such that $\Gamma_t \vdash_t A_t : B_t$.
ii) For every typed legal 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: By mutual induction on the structure of derivations.

$(Proj)$: Then $\Gamma = \Gamma', a{:}B$. Part *(i)* follows by induction. To show part *(ii)*, assume $E\,(\Gamma_t) = \Gamma', a{:}B$ and $E\,(B_t) \equiv B$ and $\Gamma_t \vdash_t B_t : s$. Note that we have $\Gamma_t = \Gamma_t', a{:}B_t'$, with $E\,(B_t') \equiv$

B. By Lemma *3.10*, $B'_t =_\beta B_t$. We want to prove that $\Gamma_t \vdash_t a : B_t$, which is accomplished as follows: since $\Gamma'_t, a{:}B'_t \vdash_t B_t : s$, by Property *1.9*, we know that $\Gamma'_t \vdash_t B'_t : s$. We apply (*Proj*) to obtain $\Gamma_t \vdash_t a : B'_t$ and, finally, (*Conv*) to conclude $\Gamma_t \vdash_t a : B_t$.

(*Weak*): Then $\Gamma = \Gamma', c{:}C$, and $\Gamma' \vdash A : B$ and $\Gamma' \vdash C : s$. To prove part *(i)*, by induction on part *(i)* to the first premise, we obtain $\Gamma'_t \vdash_t A_t : B_t$. Then use part *(ii)* to get $\Gamma'_t \vdash_t C_t : s$ (with the same context), and apply (*Weak*). For part *(ii)*, by induction we obtain A_t such that $\Gamma'_t, c{:}C_t \vdash_t A_t : B_t$. To be able to conclude the desired result by applying a (*Weak*), we also need $\Gamma'_t \vdash_t C_t : s$; this is obtained by Property *1.9* from the second premise.

(*Conv*): Then there is C, such that $\Gamma \vdash A : C$ and $\Gamma \vdash B : s$, with $B =_\beta C$. By induction, we derive $\Gamma_t \vdash_t A_t : C_t$ and $\Gamma_t \vdash_t B_t : s$. Since $E(B_t) \equiv B =_\beta C \equiv E(C_t)$, by Lemma *3.10*, we have $B_t =_\beta C_t$, and we can apply (*Conv*) to prove part *(i)*. Part *(ii)* is similar.

Introduction Rules: Part *(i)* follows immediately by induction; part *(ii)* is similar, but here we use Lemma *3.8 (ii)*.

Elimination Rules: Part *(i)* is easy (use Lemmas *3.8 (ii)* and *3.3 (i)*). For part *(ii)*, assume that from $\Gamma \vdash F : \Pi a{:}G.C$ and $\Gamma \vdash D : G$ we derived $\Gamma \vdash FD : C[D/a]$. Let Γ_t and H_t be such that $E(\Gamma_t) = \Gamma$ and $E(H_t) \equiv C[D/a]$. Then by induction $\Gamma_t \vdash_t F_t : \Pi a{:}G_t.C_t$ and $\Gamma_t \vdash_t D_t : G_t$, and we can derive $\Gamma_t \vdash_t F_t D_t : C_t[D_t/a]$, using the same elimination rule. By Lemma *3.3 (i)*, we have $E(C_t[D_t/a]) \equiv C[D/a] \equiv E(H_t)$.
Since $C_t[D_t/a]$ is a predicate of a derivable judgement, also $\Gamma_t \vdash_t C_t[D_t/a] : s$, by Property *1.11*. The term H_t is assumed to satisfy $\Gamma_t \vdash_t H_t : s$, and we may apply Lemma *3.10* to obtain $C_t[D_t/a] =_\beta H_t$. By applying (*Conv*), we derive $\Gamma_t \vdash_t F_t D_t : H_t$.

The remaining cases are easy. ■

With the last lemma, we can prove the main theorem of this subsection.

Theorem 3.12 *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 Theorem *3.4 (i)*, and Lemma *3.11 (i)*. ■

4 How to obtain an isomorphism

In this section, we will briefly discuss a way to define a cube of type assignment systems that is isomorphic to TS. As discussed above, the main problem that causes loss of isomorphism between TS and TAS, is that the erasure, through E, of two typed terms can be β-equivalent, while the originals were not (a thorough investigation on the possible alternative definitions of the (*Conv*) rule on typed systems can be found in [12]). We will show that it is possible to define another erasing function, named E', that gives rise to a second type assignment cube TAS$'$ which is isomorphic to the TS-cube (via E').

Remember that the behaviour of E was to erase type information from λ-terms. So, in case of dependencies, if A is a typed constructor, occurring in a typed kind, $E(A)$ can either coincide with A (in case A does not contain occurrences of λ-terms), or $E(A)$ can be partially typed. The new erasing function E' we will present below has a context-dependent behaviour, in the sense that it erases type information fromλ-terms, but *not* when these occur as subterms of constructors or kinds.

Definition 4.1 The new erasing function $E' : T_t \to T'_u$ is defined as follows:

$$E'(M) \;=\; E(M),$$

$$E'(\phi) = \phi,$$
$$E'(K) = K.$$

Now we will define a new type assignment cube TAS$'$. Note that, in contrast to the TAS-cube, this cube is not obtained by applying an erasing functin to all rules of TS. Instead, the new derivation rules are defined independently; however, the objects in the conclusion of each rule are in the codomain of E'.

Definition 4.2 (The TAS$'$ Cube) Let M range over Λ, ϕ range over typed constructors and K ranges over typed kinds; A, B, and C range over T'_u.

i) The untyped and typed λ-terms, typed constructors, and typed kinds are defined as before (Definitions *1.1* and *1.13*). Let T'_u be the union of the sets Λ, $Cons_t$ and $Kind_t$.

ii) The general type assignment system proves judgements of the following form:

$$\Gamma \vdash' M:\phi, \quad \Gamma \vdash' \phi:K, \quad \text{or} \quad \Gamma \vdash' K:\Box.$$

where $\phi \in Cons_t$, and $K \in Kind_t$.

iii) The type assignment rules are:

$$(Axiom)\ \frac{}{<> \vdash' * : \Box} \qquad (Conv)\ \frac{\Gamma \vdash' A:B \quad \Gamma \vdash' C:s \quad B =_\beta C}{\Gamma \vdash' A:C}$$

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

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

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

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

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

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

$$(K{-}F_C)\ \frac{\Gamma, x{:}\phi \vdash' K : \Box}{\Gamma \vdash' \Pi x{:}\phi.K : \Box} \qquad (K{-}F_K)\ \frac{\Gamma, \alpha{:}K_1 \vdash' K_2 : \Box}{\Gamma \vdash' \Pi\alpha{:}K_1.K_2 : \Box}$$

Notice that the derivation rules (E) and $(C{-}E_C)$ require derivations in TS, although restricted to typed λ-terms. This means that, officially, all rules of TS belong to the set of rules. Notice, moreover, that *only* types dependent on *typed* λ-terms are created in this way.

iv) As in Definition *1.5 (i)*, the rules can be grouped in sets. Again eight type assignment systems can be defined, whose relation can be represented as before by drawing a cube.

The main result on the relation between the TS-cube and the TAS$'$-cube is:

Theorem 4.3 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 (via E').*

Proof: i) 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:

(E): In this case, the typed derivation has the following shape:

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

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

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

$(C{-}E_C)$: The typed derivation has the following shape:

$$\mathcal{D}:: \frac{\mathcal{D}':: \Gamma \vdash_t \psi : \Pi x{:}\phi.K \quad \Gamma \vdash_t N : \phi}{\Gamma \vdash_t \psi N : K[N/x]}\ (C{-}E_C).$$

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

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

For all other rules, the definition of $\mathcal{F}$ is given by straightforward induction.

ii) The function $\mathcal{G}$: $\mathcal{D}er_u \to \mathcal{D}er_t$ can be defined by induction on the structure of $\mathcal{D} \in \mathcal{D}er_u$ in a similar way:

(E): Then the derivation has the following shape:

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

By induction, $\mathcal{G}(\mathcal{D}'):: \Gamma_t \vdash_t M_t : \xi$, where $E'(\Gamma_t) = \Gamma$, $E'(M_t) \equiv M$ and $E'(\xi) \equiv \Pi x{:}\phi.\psi$. This implies that $\Gamma_t = \Gamma$ and $\xi \equiv \Pi x{:}\phi.\psi$. So, we can define:

$$\mathcal{G}(\mathcal{D}):: \frac{\Gamma \vdash_t M_t : \Pi x{:}\phi.\psi \quad \Gamma \vdash_t N : \phi}{\Gamma \vdash_t M_t N : \psi[N/x]}\ (E).$$

$(C{-}E_C)$: In this case, the derivation has the following shape:

$$\mathcal{D}:: \frac{\mathcal{D}':: \Gamma \vdash' \psi : \Pi x{:}\phi.K \quad \Gamma \vdash_t N : \phi}{\Gamma \vdash' \psi N : K[N/x]}\ (C{-}E_C).$$

By induction, $\mathcal{G}(\mathcal{D}'):: \Gamma_t \vdash_t \theta : \xi$, where $E'(\Gamma_t) = \Gamma$, $E'(\theta) \equiv \psi$ and $E'(\xi) \equiv \Pi x{:}\phi.K$. This implies that $\Gamma_t = \Gamma$, $\theta \equiv \psi$ and $\xi \equiv \Pi x{:}\phi.\psi$. So, we can define:

$$\mathcal{G}(\mathcal{D})::\frac{\Gamma \vdash_t \psi : \Pi x{:}\phi.K \quad \Gamma \vdash_t N : \phi}{\Gamma \vdash_t \psi N : K[N/x]}\ (C\text{–}E_C).$$

For all other rules, the definition is straightforward.

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 (apparently) easy, the definition of the related cube is rather involved, since some rules require TS-derivations, and the erasing function occurs explicitly in the conclusion of rule (E). So, this cube does not satisfy the property of compositionality of derivations: not all subderivations of a derivation $\mathcal{D}$ are valid derivations in the system $\vdash'$. This is the price paid for defining a cube that is isomorphic to the typed one.

5 Related work

This paper, together with [13], can be seen as the first attempt to study type assignment systems with dependent types. In fact, all systems in the dependency-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 of Dowek [10], which is based on the typed system λP. Strictly speaking, this is not a type assignment system in the usual sense. In [10], there is no formal system to derive judgements; instead, a valid judgement of this system is defined as one of the form $\Gamma \vdash E(M) : B$, where $\Gamma \vdash_t M : B$ is a valid judgement of λP. The type checking problem for Dowek's system was shown to be undecidable in that paper. Dowek's system is equivalent to the system corresponding to λP in the TAS$'$-cube. We conjecture that this undecidability result is true for all our systems with dependencies. A further step of the work done in this paper could be made by looking for a type assignment counterpart to the Generalized Type Systems, as defined in [3,4,6].

References

[1] S. van Bakel, L. Liquori, S. Ronchi della Rocca, and P. Urzyczyn. Comparing Cubes. In A. Nerode and Yu.V. Matiyasevich, editors, *Proceedings of LFCS '94. Third International Symposium on Logical Foundations of Computer Science,* St. Petersburg, Russia, volume 813 of *Lecture Notes in Computer Science*, pages 353–365. Springer-Verlag, 1994.

[2] H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983.

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

[4] H.P. Barendregt. Lambda Calculi with Types. In S. Abramsky, Dov.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, chapter 1, pages 118–310. Clarendon Press, 1992.

[5] C.B. Ben-Yelles. g-Stratification is Equivalent to f-Stratification. *Zeitschr. f. Math. und Grundlagen d. Math*, Bd. 27:141–150, 1981.

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

[7] T. Coquand. Metamathematical Investigations of a Calculus of Constructions. In P. Odifreddi, editor, *Logic and Computer Science*, pages 91–122. Academic press, New York, 1991.

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

[9] H.B. Curry. Functionality in Combinatory Logic. In *Proc. Nat. Acad. Sci. U.S.A.*, volume 20, pages 584–590, 1934.

[10] G. Dowek. The Undecidability of Typability in the Lambda-Pi-Calculus. In M. Bezem and J.F. Groote, editors, *Proceedings of TLCA '93. International Conference on Typed Lambda Calculi and Applications,* Utrecht, the Netherlands, volume 664 of *Lecture Notes in Computer Science*, pages 139–145. Springer-Verlag, 1993.

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

[12] H. Geuvers and B. Werner. On the Church-Rosser Property for Expressive Type Systems and its Consequences for their Metatheoretic Study. In *Proceedings of the Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 320–329, 1994.

[13] P. Giannini, F. Honsell, and S. Ronchi della Rocca. Type inference: some results, some problems. *Fundamenta Informaticae*, 19(1,2):87–126, 1993.

[14] P. Giannini and S. Ronchi della Rocca. Characterization of Typings in Polymorphic Type Discipline. In *Proceedings of the Third Annual IEEE Symposium on Logic in Computer Science*, pages 61–70, 1988.

[15] J.Y. Girard. The System F of Variable Types, Fifteen years later. *Theoretical Computer Science*, 45:159–192, 1986.

[16] J.W. Klop. *Combinatory Reduction Systems.* PhD thesis, Department of Computer Science, Rijksuniversiteit Utrecht, 1980.

[17] D. Leivant. Polymorphic Type Inference. In *Proceedings 10$^{\text{th}}$ ACM Symposium on Principles of Programming Languages,* Austin Texas, pages 88–98, Austin Texas, 1983.

[18] R. Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17:348–375, 1978.

[19] Christine Paulin-Mohring. Extracting F_ω's programs from proofs in the Calculus of Constructions. In *Proceedings Sixteenth Annual ACM Symposium on Principles of Programming Languages*, Austin, Texas, 1989. Association for Computing Machinery.

[20] J.C. Reynolds. Towards a Theory of Type Structures. In B. Robinet, editor, *Proceedings of Programming Symposium,* Paris, France, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer-Verlag, 1974.

[21] J.P. Seldin. Progress Report on Generalised Functionality. *Annals of Math. Logic*, 17:29–59, 1979.