Complete restrictions of the intersection type discipline

Corrected version*

(Theoretical Computer Science, 102:135-163, 1992)

Steffen van Bakel †

Department of Informatics, Faculty of Mathematics and Informatics,
University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands.
steffen@cs.kun.nl

Abstract

In this paper the intersection type discipline as defined in [2] is studied. We will present two different and independent complete restrictions of the intersection type discipline.
The first restricted system, the strict type assignment system, is presented in section two. Its major feature is the absence of the derivation rule $(\leq)$ and it is based on a set of strict types. We will show that these together give rise to a strict filter lambda model that is essentially different from the one presented in [2]. We will show that the strict type assignment system is the nucleus of the full system, i.e. for every derivation in the intersection type discipline there is a derivation in which $(\leq)$ is used only at the very end. Finally we will prove that strict type assignment is complete for inference semantics.
The second restricted system is presented in section three. Its major feature is the absence of the type ω. We will show that this system gives rise to a filter λI-model and that type assignment without ω is complete for the λI-calculus. Finally we will prove that a lambda term is typeable in this system if and only if it is strongly normalisable.

Introduction

The popularity of functional programming has increased over the last decade. A large and still increasing number of people, computer scientists as well as manufacturers and logicians is becoming interested in functional programming languages.

A large number of functional programming languages already exist, many of them based on the lambda calculus. The calculus itself is type free, whereas it is common use to assign types to algorithms. Since the lambda calculus is a fundamental basis for functional programming languages, a type assignment system for the pure untyped lambda calculus, capable of deducing meaningful types, has been a topic of research for many years.

One of the first and most primitive ones was introduced by H.B. Curry in [8]. (See also [9]). His system expresses abstraction and application and has as its major advantage that it is decidable to determine whether a lambda term is typeable by this system. Because of this decidability it is used as a basis for type checkers used in functional programming languages. The functional programming language ML [18] for example, is in fact an extended lambda calculus and it contains a type checker based on Curry's system. Miranda, a functional programming language designed and implemented by D. Turner [25], contains a type checker based on the ML type assignment system.

* See appendix

† Research performed at the Department of Computer Science, Turin, Italy, and supported by the Netherlands Organisation for the advancement of pure research (N.W.O.).

Curry's type assignment system has however drawbacks. It is not capable of assigning a type to $\lambda x.xx$, and although the lambda terms $\lambda cd.d$ and $(\lambda xyz.xz(yz))(\lambda ab.a)$ are β-equal, the principal type schemes for these terms are different. Principal type schemes for Curry's system are defined by J.R. Hindley [13].

The intersection type discipline as presented in [2] does not contain these drawbacks. It is based on the Curry type assignment system: in addition to the type constructor '$\rightarrow$' it contains a type constructor '$\cap$' and a type constant 'ω'. These extensions were introduced to obtain a system that is closed under β-equality. The main problem of course is that of β-expansion: suppose we have derived $B \vdash M[N/x]:\sigma$[1] and also want to derive $B \vdash (\lambda x.M)N:\sigma$. This problem is solved by the introduction of the type constant ω and the intersection types. The type constant ω is the universal type, i.e. each term can be typed by ω. It can be used in the expansion to type N if N does not occur in $M[N/x]$ and there is no other type ρ such that $B \vdash N:\rho$. The intersection types are used for the cases that N occurs more than once in $M[N/x]$ and these occurrences were typed in the derivation for $B \vdash M[N/x]:\sigma$ with different types. A first introduction of a type assignment system with intersection types can be found in [3], a system with intersection types and ω is introduced in [6] and in [24].

In [2] the system as presented in [6] was strengthened further by introducing a partial order relation '$\leq$' on types as well as adding the type assignment rule $(\leq)$, and a more general form of the rules concerning intersection. The rule $(\leq)$ is introduced mainly to prove completeness of type assignment. This is achieved by showing that the set of types derivable for a lambda term in this extended system is a filter, i.e. a set closed under intersection and right closed for $\leq$ (if $\sigma \leq \tau$ and $\sigma \in d$ where d is a filter, then $\tau \in d$.) The interpretation of a lambda term by the set of types derivable for it gives a filter lambda model $\mathcal{F}$. Using this model, completeness is proved. Other interesting use of filter lambda models can be found in [4], [7], [11] and [12].

For the system as defined in [2], principal type schemes can be defined as in [23]. Instances of types can be obtained by substitution, operations of rise (applying $(\leq)$) or expansion (introducing intersection types by replacing a sub-derivation by more than one sub-derivation with the same structure, followed by an intersection introduction).

The intersection type discipline has a great expressive power: all solvable terms have types other than ω and a term has a normal form if and only if it has a type without ω occurrences. The system however is too powerful: it is closed under β-conversion. If a lambda term M is typeable by σ and $M =_\beta N$, then also N is typeable by σ. Because it is in general undecidable whether two terms are β-convertible, it is not possible to decide whether a lambda term can be typed by a type suitable for $\lambda x.x$. Moreover there are several ways to deduce a desired result, due to the presence of the derivation rules $(\cap I)$, $(\cap E)$ and $(\leq)$, which allow superfluous steps in derivations. In the system as presented in [6], these rules are not present and there is a one-one relationship between terms and derivations. In other words: the system is syntax directed.

The first restriction presented in this paper is the strict type assignment system, a type assignment system in which the $\leq$-relation and the derivation rule $(\leq)$ are no longer present. The elimination of $\leq$ induces a set of strict types, a restriction of the set of types used in the intersection type assignment system.

Strict types are the types that are strictly needed to assign a type to terms. The strict type assignment system is constructed from the set of strict types and a minor extension of the derivation rules as defined in [6]. In this way we obtain a syntax directed system. It turns out to be the nucleus of the intersection type assignment system. The strict system gives rise to a

[1] Unlike in [2], we will use the notation '$M:\sigma$' for the statement 'σ is a type for M'.

strict filter lambda model $\mathcal{F}_S$ that satisfies all major properties of the filter lambda model $\mathcal{F}$ as presented in [2], but is an essentially different lambda model.

In constructing a complete system, the semantics of types play a crucial role. As in [12], [19] and essentially following [14], a distinction can be made between several notions of type interpretations and semantic satisfiability. There are roughly three notions of type semantics that differ in the meaning of an arrow type scheme: inference type interpretations, simple type interpretations and F type interpretations. These different notions of type interpretations induce of course different notions of semantic satisfiability.

The intersection type assignment as presented in [2], is sound and complete with respect to the simple type semantics. In this paper we will show that soundness is lost if instead of simple type semantics, the inference type semantics is used. With the use of the latter we are able to prove soundness and completeness without having the necessity of introducing $\leq$. This will be done using the strict filter lambda model $\mathcal{F}_S$.

The second restriction presented is a type assignment system without ω. It is not difficult to see that, while building a derivation $B \vdash M{:}\sigma$ (where ω does not occur in σ and B) for a lambda term M that has a normal form, the type ω is only needed to type sub-terms that will be erased while reducing M to its normal form and that cannot be typed starting from B. This gives rise to the idea that if we limit ourselves to the set of lambda terms where no sub-terms will be erased, i.e. the λI-calculus, the type ω is not really needed for terms that have a normal form. The type assignment system without ω yields a λI-model and turns out to be complete for the λI-calculus with respect to the simple type semantics. The set of terms typeable by this system is just the set of all strongly normalisable lambda terms.

Because of its undecidability properties the intersection type discipline is at the present time not used in type checkers. In order to obtain a type checker based on this system, some restrictions have to be made. In this paper two restrictions of the intersection type discipline are studied, which both yield undecidable systems. So these attempts to restrict the system in preparation for the construction of a type checker, fail.

1 The intersection type discipline

The intersection type assignment system is an extension of the Curry type assignment system. It introduces intersection types and a type constant ω. Originally the system was called the 'extended type assignment system', but since a lot of different extensions of the Curry system exist, we prefer to use the name that highlights its major feature: the intersection types. In this section we give the definition of the intersection type discipline as presented in [2], together with its major features.

Definition 1.1 *i*) $\mathcal{T}$, the set of types is inductively defined by:

- *a*) All type variables $\varphi_0, \varphi_1, \ldots \in \mathcal{T}$.
- *b*) $\omega \in \mathcal{T}$.
- *c*) If σ and $\tau \in \mathcal{T}$, then $(\sigma \rightarrow \tau)$ and $(\sigma \cap \tau) \in \mathcal{T}$.

ii) On $\mathcal{T}$ the type inclusion relation $\leq$ is inductively defined by:

- *a*) $\sigma \leq \sigma$.
- *b*) $\sigma \leq \omega$.
- *c*) $\omega \leq \omega \rightarrow \omega$.
- *d*) $\sigma \cap \tau \leq \sigma$.
- *e*) $\sigma \cap \tau \leq \tau$.

f) $(\sigma\to\tau)\cap(\sigma\to\rho) \leq \sigma\to(\tau\cap\rho)$.

g) $\sigma \leq \tau \leq \rho \Rightarrow \sigma \leq \rho$.

h) $\sigma \leq \tau \;\&\; \sigma \leq \rho \Rightarrow \sigma \leq \tau\cap\rho$.

i) $\rho \leq \sigma \;\&\; \tau \leq \mu \Rightarrow \sigma\to\tau \leq \rho\to\mu$.

iii) $\sigma \sim \tau \iff \sigma \leq \tau \leq \sigma$.

iv) A statement is an expression of the form $M:\sigma$ where $M \in \Lambda$ and $\sigma \in \mathcal{T}$. M is the subject and σ the predicate of $M:\sigma$.

v) A basis is a set of statements with only variables (not necessarily distinct) as subjects.

$\mathcal{T}$ may be considered modulo $\sim$. Then $\leq$ becomes a partial order.

Notice that in the original paper [2] the type inclusion relation is defined in a slightly different way. Instead of rule 1.1 (*ii.h*) the rules

h') $\sigma \leq \tau \;\&\; \mu \leq \rho \Rightarrow \sigma\cap\mu \leq \tau\cap\rho$.

h'') $\sigma \leq \sigma\cap\sigma$.

are given. It is not difficult to show that these definitions are equivalent.

Throughout this paper, the symbol φ will be a type variable and the symbols μ, ν, η, ρ, σ, and τ will range over types.

Definition 1.2 *i*) Intersection type assignment is defined by the following natural deduction system.

$$(\to I):\; \frac{\begin{array}{c}[x{:}\sigma]\\ \vdots\\ M:\tau\end{array}}{\lambda x.M:\sigma\to\tau}\,(a) \qquad (\to E):\; \frac{M:\sigma\to\tau \qquad N:\sigma}{MN:\tau}$$

$$(\cap I):\; \frac{M:\sigma \qquad M:\tau}{M:\sigma\cap\tau} \qquad (\cap E):\; \frac{M:\sigma\cap\tau}{M:\sigma} \qquad \frac{M:\sigma\cap\tau}{M:\tau}$$

$$(\leq):\; \frac{M:\sigma}{M:\tau}\,(\sigma \leq \tau) \qquad (\omega):\; \frac{}{M:\omega}$$

(a) If $x{:}\sigma$ is the only statement about x on which $M:\tau$ depends.

ii) If $M:\sigma$ is derivable from a basis B, we write $B \vdash M{:}\sigma$.

In [2] several properties of this type assignment system are proved. Some of the more important are:

- The set of types derivable for a lambda term is a filter, i.e. a set closed under intersection and right closed for $\leq$.
- The interpretation of a lambda term by the set of types derivable for it, gives a filter lambda model $\mathcal{F}$. Using this model, completeness is proved.
- The set of normalisable terms can be characterised in the following way:

$$\exists B,\ \sigma\ [B \vdash M{:}\sigma \;\&\; B{,}\sigma\ \omega\text{-free}] \iff M \text{ has a normal form.}$$

- The set of terms having a head normal form can be characterised in the following way:

$$\exists B,\ \sigma\ [B \vdash M{:}\sigma \;\&\; \sigma \neq \omega] \iff M \text{ has a head normal form.}$$

Definition 1.3 The following properties are used in this paper and are listed here to be able to refer to them easily:

i) [2].2.8(i): $B \vdash MN{:}\tau \Rightarrow \exists \sigma \in \mathcal{T}\ [B \vdash M{:}\sigma\to\tau \;\&\; B \vdash N{:}\sigma]$.

ii) [2].2.8(iii): $B \vdash \lambda x.M : \sigma \to \tau \iff B \backslash x \cup \{x:\sigma\} \vdash M : \tau$.[2]

iii) [2].4.13(i): $\exists B,\sigma\ [B \vdash M : \sigma\ \&\ \sigma \neq \omega] \Rightarrow M$ has a head normal form.

iv) [2].4.13(ii): $\exists B,\sigma\ [B \vdash M : \sigma\ \&\ B,\sigma\ \omega$-free$] \iff M$ has a normal form.

v) [2].2.7(ii): $B \vdash x : \tau \Rightarrow \exists x{:}\tau_1, \ldots, x{:}\tau_n \in B\ [\tau_1 \cap \cdots \cap \tau_n \leq \tau]$.

vi) [12].5.6: $\rho \leq (\tau_1 \cap \cdots \cap \tau_n) \to \sigma \Rightarrow \rho = (\tau_1^1 \to \cdots \to \tau_n^s \to \sigma_1) \cap \cdots \cap (\tau_1^s \to \cdots \to \tau_n^s \to \sigma_s) \cap \rho'$, for some $\tau_1^j, \ldots, \tau_n^j, \sigma_j, \rho'$ such that $\tau_i^j \geq \tau_i$ with $i \leq i \leq n$, $1 \leq j \leq s$ and $\sigma_1 \cap \cdots \cap \sigma_s \leq \sigma$.

2 The system without derivation rule $(\leq)$

In this section we will give an extension (without the $(\leq)$-rule) of the Curry type assignment system, which in fact will be a combination of both the systems as presented in [3] and [6], and is almost the same as the one presented in [5]. We will prove that this system also yields a filter lambda model (subsection 2.1) and that type assignment in this system is complete (subsection 2.3). To achieve the completeness result we will have to use inference semantics as defined in [19] as a notion of type interpretation, instead of the simple semantics as used in [2]. Furthermore we will show that if in a derivation for $M : \sigma$ the derivation rule $(\leq)$ is used, the same statement can be derived using a derivation in which the derivation rule $(\leq)$ is at the most only used at the very end of the derivation (subsection 2.2).

2.1 Strict derivations

In this subsection we present a restricted version of the intersection type assignment system, in which the derivation rule $(\leq)$ is no longer present, together with a restricted set of types. These together will yield a lambda model, with which we prove completeness of type assignment without the derivation rule $(\leq)$.

Strict types and strict derivations are closely related. Strict derivations are syntax directed and yield strict types. The type constant ω plays a limited role in the strict type assignment system. It does not occur in an intersection subtype and occurs only on the left-hand side of an arrow type scheme. Moreover, intersection type schemes occur in strict types only as subtypes at the left hand side of an arrow type scheme.

Definition 2.1 *i*) $\mathcal{T}_s$, the set of strict types, is inductively defined by:

a) All type variables $\varphi_0, \varphi_1, \ldots \in \mathcal{T}_s$.

b) If σ, $\sigma_1, \ldots, \sigma_n$, $\tau \in \mathcal{T}_s$, then $\sigma \to \tau$, $\omega \to \tau$, $(\sigma_1 \cap \cdots \cap \sigma_n) \to \tau \in \mathcal{T}_s$.

ii) $\mathcal{T}_S$ is defined as the union of $\{\omega\}$ and the closure of $\mathcal{T}_s$ under intersection.

iii) On $\mathcal{T}_S$, the relation $\leq_s$ is defined by:

a) $\sigma \leq_s \sigma$.

b) $\sigma \leq_s \omega$.

c) $\sigma \cap \tau \leq_s \sigma\ \&\ \sigma \cap \tau \leq_s \tau$.

d) $\sigma \leq_s \tau \leq_s \rho \Rightarrow \sigma \leq_s \rho$.

e) $\sigma \leq_s \rho\ \&\ \sigma \leq_s \tau \Rightarrow \sigma \leq_s \rho \cap \tau$.

iv) $\sigma \sim_s \tau \iff \sigma \leq_s \tau \leq_s \sigma$.

v) A statement is an expression of the form $M : \sigma$ where $\sigma \in \mathcal{T}_S$ and $M \in \Lambda$. M is the subject and σ the predicate of $M : \sigma$.

vi) A basis is a set of statements with only variables as subjects.

[2] $B \backslash x$ is the basis obtained from B by erasing the statements that have x as subject.

$\mathcal{T}_S$ may be considered modulo $\sim_s$. Then $\leq_s$ becomes a partial order.

It is an easy exercise to show that the definition of $\leq_s$ is equivalent to:

i) $\sigma \leq_s \omega$.

ii) If $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$ $(n \geq 1)$, $\tau = \tau_1 \cap \cdots \cap \tau_m$ $(m \geq 1)$ and $\{\sigma_1, \ldots, \sigma_n\} \subseteq \{\tau_1, \ldots, \tau_m\}$, then $\tau \leq_s \sigma$.

It is also easy to show that if $\sigma \leq_s \tau$, then either $\tau = \omega$ or $\tau = \sigma$ or σ is an intersection type scheme in which τ occurs. Notice moreover that if $\sigma \sim_s \tau$, then either $\sigma = \tau$ or σ is an intersection type scheme and τ can be obtained from σ by permuting its strict components. In fact the differences affect none of our proofs and in the rest of the paper $\sigma = \tau$ means $\sigma \sim_s \tau$.

Definition 2.2 *i*) Strict type assignment and strict derivations are defined by the following natural deduction system (where all types displayed are strict, except σ in rule $(\rightarrow I)$):

$$(\rightarrow I): \quad \frac{\begin{array}{c}[x{:}\sigma] \qquad (\sigma \in \mathcal{T}_S)\\ \vdots \\ M:\tau\end{array}}{\lambda x.M : \sigma \rightarrow \tau}\ (a) \qquad\qquad (\cap E): \quad \frac{x:\sigma \cap \tau}{x:\sigma}$$

$$(\rightarrow E): \quad \frac{M:(\sigma_1 \cap \cdots \cap \sigma_n) \rightarrow \tau \qquad N:\sigma_1 \quad \cdots \quad N:\sigma_n}{MN:\tau} \qquad \frac{M:\omega \rightarrow \tau}{MN:\tau}$$

(a): If $x{:}\sigma$ is the only statement about x on which $M:\tau$ depends.

(b): Notice that rule $(\rightarrow E)$ consists of two parts.

If $M:\sigma$ is derivable from B using a strict derivation, we write $B \vdash_s M{:}\sigma$.

ii) We define $\vdash_s$ by: $B \vdash_s M{:}\sigma$ if and only if: $\sigma = \omega$ or there are $\sigma_1, \ldots, \sigma_n$ $(n \geq 1)$ such that $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$ and for every $i \in \{1, \ldots, n\}$ $B \vdash_s M{:}\sigma_i$.

Notice that in $B \vdash_s M{:}\sigma$ the basis can contain types that are not strict, and that $B \vdash_s M{:}\sigma$ is only defined for $\sigma \in \mathcal{T}_S$.

Notice also that the derivation rule $(\cap E)$ is only performed on variables and that the derivation rules (ω) and $(\cap I)$ are implicitly present in the derivation rule $(\rightarrow E)$. Moreover, we cannot compose a derivation in the $\vdash_s$ system with conclusion $M:\omega$ with any other derivation.

The introduction of two different notions of derivability seems somewhat superfluous. Notice that we could limit ourselves to one, by stating:

> We define $\vdash_s$ by: $B \vdash_s M{:}\sigma$ if and only if: $\sigma = \omega$ or there are $\sigma_1, \ldots, \sigma_n$ $(n \geq 1)$ such that $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$ and for every $i \in \{1, \ldots, n\}$ $M:\sigma_i$ is derivable from B using a strict derivation.

This definition would cause a lot of words in the proofs and perhaps also a lot of confusion. We therefore prefer two different notions of derivability.

Apart from the presence of ω, the type assignment defined by $\vdash_s$ is in fact the same as the one presented in [3]. Also, the one defined by $\vdash_s$ is in fact the same as in [6]. The type assignment defined by $\vdash_s$ is in fact the same as the one presented in [5], it is only different in a standard way of writing bases.

Lemma 2.3 For these notions of type assignment, the following properties hold:

i) If $\sigma \neq \omega$, *then* $B \vdash_s M{:}\sigma \iff \exists \sigma_1, \ldots, \sigma_n\ [B \vdash_s M{:}\sigma_i \ \&\ \sigma = \sigma_1 \cap \cdots \cap \sigma_n]$.

ii) $B \vdash_s MN{:}\sigma \iff \exists \tau\ [B \vdash_s M{:}\tau \rightarrow \sigma \ \&\ B \vdash_s N{:}\tau]$.

iii) $B \vdash_s M{:}\sigma \iff B \vdash_s M{:}\sigma \ \&\ \sigma \in \mathcal{T}_s$.

iv) $B \vdash_s \lambda x.M{:}\sigma \iff \exists \rho \in \mathcal{T}_S, \mu \in \mathcal{T}_s\ [\sigma = \rho \rightarrow \mu \ \&\ B \backslash x \cup \{x{:}\rho\} \vdash_s M{:}\mu]$.

v) $B \vdash_{\mathrm{s}} M{:}\sigma \iff \{x{:}\tau \in B \mid x \in fv(M)\} \vdash_{\mathrm{s}} M{:}\sigma$.

vi) $\forall \sigma,\tau \in \mathcal{T}_{\mathrm{S}}\ [(B \cup \{x{:}\sigma\} \vdash_{\mathrm{s}} M{:}\tau \Rightarrow B \cup \{x{:}\sigma\} \vdash_{\mathrm{s}} N{:}\tau)\ \&\ x \textit{ not in } B] \Rightarrow$
$\forall \rho \in \mathcal{T}_{\mathrm{S}}\ [B \vdash_{\mathrm{s}} \lambda x.M{:}\rho \Rightarrow B \vdash_{\mathrm{s}} \lambda x.N{:}\rho]$.

Proof: Easy. ■

As in [2] we aim to construct a filter lambda model. By use of names we will distinguish between the definition of filters in that paper and the ones given here.

Definition 2.4 *i*) A subset d of $\mathcal{T}_{\mathrm{S}}$ is called a strict filter if and only if:

a) $\omega \in d$.

b) $\sigma,\tau \in d \Rightarrow \sigma \cap \tau \in d$.

c) $\tau \in d\ \&\ \tau \leq_s \sigma \Rightarrow \sigma \in d$.

ii) If V is a subset of $\mathcal{T}_{\mathrm{S}}$, then $\uparrow_{\mathrm{s}} V$ is the smallest strict filter that contains V, and $\uparrow_{\mathrm{s}}\sigma = \uparrow_{\mathrm{s}}\{\sigma\}$. If no confusion is possible, we will omit the subscript on $\uparrow$.

iii) $\mathcal{F}_{\mathrm{S}} = \{d \subseteq \mathcal{T}_{\mathrm{S}} \mid d \text{ is a strict filter}\}$. We define application on $\mathcal{F}_{\mathrm{S}}$, $\cdot : \mathcal{F}_{\mathrm{S}} \times \mathcal{F}_{\mathrm{S}} \to \mathcal{F}_{\mathrm{S}}$ by: $d \cdot e = \uparrow\{\tau \mid \exists \sigma \in e\ [\sigma \to \tau \in d]\}$.

The application on filters as defined in [2] is not useful in our approach, since it would not be well defined. We must force the application to yield filters, since in each arrow type scheme $\sigma \to \tau \in \mathcal{T}_{\mathrm{S}}$, τ is strict.

$\langle \mathcal{F}_{\mathrm{S}}, \subseteq \rangle$ is a cpo and henceforward we will consider it with the corresponding Scott topology.

Because of the remark made after 2.1, condition 2.4 (*i.c*) can be replaced by:

2.4.*i.c'*) $\sigma \cap \tau \in d \Rightarrow \sigma \in d\ \&\ \tau \in d$.

Notice that a strict filter generated by a finite number of types is finite. Let for example σ be a strict type, then $\uparrow_{\mathrm{s}}\sigma = \{\sigma,\omega\}$ (where by $\sim_{\mathrm{s}}$ we identify σ and $\sigma\cap\sigma$). If σ is an intersection of strict types, $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$, then $\uparrow_{\mathrm{s}}\sigma$ contains 2^n elements, namely $\{\sigma_1,\ldots,\sigma_n,\sigma_1\cap\sigma_2,\sigma_1\cap\sigma_3,\ldots,\sigma_{n-1}\cap\sigma_n,\sigma_1\cap\sigma_2\cap\sigma_3,\ldots,\sigma_1\cap\cdots\cap\sigma_n,\omega\}$. Of course $\mathcal{F}_{\mathrm{S}}$ contains also infinite elements.

Lemma 2.5 For strict filters the following properties hold:

i) $\sigma \neq \omega\ \&\ \sigma \in \uparrow V\ \&\ V \subseteq \mathcal{T}_{\mathrm{s}} \iff \exists \sigma_1,\ldots,\sigma_n\ [\sigma = \sigma_1 \cap \cdots \cap \sigma_n\ \&\ \forall i \in \{1,\ldots,n\}\ [\sigma_i \in V]]$.

ii) $\sigma \in \mathcal{T}_{\mathrm{s}}\ \&\ \sigma \in \uparrow V\ \&\ V \subseteq \mathcal{T}_{\mathrm{s}} \Rightarrow \sigma \in V$.

iii) $\sigma \in \uparrow\tau \iff \tau \leq_s \sigma$.

iv) $\sigma \in \uparrow\{\tau \mid B \vdash_{\mathrm{s}} M{:}\tau\} \iff \sigma \in \{\tau \mid B \vdash_{\mathrm{s}} M{:}\tau\}$.

v) $\{x{:}\sigma\} \vdash_{\mathrm{s}} x{:}\tau \iff \sigma \leq_s \tau$.

Proof: Easy. ■

Theorem 2.6 *i) If* $B \vdash_{\mathrm{s}} M{:}\sigma$ *and* $\sigma \leq_s \tau$*, then* $B \vdash_{\mathrm{s}} M{:}\tau$.

ii) $\{\sigma \in \mathcal{T}_{\mathrm{S}} \mid B \vdash_{\mathrm{s}} M{:}\sigma\} \in \mathcal{F}_{\mathrm{S}}$.

Proof: i) By induction on $\leq$ s.

ii) By 2.5 (*iv*). ■

Definition 2.7 We define $F : \mathcal{F}_{\mathrm{S}} \to [\mathcal{F}_{\mathrm{S}} \to \mathcal{F}_{\mathrm{S}}]$ and $G : [\mathcal{F}_{\mathrm{S}} \to \mathcal{F}_{\mathrm{S}}] \to \mathcal{F}_{\mathrm{S}}$ by:

i) $F\ d\ e = d \cdot e$.

ii) $G\ f = \uparrow\{\sigma \to \tau \in \mathcal{T}_{\mathrm{S}} \mid \tau \in f(\uparrow\sigma)\}$.

It is easy to check that F and G are continuous.

Theorem 2.8 $\langle \mathcal{F}_S, \cdot \rangle$ *with F and G as defined in 2.7 is a lambda model.*

Proof: By [1].5.4.1 it is sufficient to prove that $F \circ G = id_{[\mathcal{F}_S \to \mathcal{F}_S]}$.

$$\begin{array}{ll} F \circ G\ f\ d = F\, G(f)\, d = G(f) \cdot d = \uparrow\{\sigma \to \tau \in \mathcal{T}_S \mid \tau \in f(\uparrow\sigma)\} \cdot d = & \\ \uparrow\{\mu \mid \exists \rho \in d\ [\rho \to \mu \in \uparrow\{\sigma \to \tau \mid \tau \in f(\uparrow\sigma)\}]\} = & (2.5\,(ii)) \\ \uparrow\{\mu \mid \exists \rho \in d\ [\mu \in f(\uparrow\rho)]\} = & \\ f(d). & \end{array}$$

■

Remark that $\mathcal{F}_S$ and the filter lambda model $\mathcal{F}$ defined in [2] are not isomorphic as complete lattices, since for example in $\mathcal{F}$ the filter $\uparrow(\sigma \cap \tau) \to \sigma$ is contained in $\uparrow\sigma \to \sigma$ but in $\mathcal{F}_S$ the strict filter $\uparrow_s(\sigma \cap \tau) \to \sigma$ is not contained in $\uparrow_s\sigma \to \sigma$. Moreover they are not isomorphic as lambda models since in $\mathcal{F}$ the meaning of $\lambda xy.xy$ is contained in the meaning of $\lambda x.x$, while this does not hold in $\mathcal{F}_S$ (see the examples after 2.11). Another difference is that while the analogue of G in $\mathcal{F}$ chooses the minimal representative of functions, this is not the case in $\mathcal{F}_S$.

Definition 2.9 Let ξ be a valuation of term variables in $\mathcal{F}_S$.

i) $[\![M]\!]_\xi$, the interpretation of terms in $\mathcal{F}_S$ via ξ is inductively defined by:

- *a)* $[\![x]\!]_\xi = \xi(x)$.
- *b)* $[\![MN]\!]_\xi = F\ [\![M]\!]_\xi\ [\![N]\!]_\xi$.
- *c)* $[\![\lambda x.M]\!]_\xi = G\ (\boldsymbol{\lambda} v \in \mathcal{F}_S.[\![M]\!]_{\xi(v/x)})$.

ii) $B_\xi = \{x{:}\sigma \mid \sigma \in \xi(x)\}$.

Theorem 2.10 *For all M, ξ: $[\![M]\!]_\xi = \{\sigma \in \mathcal{T}_S \mid B_\xi \vdash_s M{:}\sigma\}$.*

Proof: By induction on the structure of lambda terms.

i) $[\![x]\!]_\xi = \xi(x)$. Since $\{y{:}\rho \mid \rho \in \xi(y)\} \vdash_s x{:}\sigma \iff \sigma \in \xi(x)$.

ii)
$$\begin{array}{ll} [\![MN]\!]_\xi = \uparrow\{\tau \mid \exists \sigma\ [B_\xi \vdash_s N{:}\sigma\ \&\ B_\xi \vdash_s M{:}\sigma \to \tau]\} = & (2.3\,(ii)\ \&\ (iii)) \\ \uparrow\{\tau \mid B_\xi \vdash_s MN{:}\tau\} = & (2.6(ii)) \\ \{\tau \mid B_\xi \vdash_s MN{:}\tau\} & \end{array}$$

iii)
$$\begin{array}{ll} [\![\lambda x.M]\!]_\xi = \uparrow\{\sigma \to \tau \mid B_{\xi(\uparrow\sigma/x)} \vdash_s M{:}\tau\} = & (2.3\,(iii)) \\ \uparrow\{\sigma \to \tau \mid B_{\xi(\uparrow\sigma/x)} \vdash_s M{:}\tau\} = & (B'_\xi = B_\xi \backslash x) \\ \uparrow\{\sigma \to \tau \mid B'_\xi \cup \{x{:}\mu \mid \mu \in \uparrow\sigma\} \vdash_s M{:}\tau\} = & \\ \uparrow\{\sigma \to \tau \mid B'_\xi \cup \{x{:}\sigma\} \vdash_s M{:}\tau\} = & (2.3\,(iv)) \\ \uparrow\{\sigma \to \tau \mid B'_\xi \vdash_s \lambda x.M{:}\sigma \to \tau\} = & (2.3\,(v)) \\ \uparrow\{\sigma \to \tau \mid B_\xi \vdash_s \lambda x.M{:}\sigma \to \tau\} = & (2.3\,(iv)\ \&\ 2.5\,(iv)) \\ \{\rho \mid B_\xi \vdash_s \lambda x.M{:}\rho\}. & \end{array}$$

■

Corollary 2.11 *If $M =_\beta N$ and $B \vdash_s M{:}\sigma$, then $B \vdash_s N{:}\sigma$, so the following rule is a derived rule in $\vdash_s$:*

$$(=_\beta): \frac{M:\sigma}{N:\sigma}\ (M =_\beta N)$$

Proof: Since $\mathcal{F}_S$ is a lambda model, we know that if $M =_\beta N$, then $[\![M]\!]_\xi = [\![N]\!]_\xi$; so

$$\{\sigma \in \mathcal{T}_S \mid B_\xi \vdash_s M{:}\sigma\} = \{\sigma \in \mathcal{T}_S \mid B_\xi \vdash_s N{:}\sigma\}.$$

■

Notice that because of the way in which $\vdash_s$ is defined, Corollary 2.11 also holds if $\vdash_s$ is replaced by $\vdash_s$.

Example 2.12 By using 2.3 and 2.5 we can show the following:

i) If M is a closed term, then for all ξ, $[\![M]\!]_\xi = \{\sigma \in \mathcal{T}_S \mid \,\vdash_s M{:}\sigma\}$. So for closed terms we can omit the subscript ξ.

ii) $[\![\lambda xy.xy]\!]_\xi = \uparrow\{\rho\to\sigma\to\tau \mid \exists\sigma' \,[\rho \leq_s \sigma'\to\tau \;\&\; \sigma \leq_s \sigma']\}$.

iii) $[\![\lambda x.x]\!]_\xi = \uparrow\{\sigma\to\tau \mid \sigma \leq_s \tau\}$.

iv) $[\![\lambda x.xy]\!]_\xi = \xi(y)$.

If we take for example $\mu = (\sigma\to\tau)\to(\sigma\cap\rho)\to\tau$, then it is easy to check that $\mu \in [\![\lambda xy.xy]\!]_\xi$ and $\mu \notin [\![\lambda x.x]\!]_\xi$, so $[\![\lambda xy.xy]\!]_\xi$ is not contained in $[\![\lambda x.x]\!]_\xi$.

Notice that if M is a closed term, $[\![M]\!]_\xi$ is infinite. If M is not closed, it can be that $[\![M]\!]_\xi$ is finite since ξ can select also finite filters. However, we can limit $\mathcal{F}_S$ by selecting only infinite strict filters. Notice that this would still give us a lambda model that is different from $\mathcal{F}$.

Theorem 2.13 *If M is in normal form, then there are B and σ such that $B \vdash_s M{:}\sigma$, and in this derivation ω does not occur.*

Proof: By induction on the structure of lambda terms in normal form.

i) $M \equiv x$. Take σ strict, such that ω does not occur in σ. Then $\{x{:}\sigma\} \vdash_s x{:}\sigma$.

ii) $M \equiv \lambda x.M'$, with M' in normal form. By induction there are B and τ such that $B \vdash_s M'{:}\tau$ and ω does not occur in this derivation. In order to perform the $(\to I)$-step, B must contain (whether or not x is free in M') a statement with subject x and predicate, say, σ. But then of course $B\backslash x \vdash_s \lambda x.M'{:}\sigma\to\tau$ and ω does not occur in this derivation.

iii) $M \equiv xM_1 \ldots M_n$, with $M_1, \ldots, M_n$ in normal form. By induction there are $B_1,\ldots,B_n$ and $\sigma_1,\ldots,\sigma_n$ such that for every $i \in \{1,\ldots,n\}$, $B_i \vdash_s M_i{:}\sigma_i$ and ω does not occur in these derivations.

Take τ strict, such that ω does not occur in τ, and $B = \cap_{i\in\{1,\ldots,n\}} B_i \cap \{x{:}\sigma_1\to\cdots\to\sigma_n\to\tau\}$. Then $B \vdash_s xM_1\ldots M_n{:}\tau$ and in this derivation ω does not occur. ■

Theorem 2.14 *If M is in head normal form, then there are B and σ such that $B \vdash_s M{:}\sigma$.*

Proof: By induction on the structure of lambda terms in head normal form.

i) $M \equiv x$. Take σ strict, then $x{:}\sigma \vdash_s x{:}\sigma$.

ii) $M \equiv \lambda x.M'$, with M' in head normal form. By induction there are B and τ such that $B \vdash_s M'{:}\tau$. As in the previous theorem, B must contain a statement with subject x and predicate, say, σ. But then of course $B\backslash x \vdash_s \lambda x.M'{:}\sigma\to\tau$.

iii) $M \equiv xM_1 \ldots M_n$, with $M_1, \ldots, M_n$ lambda terms. Take τ strict, then also $\omega\to\omega\to\cdots\to\omega\to\tau$ is strict, and $\{x{:}\omega\to\omega\to\cdots\to\omega\to\tau\} \vdash_s xM_1\ldots M_n{:}\tau$. ■

Theorem 2.15 $\exists B,\sigma\,[B \vdash_s M{:}\sigma \;\&\; B,\sigma$ *ω-free*$] \iff$ *M has a normal form.*

Proof: $(\Rightarrow)$: If $B \vdash_s M{:}\sigma$ and B, σ ω-free, then $B \vdash_s M{:}\sigma$ and B, σ ω-free. Then by 1.3 (*iv*) M has a normal form.

$(\Leftarrow)$: By 2.13 and 2.11. ■

Notice that in part (ii) of the proof, because of Corollary 2.11 we can only state that if $M =_\beta N$ and $B \vdash_s M{:}\sigma$, then $B \vdash_s N{:}\sigma$. From 2.13 we can conclude that B and σ do not contain ω, but the property that ω does not occur at all in the derivation is, in general, lost.

Theorem 2.16 $\exists B,\sigma\,[B \vdash_s M{:}\sigma] \iff$ *M has a head normal form.*

Proof: $(\Rightarrow)$: If $B \vdash_s M{:}\sigma$, then $B \vdash M{:}\sigma$ and $\sigma \neq \omega$. Then by 1.3 (*iii*) M has a head normal form.

$(\Leftarrow)$: By 2.14 and 2.11. ■

Corollary 2.17 *i)* $\exists B,\sigma\ [B \vdash_s M:\sigma\ \&\ B,\sigma\ \omega - free] \iff M$ *has a normal form.*
ii) $\exists B,\sigma\ [B \vdash_s M:\sigma\ \&\ \sigma \neq \omega] \iff M$ *has a head normal form.* ■

2.2 The relation between $\vdash_s$ and $\vdash$

The intersection type assignment is not conservative over the strict type assignment. So the following does not hold:

> Suppose all types occurring in B and σ are elements of $\mathcal{T}_S$. Then $B \vdash_s M:\sigma \iff B \vdash M:\sigma$.

As a counter example for $\Leftarrow$, take $\{x:\sigma\rightarrow\sigma\} \vdash x:(\sigma\cap\tau)\rightarrow\sigma$. It is not possible to derive $x : (\sigma\cap\tau)\rightarrow\sigma$ form the basis $\{x:\sigma\rightarrow\sigma\}$ in $\vdash_s$.

Of course the implication in the other direction holds: $B \vdash_s M:\sigma$ implies $B \vdash M:\sigma$. The relation between the two systems is however stronger. Theorem 2.23 states that every statement obtainable in the intersection type assignment system can be obtained by a derivation in which the rule $(\leq)$ is, if necessary, only performed as the last step. The proof is based on the fact that for every $\sigma \in \mathcal{T}$ there is a $\sigma' \in \mathcal{T}_S$ such that $\sigma \sim \sigma'$ (Lemma 2.19; the same result has been stated in [14], \$4), and the approximation theorem as given in [23].

Definition 2.18 *i*) The set $\mathcal{N}$ of $\lambda\perp$-normal forms or approximate normal forms is inductively defined by:

- *a*) All term variables are in $\mathcal{N}$, $\perp$ is in $\mathcal{N}$.
- *b*) If A is in $\mathcal{N}$, $A \neq \perp$, then $\lambda x.A$ is in $\mathcal{N}$.
- *c*) If $A_1,\ldots,A_n$ are in $\mathcal{N}$, then $xA_1\cdots A_n$ is in $\mathcal{N}$.

ii) $A \in \mathcal{N}$ is a direct approximant of $M \in \lambda$ if A matches M except for occurrences of $\perp$.

iii) $A \in \mathcal{N}$ is an approximant of $M \in \lambda$ (notation: $A \leq M$) if there is an $M' =_\beta M$ such that A is a direct approximant of M'.

iv) $\mathcal{A}(M) = \{A \in \mathcal{N} \mid A \leq M\}$.

v) The type assignment rules of Definition 1.2 (*i*) and 2.2 (*i*) are generalised to elements of $\mathcal{N}$ by allowing the terms to be elements of $\lambda\perp$.

Lemma 2.19 (CF. [14]) *For every $\sigma \in \mathcal{T}$ there is a $\sigma' \in \mathcal{T}_S$ such that $\sigma \sim \sigma'$.*

Proof: By induction on the structure of types in $\mathcal{T}$.

i) $\sigma = \omega$, or σ is a type variable: trivial.

ii) $\sigma = \rho\rightarrow\tau$. By induction there are ρ' and $\tau' \in \mathcal{T}_S$ such that $\rho \sim \rho'$ and $\tau \sim \tau'$.

- *a*) $\tau' = \omega$. Take $\sigma' = \omega$.
- *b*) $\tau' = \tau_1\cap\cdots\cap\tau_m$, each $\tau_i \in \mathcal{T}_s$. Take $\sigma' = (\rho'\rightarrow\tau_1)\cap\cdots\cap(\rho'\rightarrow\tau_1)$.
- *c*) τ' is strict, then take $\sigma' = \rho'\rightarrow\tau'$

iii) $\sigma = \rho\cap\tau$. By induction there are ρ' and $\tau' \in \mathcal{T}_S$ such that $\rho \sim \rho'$ and $\tau \sim \tau'$.

- *a*) $\rho' = \omega$. Take $\sigma' = \tau'$.
- *b*) $\tau' = \omega$. Take $\sigma' = \rho'$.
- *c*) $\rho' \neq \omega\ \&\ \tau' \neq \omega$. Take $\sigma' = \rho'\cap\tau'$. ■

Notice that Lemma 2.19 is not a proof for the statement that $\mathcal{T}_S$ modulo $\sim_s$ is isomorphic to $\mathcal{T}$ modulo $\sim$. For example, take $\sigma_1 = (\rho\cap\tau)\rightarrow\rho$ and $\sigma_2 = (\tau\cap\rho)\rightarrow\rho$. Then obviously $\sigma_1 \sim \sigma_2$, $\sigma_1' = \sigma_1$, $\sigma_2' = \sigma_2$, but not $\sigma_1' \sim_s \sigma_2'$. By proving Lemma 2.19 we only prove that we can find a σ' for each σ, not that this σ' is unique. In fact, by replacing 2.19 (*iii.c*) by

> c. $\rho' \neq \omega\ \&\ \tau' \neq \omega$. Take $\sigma' = \rho'\cap\tau'$ or $\sigma' = \tau'\cap\rho'$.

we would be able to find both σ_1' and σ_2' as types in $\mathcal{T}_S$ that fit our purpose. See also the remark after 2.28.

We will now prove the main theorem of this subsection, by showing that the $\vdash_s$ system is in fact the nucleus of the intersection type discipline. We will do this by proving first for terms in $\mathcal{N}$ that the derivation rule $(\leq)$ can be transferred to the very end of a derivation and afterwards generalising this result to arbitrary lambda terms.

Definition 2.20 We write $B \geq B'$, if the following is true. If x occurs in B, then x occurs in B', and if $\sigma_1,\ldots,\sigma_n$ are all predicates of statements that have x as subject in B then there are $\tau_1,\ldots,\tau_m$, predicates of statements that have x as subject in B', such that $\sigma_1\cap\cdots\cap\sigma_n \geq \tau_1\cap\cdots\cap\tau_m$.

Theorem 2.21 *If A is in $\lambda\bot$-normal form and $B \vdash A:\sigma$ then there are B', $\sigma' \in \mathcal{T}_S$ such that $B' \vdash_s A:\sigma'$, $\sigma' \leq \sigma$ and $B' \geq B$.*

Proof: The proof is given by induction on the structure of terms in $\lambda\bot$-normal form. All cases where $\sigma \sim \omega$ are trivial, because then we can take $B' = \emptyset$ and $\sigma' = \omega$. Therefore in the rest of the proof, we will assume $\sigma \not\sim \omega$.

i) $B \vdash x:\sigma$. By 1.3 (*v*) there are $x{:}\sigma_1,\ldots,x{:}\sigma_n$ in B such that $\sigma_1\cap\cdots\cap\sigma_n \leq \sigma$. By Lemma 2.19 there is an $\sigma' \in \mathcal{T}_S$ such that $\sigma_1\cap\cdots\cap\sigma_n \sim \sigma'$. Then $\{x{:}\sigma'\} \geq B$ and $\sigma' \leq \sigma$.

ii) $B \vdash \lambda x.A':\sigma$, with $A' \neq \bot$. Then there are $\rho_1, \ldots, \rho_n, \mu_1, \ldots, \mu_n$ such that $\sigma = (\rho_1 \to \mu_1)\cap \cdots\cap(\rho_n \to \mu_n)$. So, by $(\cap E)$ and 1.3 (*ii*) for every $i \in \{1,\ldots,n\}$ we have $B\cup\{x{:}\rho_i\} \vdash A':\mu_i$. We can assume, without loss of generality, that each μ_i is an element of $\mathcal{T}_s$. By induction there are B_i and ρ_i', $\mu_i' \in \mathcal{T}_S$ such that $B_i\cup\{x{:}\rho_i'\} \vdash_s A':\mu_i'$, $\mu_i' \leq \mu_i$ and $B_i\cup\{x{:}\rho_i'\} \geq B\cup\{x{:}\rho_i\}$. We can assume, without loss of generality, that each μ_i' is an element of $\mathcal{T}_s$. Then for all $i \in \{1,\ldots,n\}$ $B_i \vdash_s \lambda x.A':\rho_i' \to \mu_i'$, $\rho_i' \to \mu_i' \leq \rho_i \to \mu_i$ and $B_i \geq B$. So

$$\cup_{i\in\{1,\ldots,n\}}B_i \vdash_s \lambda x.A':(\rho_1' \to \mu_1')\cap\cdots\cap(\rho_n' \to \mu_n'),$$

$(\rho_1' \to \mu_1')\cap\cdots\cap(\rho_n' \to \mu_n') \leq \sigma$ and $\cup_{i\in\{1,\ldots,n\}}B_i \geq B$.

iii) $B \vdash xA_1\ldots A_n:\sigma$. By 1.3 (*i*) there are $\tau_1,\ldots,\tau_n \in \mathcal{T}$ such that $B \vdash x:\tau_1 \to\cdots\to \tau_n \to \sigma$, $B \vdash A_1:\tau_1$, $\ldots$, and $B \vdash A_n:\tau_n$. By induction $B_i \vdash_s A_i:\tau_i'$, $\tau_i' \leq \tau_i$ and $B_i \geq B$. Take $B' = \cup_{i\in\{1,\ldots,n\}}B_i$, then $B' \geq B$. By Lemma 2.19 there is a $\sigma' \in \mathcal{T}_S$ such that $\sigma \sim \sigma'$. Let $\sigma' = \sigma_1\cap\cdots\cap\sigma_k$ where each $\sigma_i \in \mathcal{T}_s$ and $k \geq 1$. Because of

$$\tau_1 \to\cdots\to \tau_n \to \sigma \leq (\tau_1' \to\cdots\to \tau_n' \to \sigma_1)\cap\cdots\cap(\tau_1' \to\cdots\to \tau_n' \to \sigma_k)$$

and 1.3 (*v*), we have

$$B'\cap\{x{:}(\tau_1' \to\cdots\to \tau_n' \to \sigma_1)\cap\cdots\cap(\tau_1' \to\cdots\to \tau_n' \to \sigma_k)\} \geq B.$$

Also for every $f \in \{1,\ldots,k\}$ $B'\cap\{x{:}\tau_1' \to\cdots\to \tau_n' \to \sigma_f\} \vdash_s xA_1\cdots A_n:\sigma_f$.
So $B'\cap\{x{:}(\tau_1' \to\cdots\to \tau_n' \to \sigma_1)\cap\cdots\cap(\tau_1' \to\cdots\to \tau_n' \to \sigma_k)\} \vdash_s xA_1\ldots A_n:\sigma'$. ∎

Theorem 2.22 *i) $B \vdash M:\sigma \iff \exists A \in \mathcal{A}(M)\ [B \vdash A:\sigma]$.*
ii) $B \vdash_s M:\sigma \iff \exists A \in \mathcal{A}(M)\ [B \vdash_s A:\sigma]$.

Proof: i) [23].2.13.

ii) The structure of the proof of [23].2.13 is the same as the structure of the proof for this part. ∎

Theorem 2.23 *If $B \vdash M:\sigma$ then there are B', $\sigma' \in \mathcal{T}_S$ such that $B' \vdash_s M:\sigma'$, $\sigma' \leq \sigma$ and $B' \geq B$.*

Proof: If $B \vdash M:\sigma$ then, by 2.22 (*i*) there is an $A \in \mathcal{A}(M)$ such that $B \vdash A:\sigma$. Then by Definition 2.18 there is an M' such that $M' =_\beta M$ and A is a direct approximant of M'. By Theorem 2.21 there are B', $\sigma' \in \mathcal{T}_S$ such that $B' \vdash_s A:\sigma'$, $\sigma' \leq \sigma$ and $B' \geq B$. Then by Theorem 2.22 (*ii*)

$B' \vdash_s M : \sigma'$. ■

2.3 Soundness and completeness of strict type assignment

In this subsection we will prove completeness for the $\vdash_s$ system. This is done in a way very similar to the one used in [2], using the strict filter lambda model as defined in subsection 2.1. At one very crucial point the completeness proof in this subsection differs from the one in [2]. In that paper the simple type semantic is inductively defined whereas our approach will be to give a map from $\mathcal{T}_S$ to $\wp(\mathcal{F}_S)$ and prove that it is a type interpretation. It will be a different kind of type interpretation than the one used in [2], because the latter would not suffice in our case.

Following essentially [19], we distinguish between several kinds of type interpretations.

Definition 2.24 *i*) Let $\mathcal{M} = \langle \mathcal{D}, \cdot, \varepsilon \rangle$ be a continuous lambda model. A mapping $v : \mathcal{T} \rightarrow \wp(\mathcal{D})$ is a type interpretation if and only if:

a) $\{\varepsilon \cdot d \mid \forall e\, [e \in v(\sigma) \Rightarrow d \cdot e \in v(\tau)]\} \subseteq v(\sigma \rightarrow \tau)$.

b) $v(\sigma \rightarrow \tau) \subseteq \{d \mid \forall e\, [e \in v(\sigma) \Rightarrow d \cdot e \in v(\tau)]\}$.

c) $v(\sigma \cap \tau) = v(\sigma) \cap v(\tau)$.

ii) Following [15] we say that a type interpretation is simple if and only if:
$v(\sigma \rightarrow \tau) = \{d \mid \forall e\, [e \in v(\sigma) \Rightarrow d \cdot e \in v(\tau)]\}$.

iii) On the other hand, a type interpretation is called an F type interpretation if it satisfies:
$v(\sigma \rightarrow \tau) = \{\varepsilon \cdot d \mid \forall e\, [e \in v(\sigma) \Rightarrow d \cdot e \in v(\tau)]\}$.

Notice that in part (ii) the containment relation $\subseteq$ of part (i.b) is replaced by $=$, and that in part (iii) the same is done with regard to part (i.a).

These notions of type interpretation lead naturally to the following definitions for semantic satisfiability (called respectively inference, simple and F-semantics).

Definition 2.25 We define $\models$ by: (where $\mathcal{M}$ is a lambda model, ξ a valuation and v a type interpretation)

i) $\mathcal{M}, \xi, v \models M : \sigma \iff [\![M]\!]_\xi^{\mathcal{M}} \in v(\sigma)$.

ii) $\mathcal{M}, \xi, v \models B \iff \mathcal{M}, \xi, v \models x{:}\sigma$ for every $x{:}\sigma \in B$.

iii) *a*) $B \models M : \sigma \iff \forall \mathcal{M}, \xi, v\, [\mathcal{M}, \xi, v \models B \Rightarrow \mathcal{M}, \xi, v \models M : \sigma]$.

b) $B \models_s M : \sigma \iff \forall \mathcal{M}, \xi,$ simple type interpretations $v\, [\mathcal{M}, \xi, v \models B \Rightarrow \mathcal{M}, \xi, v \models M : \sigma]$.

c) $B \models_F M : \sigma \iff \forall \mathcal{M}, \xi, F$ type interpretations $v\, [\mathcal{M}, \xi, v \models B \Rightarrow \mathcal{M}, \xi, v \models M : \sigma]$.

If no confusion is possible, we will omit the superscript on $[\![\cdots]\!]$.

Theorem 2.26 Soundness: $B \vdash_s M : \sigma \Rightarrow B \models M : \sigma$.

Proof: By induction on the structure of derivations. ■

The notion of derivability $\vdash$ as defined in 1.2 is not sound for $\models$. Take for example the statement $\lambda x.x : (\sigma \rightarrow \sigma) \rightarrow (\sigma \cap \tau) \rightarrow \sigma$. This statement is derivable in the system $\vdash$, but it is not valid in the strict filter lambda model.

Definition 2.27 *i*) We define a map $v_0 : \mathcal{T}_S \rightarrow \wp(\mathcal{F}_S)$ by $v_0(\sigma) = \{d \in \mathcal{F}_S \mid \sigma \in d\}$.

ii) $\xi_B(x) = \{\sigma \in \mathcal{T}_S \mid B \vdash_s x : \sigma\}$.

Theorem 2.28 *The map v_0 is a type interpretation.*

Proof: We check the conditions of 2.24 (*i*).

$$\begin{array}{lll}
i)\ \forall e\ [e \in \nu_0(\sigma) \Rightarrow d \cdot e \in \nu_0(\tau)] & \Rightarrow & \\
\quad \forall e\ [e \in \nu_0(\sigma) \Rightarrow \varepsilon \cdot d \cdot e \in \nu_0(\tau)] & \Rightarrow & (\text{take } e = \uparrow\sigma) \\
\quad \tau \in \varepsilon \cdot d \cdot \uparrow\sigma & \Rightarrow & (2.5\,(ii)) \\
\quad \exists \rho \in \uparrow\sigma, \nu \in d, \eta\ [\nu \leq s\eta \rightarrow \tau \,\&\, \rho \leq_s \eta] & \Rightarrow & (2.5\,(iii)) \\
\quad \exists \nu \in d, \eta\ [\nu \leq_s \eta \rightarrow \tau \,\&\, \sigma \leq_s \eta] & \Rightarrow & \\
\quad \sigma \rightarrow \tau \in \uparrow\{\rho \rightarrow \mu \mid \exists \nu \in d, \eta\ [\nu \leq_s \eta \rightarrow \mu \,\&\, \rho \leq_s \eta]\} & \Rightarrow & \\
\quad \sigma \rightarrow \tau \in \uparrow\{\eta \mid \exists \nu \in d\ [\nu \rightarrow \eta \in e]\} & \Rightarrow & \\
\quad e \cdot d \in \nu_0(\sigma \rightarrow \tau). & &
\end{array}$$

ii) Easy.

iii) Trivial. ■

Notice that although $\nu_0(\sigma\cap\tau) = \nu_0(\tau\cap\sigma)$, the sets $\nu_0((\sigma\cap\tau)\rightarrow\sigma)$ and $\nu_0((\tau\cap\sigma)\rightarrow\sigma)$ are incompatible. We can only show that both contain

$$\{\varepsilon \cdot d \mid \forall e\ [e \in \nu_0(\sigma)\cap\nu_0(\tau) \Rightarrow d \cdot e \in \nu_0(\sigma)]\}$$

and are both contained in

$$\{d \mid \forall e\ [e \in \nu_0(\sigma)\cap\nu_0(\tau) \Rightarrow d \cdot e \in \nu_0(\sigma)]\}.$$

However, it is not difficult to prove that $\varepsilon \cdot \uparrow(\sigma\cap\tau)\rightarrow\sigma = \varepsilon \cdot \uparrow(\tau\cap\sigma)\rightarrow\sigma$, so the filters $\uparrow(\sigma\cap\tau)\rightarrow\sigma$ and $\uparrow(\tau\cap\sigma)\rightarrow\sigma$ represent the same function.

Lemma 2.29 *i) $B \vdash_{\mathrm{s}} M{:}\sigma$ if and only if $B_{\xi_B} \vdash_{\mathrm{s}} M{:}\sigma$.*

ii) $\mathcal{F}_{\mathrm{S}}, \xi_B, \nu_0 \models B$.

Proof: *i*) Because for every x, $\xi_B(x)$ is a strict filter.

ii) $x{:}\sigma \in B \Rightarrow (i)\ \sigma \in \{\tau \mid B_{\xi_B} \vdash_{\mathrm{s}} x{:}\tau\} \Rightarrow \sigma \in [\![x]\!]_{\xi_B}$. So $[\![x]\!]_{\xi_B} \in \{d \in \mathcal{F}_{\mathrm{S}} \mid \sigma \in d\} = \nu_0(\sigma)$. ■

The system of [2] has been proved complete with respect to the simple type semantics. The system $\vdash_{\mathrm{s}}$ however is not complete in this semantics. This is due to the fact that if we take v to be a type interpretation from $\mathcal{T}_{\mathrm{S}}$ to $\wp(\mathcal{F}_{\mathrm{S}})$, the set

$$\{d \mid \forall e\ [e \in v(\sigma) \Rightarrow d \cdot e \in v(\tau)]\}$$

is not contained in $v(\sigma\rightarrow\tau)$, since we don't allow ω or an intersection type scheme at the right hand side of an arrow type scheme. If instead we use the notion of type interpretation as defined in 2.24 (*i*), because of Theorem 2.28 completeness can be proved.

Theorem 2.30 Completeness: *Let $\sigma \in \mathcal{T}_{\mathrm{S}}$, then $B \models M:\sigma \Rightarrow B \vdash_{\mathrm{s}} M{:}\sigma$.*

$$\begin{array}{lr}
\textit{Proof: } B \models M:\sigma \Rightarrow & (2.25\ (iii.a),\ 2.29\,(i)\ \&\ 2.28) \\
\quad \mathcal{F}_{\mathrm{S}}, \xi_B, \nu_0 \models M:\sigma \Rightarrow & (2.25\,(i)\ \&\ 2.28) \\
\quad [\![M]\!]_{\xi_B} \in \nu_0(\sigma) \Rightarrow & (2.27\,(i)) \\
\quad \sigma \in [\![M]\!]_{\xi_B} \Rightarrow & (2.10) \\
\quad B_{\xi_B} \vdash_{\mathrm{s}} M{:}\sigma \Rightarrow & (2.29\,(i)) \\
\quad B \vdash_{\mathrm{s}} M{:}\sigma. &
\end{array}$$

■

3 The system without ω

In this section we present a type assignment system that is a restriction of the intersection type assignment system. The restriction is the elimination of the type constant ω. We will show that the intersection type assignment system without ω yields a filter model for the λI-calculus (subsection 3.1), show that for the λI-calculus the intersection type assignment is conservative over the one without ω (subsection 3.2) and prove that this type assignment is complete for the λI-calculus with respect to the simple type semantics (subsection 3.3). Furthermore we will prove that each term typeable by the system without ω is strongly normalisable (subsection 3.4).

While obtaining these results we could of course use the result of the previous section and look at the system without $\leq$ and without ω, but since this is a more restricted system we prefer the approach we use in this section. Also the proofs of various lemmas in subsection 3.4 are greatly facilitated by the presence of derivation rule $(\leq)$. In fact, the strong normalisation property for the system without $\leq$ and ω follows immediately from the results of 3.4. Moreover we could prove a completeness result for this system with respect to the inference semantics.

3.1 ω-free derivations

In this subsection we present a restriction of the intersection type assignment system in which the type ω is removed. This system yields a filter λI-model.

Definition 3.1 *i*) $\mathcal{T}_{\not\omega}$, the set of ω-free types is inductively defined by:

a) All type variables $\varphi_0, \varphi_1, \ldots \in \mathcal{T}_{\not\omega}$.

b) If $\sigma, \tau \in \mathcal{T}_{\not\omega}$, then $\sigma\cap\tau, \sigma\rightarrow\tau \in \mathcal{T}_{\not\omega}$.

ii) On $\mathcal{T}_{\not\omega}$ the type inclusion relation $\leq$ is as defined in 1.1 (*ii*), but without rules 1.1 (*ii.b*) and 1.1 (*ii.c*).

iii) If $M:\sigma$ is derivable from a basis B, using only ω-free types and the derivation rules $(\cap I)$, $(\cap E)$, $(\rightarrow I)$, $(\rightarrow E)$ or $(\leq)$ of the system in 1.2 (*i*), we write $B \vdash_{\not\omega} M:\sigma$.

Lemma 3.2 i) $B \vdash_{\not\omega} MN:\sigma \iff \exists\,\tau\,[B \vdash_{\not\omega} M:\tau\rightarrow\sigma \;\&\; B \vdash_{\not\omega} N:\tau]$.

ii) $B \vdash_{\not\omega} \lambda x.M:\sigma\rightarrow\tau \iff B\backslash x \cup \{x{:}\sigma\} \vdash_{\not\omega} M:\tau$.

iii) If $B \vdash_{\not\omega} \lambda x.M:\rho$, *then there are* $\sigma_1,\ldots,\sigma_n$, $\tau_1,\ldots,\tau_n$ *such that* $\rho = (\sigma_1\rightarrow\tau_1)\cap\cdots\cap(\sigma_n\rightarrow\tau_n)$.

iv) If $B\backslash z \cup \{z{:}\sigma\} \vdash_{\not\omega} Mz:\tau$ *and* $z \notin fv(M)$, *then* $B \vdash_{\not\omega} M:\sigma\rightarrow\tau$.

Proof: By induction on the structure of derivations, using 1.3 (*ii*) to prove (ii). The proof given for this part in [2] does not depend on ω. ∎

Definition 3.3 *i*) A subset d of $\mathcal{T}_{\not\omega}$ is called an I-filter if

a) $\sigma, \tau \in d \Rightarrow \sigma\cap\tau \in d$.

b) $\sigma \geq \tau \in d \Rightarrow \sigma \in d$.

ii) $\mathcal{F}_{-\omega} = \{d \subseteq \mathcal{T}_{\not\omega} \mid d \text{ is an I-filter}\}$. We define application on $\mathcal{F}_{-\omega}$, $\cdot : \mathcal{F}_{-\omega} \times \mathcal{F}_{-\omega} \rightarrow \mathcal{F}_{-\omega}$ by:

$$d \cdot e = \{\tau \mid \exists\sigma \in e\,[\sigma\rightarrow\tau \in d]\}.$$

iii) If V is a subset of $\mathcal{T}_{\not\omega}$, then $\uparrow_\omega V$ is the smallest I-filter that contains V. Also $\uparrow_\omega\sigma = \uparrow_\omega\{\sigma\}$. If no confusion is possible, we will omit the subscript on $\uparrow$.

Notice that the empty set, $\emptyset$ is the bottom element of $\mathcal{F}_{-\omega}$.

Lemma 3.4 Suppose f is a continuous function from $\mathcal{F}_{-\omega}$ to $\mathcal{F}_{-\omega}$. Then for every $\rho, \mu \in \mathcal{T}_{\not\omega}$: if $\rho\rightarrow\mu \in \uparrow\{\sigma\rightarrow\tau \mid \tau \in f(\uparrow\sigma)\}$, then $\mu \in f(\uparrow\rho)$.

Proof: By definition of filters and the fact that f is continuous. ∎

Let $\langle\mathcal{D},\leq\rangle$ be a cpo with least element $\perp$. The set of strict functions is defined as usual, i.e. as the set of continuous functions that at least map $\perp$ onto $\perp$. We denote by $[\mathcal{D}\rightarrow_\perp\mathcal{D}]$ the set of strict functions from $\mathcal{D}$ to $\mathcal{D}$.

Definition 3.5 We define $F : \mathcal{F}_{-\omega} \to [\mathcal{F}_{-\omega} \to_\perp \mathcal{F}_{-\omega}]$ and $G : [\mathcal{F}_{-\omega} \to_\perp \mathcal{F}_{-\omega}] \to \mathcal{F}_{-\omega}$ by:

i) $F\ d\ e = d \cdot e$.

ii) $G\ f = \uparrow\{\sigma \to \tau \in \mathcal{T}_{\not\omega} \mid \tau \in f(\uparrow\sigma)\}$.

It is again easy to check that F and G are continuous.

Definition 3.6 ([16]) Let $\mathcal{D}$ be a set and $\cdot$ a binary relation on $\mathcal{D}$. The structure $\langle \mathcal{D}, \cdot, \varepsilon \rangle$ is called a λI-model if and only if in $\mathcal{D}$ there are five elements **i**, **b**, **c**, **s** and ε that satisfy the following conditions:

i) $\mathbf{i} \cdot d = d$.

ii) $((\mathbf{b} \cdot d) \cdot e) \cdot f = d \cdot (e \cdot f)$.

iii) $((\mathbf{c} \cdot d) \cdot e) \cdot f = (d \cdot f) \cdot e$.

iv) $((\mathbf{s} \cdot d) \cdot e) \cdot f = (d \cdot f) \cdot (e \cdot f)$.

v) $(\varepsilon \cdot d) \cdot e = d \cdot e \;\&\; \forall d \in \mathcal{D}\ [e \cdot d = f \cdot d \Rightarrow \varepsilon \cdot e = \varepsilon \cdot f] \;\&\; \varepsilon \cdot \varepsilon = \varepsilon$.

Moreover, in [10] the following is stated:

Proposition 3.7 ([10]) *If $\langle \mathcal{D}, \leq \rangle$ is a cpo and there are continuous maps $F : \mathcal{D} \to [\mathcal{D} \to_\perp \mathcal{D}]$ and $G : [\mathcal{D} \to_\perp \mathcal{D}] \to \mathcal{D}$ such that:*

i) $F \circ G = id_{[\mathcal{D} \to_\perp \mathcal{D}]}$

ii) $G \circ F \in [\mathcal{D} \to_\perp \mathcal{D}]$

Then D is a λI-model. ■

Theorem 3.8 *F and G as defined in 3.5 yield a λI-model.*

Proof: It is sufficient to check that the conditions of 3.7 are fulfilled.

i) $(F \circ G(f))(d) =$
$\{\mu \mid \exists \rho \in d[\rho \to \mu \in \uparrow\{\sigma \to \tau \mid \tau \in f(\uparrow\sigma)\}]\} =$ (3.4)
$\{\mu \mid \exists \rho \in d[\mu \in f(\uparrow\rho)]\} =$
$f(d)$.

ii) $G \circ F\ (\emptyset) = \uparrow\{\rho \to \mu \mid \mu \in \{\sigma \mid \exists \tau \in \uparrow\rho[\tau \to \sigma \in \emptyset\,]\}\} = \emptyset$. ■

That the type discipline without ω gives rise to a model for the λ I-calculus, is also proved in [16]. The technique used there is to build, using Scott's inverse limit construction, a model M2 satisfying the equation $\mathcal{D} \simeq \mathcal{P}_\omega \times [\mathcal{D} \to_\perp \mathcal{D}]$, with $\mathcal{D}_0 = \mathcal{P}_\omega$ (where $\mathcal{P}_\omega$ is the powerset of natural numbers) and $i : \mathcal{D}_0 \to \mathcal{P}_\omega \times [\mathcal{D}_0 \to \perp \mathcal{D}_0]$ is defined by $i(d) = \langle d, \lambda x.\perp \rangle$ (see also [1], exercise 18.4.26 and [20].)

It is straightforward to verify that $\mathcal{F}_{-\omega}$ is a solution of the same domain equation.

Definition 3.9 Let ξ be a valuation of term variables in $\mathcal{F}_{-\omega}$.

i) $[\![M]\!]_\xi$, the interpretation of λI-terms in $\mathcal{F}_{-\omega}$ via ξ is inductively defined by:

a) $[\![x]\!]_\xi = \xi(x)$.

b) $[\![MN]\!]_\xi = F\ [\![M]\!]_\xi\ [\![N]\!]_\xi$.

c) $[\![\lambda x.M]\!]_\xi = G\ (\lambda\!\!\lambda v \in \mathcal{F}_{-\omega}.[\![M]\!]_{\xi(v/x)})$.

ii) $B_\xi = \{x{:}\sigma \mid \sigma \in \xi(x)\}$.

Notice that $\lambda\!\!\lambda$ is well defined in λI-models, since $(\lambda\!\!\lambda v \in \mathcal{F}_{-\omega}.[\![M]\!]_{\xi(v/x)})\emptyset = \emptyset$.

Theorem 3.10 *For all $M \in \lambda I$, ξ : $[\![M]\!]_\xi = \{\sigma \mid B_\xi \vdash_{\not\omega} M{:}\sigma\}$.*

Proof: By induction on the structure of lambda terms.

i) $[\![x]\!]_{\xi} = \xi(x)$. Since $\{y{:}\rho \mid \rho \in \xi(y)\} \vdash_{\not\omega} x{:}\sigma \Longleftrightarrow \sigma \in \xi(x)$.

ii) $[\![MN]\!]_{\xi} =$

$\{\tau \mid \exists \sigma\, [B_{\xi} \vdash_{\not\omega} N{:}\sigma \;\&\; B_{\xi} \vdash_{\not\omega} M{:}\sigma \to \tau]\} =$ (3.2 (*i*))

$\{\tau \mid B_{\xi} \vdash_{\not\omega} MN{:}\tau\}$.

iii) $[\![\lambda x.M]\!]_{\xi} =$

$\uparrow\{\sigma \to \tau \mid \tau \in \{\rho \mid B_{\xi(\uparrow\sigma/x)} \vdash_{\not\omega} M{:}\rho\}\} =$

$\uparrow\{\sigma \to \tau \mid B_{\xi(\uparrow\sigma/x)} \vdash_{\not\omega} M{:}\tau\} =$ $(B'_{\xi} = B_{\xi}\backslash x)$

$\uparrow\{\sigma \to \tau \mid B'_{\xi} \cup \{x{:}\rho \mid \rho \in \uparrow\sigma\} \vdash_{\not\omega} M{:}\tau\} =$

$\uparrow\{\sigma \to \tau \mid \exists \mu \in \uparrow\sigma [B'_{\xi} \cup \{x{:}\mu\} \vdash_{\not\omega} M{:}\tau]\} =$ (3.2 (*ii*))

$\uparrow\{\sigma \to \tau \mid \exists \mu \in \uparrow\sigma [B'_{\xi} \vdash_{\not\omega} \lambda x.M{:}\mu \to \tau]\} =$ $(\leq)$

$\uparrow\{\sigma \to \tau \mid B'_{\xi} \vdash_{\not\omega} \lambda x.M{:}\sigma \to \tau\} =$ (3.2 (*iii*))

$\{\rho \mid B'_{\xi} \vdash_{\not\omega} \lambda x.M{:}\rho\}$. ■

Remark that F and G do not yield a lambda model. For example: take $\mathbf{Z} = \lambda xy.y$, $\mathbf{D} = \lambda x.xx$ and $\mathbf{I} = \lambda x.x$. Then clearly $\mathbf{Z}(\mathbf{DD}) =_{\beta} \mathbf{I}$ and $\vdash_{\not\omega} \mathbf{I}{:}\sigma \to \sigma$ but we cannot give a derivation without ω for $\mathbf{Z}(\mathbf{DD}) : \sigma \to \sigma$.

3.2 The relation between $\vdash_{-\omega}$ and $\vdash$

Type assignment in the intersection type assignment system is not fully conservative over the type assignment without ω. If for example we have $B \vdash M{:}\sigma$ such that B and σ are ω-free, but M contains a sub-term that has no normal form, ω is needed in the derivation. (See the final remark of the previous subsection.)

However, we can prove that for every lambda-term M such that $B \vdash M{:}\sigma$ with B and σ ω-free, there is an M' such that M reduces to M' and $B \vdash_{\not\omega} M'{:}\sigma$. We will show this by proving for terms in normal form that each ω-free predicate, starting from a ω-free basis, can be derived in $\vdash_{\not\omega}$, and afterwards use 1.3 (*iii*). We will use the same technique to prove a conservativity result.

Lemma 3.11 If M is in normal form and $B \vdash M{:}\sigma$ such that B and σ are ω-free, then $B \vdash_{\not\omega} M{:}\sigma$.

Proof: The proof is given by induction on the structure of terms in normal form.

i) $B \vdash x{:}\sigma$. Then by 1.3 (*v*) there are $x{:}\sigma_1, \ldots, x{:}\sigma_n$ in B such that $\sigma_1 \cap \cdots \cap \sigma_n \leq \sigma$. Then obviously $B \vdash_{\not\omega} x{:}\sigma$.

ii) $B \vdash \lambda x.M'{:}\sigma$. Then $\sigma \equiv (\rho_1 \to \mu_1) \cap \cdots \cap (\rho_n \to \mu_n)$ for some $n \geq 1$, and by 1.3 (*ii*) for every $i \in \{1, \ldots, n\}$: $B \cup \{x{:}\rho_i\} \vdash M'{:}\mu_i$. By induction for every $i \in \{1, \ldots, n\}$: $B \cup \{x{:}\rho_i\} \vdash_{\not\omega} M'{:}\mu_i$. So $B \vdash_{\not\omega} \lambda x.M'{:}\sigma$.

iii) $B \vdash xM_1 \ldots M_n{:}\sigma$. By 1.3 (*i*) there are $\tau_1, \ldots, \tau_n$ such that $B \vdash x{:}\tau_1 \to \cdots \to \tau_n \to \sigma$, $B \vdash M_1{:}\tau_1$, …, and $B \vdash M_n{:}\tau_n$.

By 1.3 (*v*) there are $x{:}\rho_1, \ldots, x{:}\rho_n$ in B such that $\rho_1 \cap \cdots \cap \rho_n \leq \tau_1 \to \cdots \to \tau_n \to \sigma$.

By 1.3 (*vi*) this implies $\rho_1 \cap \cdots \cap \rho_n = (\tau^1_1 \to \cdots \to \tau^1_n \to \sigma^1) \cap \cdots \cap (\tau^s_1 \to \cdots \to \tau^s_n \to \sigma^s) \cap \rho'$, for $\tau^j_1, \ldots, \tau^j_n, \sigma^j$ such that $\tau^j_i \geq \tau_i$ with $i \leq i \leq n$, $1 \leq j \leq s$ and $\sigma^1 \cap \cdots \cap \sigma^s \leq \sigma$.

Then by $(\leq)$ and $(\cap I)$ for every $i \in \{1, \ldots, n\}$ we have $B \vdash M_i{:}\tau^1_i \cap \cdots \cap \tau^s_i$.

Since each τ^j_i occurs in a statement in the basis, the induction hypothesis is applicable and for every $i \in \{1, \ldots, n\}$ we have $B \vdash_{\not\omega} M_i{:}\tau^1_i \cap \cdots \cap \tau^s_i$. Also

$$(\tau^1_1 \to \cdots \to \tau^1_n \to \sigma^1) \cap \cdots \cap (\tau^s_1 \to \cdots \to \tau^s_n \to \sigma^s) \leq$$
$$(\tau^1_1 \cap \cdots \cap \tau^s_1) \to \cdots \to (\tau^1_n \cap \cdots \cap \tau^s_n) \to (\sigma^1 \cap \cdots \cap \sigma^s),$$

so $B \vdash x{:}(\tau_1^1\cap\cdots\cap\tau_1^s)\rightarrow\cdots\rightarrow(\tau_n^1\cap\cdots\cap\tau_n^s)\rightarrow(\sigma^1\cap\cdots\cap\sigma^s)$ and by part (i)

$$B \vdash_{\not\omega} x{:}(\tau_1^1\cap\cdots\cap\tau_1^s)\rightarrow\cdots\rightarrow(\tau_n^1\cap\cdots\cap\tau_n^s)\rightarrow(\sigma^1\cap\cdots\cap\sigma^s).$$

But then by $(\rightarrow E)$ $B \vdash_{\not\omega} xM_1\ldots M_n{:}\sigma_1\cap\cdots\cap\sigma_n$ and by $(\leq)$ $B \vdash_{\not\omega} xM_1\ldots M_n{:}\sigma$. ■

Theorem 3.12 *If $B \vdash M{:}\sigma$ where ω does not occur in B and σ, then there is an M' such that M reduces to M' and $B \vdash_{\not\omega} M'{:}\sigma$.*

Proof: If $B \vdash M{:}\sigma$ where ω does not occur in B and σ, then by 1.3 (*iii*), M has a normal form M'. Then also $B \vdash M'{:}\sigma$. By the previous lemma we have that $B \vdash_{\not\omega} M'{:}\sigma$. ■

As remarked in the introduction, if we are interested in deriving types without ω occurrences, the type constant ω is only needed in the intersection type discipline to type sub-terms N of M that will be erased while reducing M. In fact, if there is a type ρ such that $B \vdash_{\not\omega} N{:}\rho$, then even for this N we would not need to use ω. Unfortunately there are lambda terms M that contain a sub-term N that must be typed with ω in $B \vdash M{:}\sigma$, even if ω does not occur in B and σ. We can even find strongly normalisable lambda terms that contain such a sub-term (see also the remark made after 3.20). So to prove Theorem 3.12 we have to go down all the way to the set of lambda terms in normal form, since only these do not contain sub-terms that will be erased.

Theorem 3.13 Conservativity: *If M is a λI-term and $B \vdash M{:}\sigma$ where ω does not occur in B and σ, then $B \vdash_{\not\omega} M{:}\sigma$.*

Proof: If $B \vdash M{:}\sigma$ and B, σ are ω-free, then by 1.3 (*iii*), M has a normal form M'. Then also $B \vdash M'{:}\sigma$. By Lemma 3.11 we have $B \vdash_{\not\omega} M'{:}\sigma$. Because M and M' are λI-terms, by Corollary 3.25 we obtain $B \vdash_{\not\omega} M{:}\sigma$. ■

3.3 The type assignment without ω is complete for the λI-calculus

In this subsection completeness of type assignment without ω for the λI-calculus is proved using the method of [2]. The notions of type interpretation as defined in 2.24 lead also in the case of the λI-calculus in a natural way to the following definitions for semantic satisfiability.

Definition 3.14 As in 2.25 we define $\models$ by: (where $\mathcal{M}$ is a λI-model, ξ a valuation and v a type interpretation)

i) $\mathcal{M},\xi,v \models M : \sigma \iff [\![M]\!]_\xi^{\mathcal{M}} \in v(\sigma)$.

ii) $\mathcal{M},\xi,v \models B \iff \mathcal{M},\xi,v \models x{:}\sigma$ for every $x{:}\sigma \in B$.

iii) *a*) $B \models M : \sigma \iff \forall \mathcal{M},\xi,v\ [\mathcal{M},\xi,v \models B \Rightarrow \mathcal{M},\xi,v \models M : \sigma]$.

b) $B \models_s M : \sigma \iff$
$\forall \mathcal{M},\xi$, simple type interpretations $v\ [\mathcal{M},\xi,v \models B \Rightarrow \mathcal{M},\xi,v \models M : \sigma]$.

c) $B \models_F M : \sigma \iff \forall \mathcal{M},\xi,F$ type interpretations $v\ [\mathcal{M},\xi,v \models B \Rightarrow \mathcal{M},\xi,v \models M : \sigma]$.

We consider only the simple type semantics, since $\vdash_{\not\omega}$ is not sound for all type interpretations. For example $\{y{:}((\varphi_1\cap\varphi_2)\rightarrow\varphi_3)\rightarrow\varphi_4,\ x{:}\varphi_1\rightarrow\varphi_3\} \vdash_{\not\omega} yx{:}\varphi_4$ but this is not semantically valid for all type interpretations.

Theorem 3.15 Soundness: *If $B \vdash_{\not\omega} M{:}\sigma$ then $B \models_s M : \sigma$.*

Proof: By induction on the structure of derivations. ■

Definition 3.16 *i*) We define a map $\nu_0 : \mathcal{T}_{\not\omega} \rightarrow \wp(\mathcal{F}_{-\omega})$ by $\nu_0(\sigma) = \{d \in \mathcal{F}_{-\omega} \mid \sigma \in d\}$.

ii) $\xi_B(x) = \{\sigma \in \mathcal{T}_{\not\omega} \mid B \vdash_{\not\omega} x{:}\sigma\}$

Theorem 3.17 *i) The map ν_0 is a simple type interpretation.*
ii) $B \vdash_{\not\omega} M:\sigma$ if and only if $B_{\xi_B} \vdash_{\not\omega} M:\sigma$.
iii) $\mathcal{F}_{-\omega}, \xi_B, \nu_0 \models_s B$.

Proof: i) Easy.
ii) Because for every x, $\xi_B(x)$ is an I-filter.
iii) $x{:}\sigma \in B \Rightarrow \sigma \in \{\tau \mid B_{\xi_B} \vdash_{\not\omega} x{:}\tau\} \Rightarrow \sigma \in [\![x]\!]_{\xi_B}$. So $[\![x]\!]_{\xi_B} \in \{d \in \mathcal{F}_{-\omega} \mid \sigma \in d\}$. ∎

Theorem 3.18 Completeness: *Let M be a λI-term and suppose ω does not occur in B and σ. If $B \models_s M:\sigma$ then $B \vdash_{\not\omega} M:\sigma$.*

Proof: $B \models_s M:\sigma \Rightarrow$ (3.14(*iii*),3.17(*i*) & 3.17(*iii*))
$\mathcal{F}_{-\omega}, \xi_B, \nu_0 \models_s M:\sigma \Rightarrow$ (3.14(*i*) & 3.17(*i*))
$[\![M]\!]_{\xi_B} \in \nu_0(\sigma) \Rightarrow$ (3.16(*i*))
$\sigma \in [\![M]\!]_{\xi_B} \Rightarrow$ (3.10)
$B_{\xi_B} \vdash_{\not\omega} M:\sigma \Rightarrow$ (3.17(*ii*))
$B \vdash_{\not\omega} M:\sigma$. ∎

3.4 Strong normalisation result for the system without ω

In this subsection we show that the set of lambda terms typeable by means of the derivation rules (∩I), (∩E), (→I) and (→E) is exactly the set of strongly normalisable terms.

The same result has been given in [6], [17] and [21]. However, the proof in [17] is too brief, the proof in [21] gives few details and the proof in [6] is not complete. In this subsection we present a complete and formal proof. In [22] a similar result is proved: $B \vdash_{\not\omega} M:\sigma \iff M$ is strongly normalisable.

To prove that each term typeable by the rules $(\cap I)$, $(\cap E)$, $(\rightarrow I)$ and $(\rightarrow E)$ is strongly normalisable, we will prove even more: we will show that if $B \vdash_{\not\omega} M:\sigma$ (i.e. using also rule $(\leq)$), then M is strongly normalisable. In [22] this result is given in Corollary 6.3 and is obtained from the theorem that the procedure PP' (as defined in [22], Section 6) finds a principal pair for all and nothing but the strongly normalisable terms. In this subsection we present a proof for the same result, different from the one given in [22]. The proof that all strongly normalisable terms are typeable in the system without ω and $(\leq)$ is given in Corollary 3.26.

Notice that an I-filter can be empty. A direct result of the main theorem of this subsection will be that $[\![\cdots]\!]$ as defined in 3.9 will map all unsolvable terms ('unsolvable' is in the λI-calculus exactly the same as 'not having a normal form', as well as that 'normalisable' and 'strongly normalisable' coincide) onto the empty filter.

Notice also that we no longer restrict ourselves to the λI-terms, but prove the statement for the full λK-calculus.

Fact 3.19 In the sequel, we will accept the following without proof:
i) If $xM_1 \ldots M_n$ and N are strongly normalisable, then so is $xM_1 \ldots M_n N$.
ii) If Mz is strongly normalisable (where z does not occur free in M), then so is M.
iii) If $M[N/x]$ and N are strongly normalisable, then so is $(\lambda x.M)N$. ∎

Lemma 3.20 If $B \vdash_{\not\omega} M[N/x]\vec{P}:\tau$ and $B \vdash_{\not\omega} N:\rho$, then $B \vdash_{\not\omega} (\lambda x.M)N\vec{P}:\tau$.

Proof: We can assume that x does not occur in B.
i) N occurs n times in $M[N/x]$, each time typed by, say, σ_i.

$B \vdash_{\not\omega} M[N/x]P_1 \cdots P_m : \tau \Rightarrow$
$\exists \tau_1, \ldots, t_m \; [B \vdash_{\not\omega} M[N/x] : \tau_1 \to \cdots \to \tau_m \to \tau \;\&\; \forall i \in \underline{m} \; [B \vdash_{\not\omega} P_i : \tau_i]] \Rightarrow$
$\exists \tau_1, \ldots, t_m, \sigma_1, \ldots, \sigma_n \; [B \cup \{x{:}\sigma_1 \cap \cdots \cap \sigma_n\} \vdash_{\not\omega} M : \tau_1 \to \cdots \to \tau_m \to \tau \;\&$
$\quad B \vdash_{\not\omega} N : \sigma_1 \cap \cdots \cap \sigma_n \;\&\; \forall i \in \underline{m} \; [B \vdash_{\not\omega} P_i : \tau_i]] \Rightarrow$
$\exists \tau_1, \ldots, t_m, \sigma_1, \ldots, \sigma_n \; [B \vdash_{\not\omega} \lambda x.M : (\sigma_1 \cap \cdots \cap \sigma_n) \to \tau \;\&$
$\quad B \vdash_{\not\omega} N : \sigma_1 \cap \cdots \cap \sigma_n \;\&\; \forall i \in \underline{m} \; [B \vdash_{\not\omega} P_i : \tau_i]] \Rightarrow$
$B \vdash_{\not\omega} (\lambda x.M)NP_1 \cdots P_m : \tau.$

ii) N does not occur in $M[N/x]$, so $x \notin fv(M)$.
$B \vdash_{\not\omega} MP_1 \cdots P_n : \tau \;\&\; B \vdash_{\not\omega} N : \rho \Rightarrow \qquad (x \notin fv(M))$
$\exists \tau_1, \ldots, t_m \; [B \vdash_{\not\omega} M : \tau_1 \to \cdots \to \tau_m \to \tau \;\&\; B \vdash_{\not\omega} N : \rho \;\&\; \forall i \in \underline{m} \; [B \vdash_{\not\omega} P_i : \tau_i]] \Rightarrow$
$\exists \tau_1, \ldots, t_m \; [B \cup \{x{:}\rho\} \vdash_{\not\omega} M : \tau_1 \to \cdots \to \tau_m \to \tau \;\&$
$\quad B \vdash_{\not\omega} N : \rho \;\&\; \forall i \in \underline{m} \; [B \vdash_{\not\omega} P_i : \tau_i]] \Rightarrow$
$\exists \tau_1, \ldots, t_m \; [B \vdash_{\not\omega} \lambda x.M : \rho \to \tau \;\&\; B \vdash_{\not\omega} N : \rho \;\&\; \forall i \in \underline{m} \; [B \vdash_{\not\omega} P_i : \tau_i]] \Rightarrow$
$B \vdash_{\not\omega} (\lambda x.M)NP_1 \cdots P_m : \tau.$ ∎

Notice that the condition $B \vdash_{\not\omega} N : \rho$ in the formulation of the lemma is essential. As a counter example take the two lambda terms $\lambda yz.(\lambda b.z)(yz)$ and $\lambda yz.z$. Notice that the first strongly reduces to the latter. We know that $\{z{:}\sigma, y{:}\tau\} \vdash_{\not\omega} z : \sigma$, but it is impossible to give a derivation for $(\lambda b.z)(yz) : \sigma$ from the same basis without using ω. This is caused by the fact that we can only type $(\lambda b.z)(yz)$ in the system without ω from a basis in which the predicate for y is an arrow type scheme. We can for example derive $\{z{:}\sigma, y{:}\sigma \to \tau\} \vdash_{\not\omega} (\lambda b.z)(yz) : \sigma$. We can therefore only state that we can derive $\vdash_{\not\omega} \lambda yz.(\lambda b.z)(yz) : (\sigma \to \tau) \to \sigma \to \sigma$ and also can derive $\vdash_{\not\omega} \lambda yz.z : \tau \to \sigma \to \sigma$ but that we are not able to give a derivation without ω for the statement $\lambda yz.(\lambda b.z)(yz) : \tau \to \sigma \to \sigma$. So the type assignment without ω is not closed for β-equality, but of course that is also not needed. We only want to be able to derive *a* type for each strongly normalisable term, no matter what basis or type are used.

Notice that for the λI-calculus the condition $B \vdash_{\not\omega} N : \rho$ is not needed. So the following is an immediate result:

Corollary 3.21 Let M and M' be λI-terms, such that M reduces to M' and there are B and σ such that $B \vdash_{\not\omega} M' : \sigma$. Then $B \vdash_{\not\omega} M : \sigma$. ∎

We will now show that all strongly normalisable terms are typeable in our system. The proof of the crucial lemma for this result as presented below (Lemma 3.23) goes by induction on the left-most outer-most reduction path. The following lemma formulates a subject expansion result for our system with respect to left-most outer-most reduction.

Definition 3.22 An occurrence of a redex $\mathrm{R} = (\lambda x.P)Q$ in a term M is called the *left-most outer-most redex of M* ($lor(M)$), if and only if:

i) there is no redex R' in M such that $\mathrm{R}' = \mathrm{C}[\mathrm{R}]$ (*outer-most*);

ii) there is no redex R' in M such that $M = \mathrm{C}_0[\mathrm{C}_1[\mathrm{R}']\mathrm{C}_2[\mathrm{R}]]$ (*left-most*).

$M \to_{lor} N$ is used to indicate that M reduces to N by contracting $lor(M)$.

Lemma 3.23 Let $M \to_{lor} N$, $lor(M) = (\lambda x.P)Q$, $B_1 \vdash_{\not\omega} N : \sigma$ with σ not an intersection, and $B_2 \vdash_{\not\omega} Q : \tau$, then there exists B_3, ρ such that $B_3 \leq B_1$, $\sigma \leq \rho$, and $B_3 \vdash_{\not\omega} M : \rho$.

Proof: By induction on the structure of terms.

$(M = VP_1 \cdots P_n)$: Then there are μ_j $(j \in \underline{n})$, such that $B_1 \vdash_{\not\omega} V' : \mu_1 \to \cdots \to \mu_n \to \sigma$ and $B_1 \vdash_{\not\omega} P_i : \mu_i$, for all $i \in \underline{n}$, and either:

a) V is a redex $(\lambda y.P)Q$, so $lor(M) = V$, and $V' \equiv P[Q/y]$. Let $B_3 = B_1 \cap B_2$ and $\rho = \sigma \cap \tau$, then by weakening and Lemma 3.20, $B_3 \vdash_{\not\omega} (\lambda y.P)Q : \mu_1 \to \cdots \to \mu_n \to \delta$.

b) $V \equiv z$ and there is an $i \in \underline{n}$ such that $lor(M) = lor(P_j)$, $N \equiv zP_1 \cdots P_i' \cdots P_n$, and $P_i \to_{lor} P_i'$. Then, by induction, there are B', μ_j' such that $\mu_j \leq \mu_j'$, and $B' \vdash_{\not\omega} P_j : \mu_j'$. Let $V' = z$, take $B_3 = B_1 \cap B' \cap z{:}\mu_1 \to \cdots \to \mu_j' \to \cdots \to \mu_n \to \phi$, and $\rho = \sigma \cap \rho$, then

$$B_3 \vdash_{\not\omega} z : \mu_1 \to \cdots \to \mu_j' \to \cdots \to \mu_n \to \delta.$$

In both cases, $B_3 \leq B_1$, $\rho \leq \sigma$, and $B_3 \vdash_{\not\omega} VP_1 \cdots P_n : \delta$.

$(M = \lambda y.M')$: If $M \to_{lor} N$, then $N = \lambda y.N'$ and $M' \to_{lor} N'$. Then there exists α and β such that $\sigma = \alpha \to \beta$, and $B_1, y{:}\alpha \vdash_{\not\omega} N' : \beta$. By induction, there exists $B' \leq B_1$, $\alpha' \leq \alpha$, and $\delta' \leq \delta$ such that $B', y{:}\alpha' \vdash_{\not\omega} M' : \beta'$. Then, by rule $(\to I)$, $B' \vdash_{\not\omega} \lambda y.M' : \alpha' \to \beta'$. Notice that $\alpha \to \beta \leq \alpha' \to \beta'$; take $B_3 = B'$ and $\rho = \alpha' \to \beta'$. ∎

Notice that the condition $B_2 \vdash_{\not\omega} N : \tau$ in the formulation of the lemma is essential. As a counter example take the two lambda terms $\lambda yz.(\lambda b.z)(yz)$ and $\lambda yz.z$. Notice that the first strongly reduces to the latter. We know that $\{z{:}\sigma, y{:}\tau\} \vdash_{\not\omega} z : \sigma$, but it is impossible to give a derivation for $(\lambda b.z)(yz) : \sigma$ from the same basis without using ω. This is caused by the fact that we can only type $(\lambda b.z)(yz)$ in the system without ω from a basis in which the predicate for y is an arrow type scheme. We can for example derive $\{z{:}\sigma, y{:}\sigma \to \tau\} \vdash_{\not\omega} (\lambda b.z)(yz) : \sigma$. We can therefore only state that we can derive $\vdash_{\not\omega} \lambda yz.(\lambda b.z)(yz) : (\sigma \to \tau) \to \sigma \to \sigma$ and also can derive $\vdash_{\not\omega} \lambda yz.z : \tau \to \sigma \to \sigma$ but that we are not able to give a derivation without ω for the statement $\lambda yz.(\lambda b.z)(yz) : \tau \to \sigma \to \sigma$. So the type assignment without ω is not closed for β-equality, but of course that is also not needed. We only want to be able to derive *a* type for each strongly normalisable term, no matter what basis or type are used.

Theorem 3.24 *If M is strongly normalisable, then there are σ and B such that $B \vdash_{\not\omega} M : \sigma$ and in the derivation the derivation rule $(\leq)$ is not used.*

Proof: By induction on the maximum of the lengths of reduction sequences for a strongly normalisable term to its normal form (denoted by $\#(M)$).

i) If $\#(M) = 0$, then M is in normal form, and by Lemma 3.11, there exist B and σ such that $B \vdash_{\not\omega} M : \sigma$.

ii) If $\#(M) \geq 1$, so M contains a redex, then let $M \to_{lor} N$ by contracting $(\lambda x.P)Q$. Then $\#(N) < \#(M)$, and $\#(Q) < \#(M)$ (since Q is a proper subterm of a redex in M), so by induction $B \vdash_{\not\omega} N : \sigma$ and $B' \vdash_{\not\omega} Q : \tau$, for some B, B', σ, and τ. Then, by Lemma 3.23, there exist B_1, ρ such that $B_1 \vdash_{\not\omega} M : \rho$. ∎

Notice that for the λI-calculus the condition $B \vdash_{\not\omega} N : \rho$ is not needed. So the following is an immediate result:

Corollary 3.25 Let M and M' be λI-terms, such that M reduces to M' and there are B and σ such that $B \vdash_{\not\omega} M' : \sigma$. Then $B \vdash_{\not\omega} M : \sigma$. ∎

Lemma 3.20 is also essentially the proof for the statement that each strongly normalisable term can be typed in the system $\vdash_{\not\omega}$.

Corollary 3.26 If M is strongly normalisable, then there are σ and B such that $B \vdash_{\not\omega} M : \sigma$ and in the derivation the derivation rule $(\leq)$ is not used.

Proof: If M is strongly normalizable, then by using the inside-out reduction strategy (see [1], Definition 14.2.11) the normal form of M will be reached. This strategy has the special property that a redex $(\lambda x.P)Q$ can only be contracted if Q is in normal form. The proof is completed by induction on the inside out reduction path, using Lemma 3.20 and Theorem 2.13. ■

In order to prove that each term typeable in $\vdash$ is strongly normalisable we introduce a notion of computability. From now on, we will abbreviate 'M is strongly normalisable' by $SN(M)$.

Definition 3.27 (CF. [21]) $Comp(B,M,\rho)$ is inductively defined by:

i) $Comp(B,M,\varphi) \iff B \vdash_{\omega} M:\varphi \;\&\; SN(M)$.

ii) $Comp(B,M,\sigma\rightarrow\tau) \iff (Comp(B',N,\sigma) \Rightarrow Comp(B\cap B',MN,\tau))$.

iii) $Comp(B,M,\sigma\cap\tau) \iff (Comp(B,M,\sigma) \;\&\; Comp(B,M,\tau))$.

Lemma 3.28 Let σ and τ be such that $\sigma \leq \tau$. Then $Comp(B,M,\sigma)$ implies $Comp(B,M,\tau)$.

Proof: By straightforward induction on the definition of $\leq$. ■

Theorem 3.29 *i) If $B \vdash_{\omega} xM_1 \ldots M_n:\rho$ and $SN(xM_1 \ldots M_n)$, then $Comp(B,xM_1 \ldots M_n,\rho)$.*
ii) If $Comp(B,M,\rho)$, then $B \vdash_{\omega} M:\rho$ and $SN(M)$.

Proof: Simultaneously by induction on the structure of types. The only interesting case is when $\rho \equiv \sigma\rightarrow\tau$, the other cases are dealt with by induction.

i) $B \vdash_{\omega} xM_1 \ldots M_n:\sigma\rightarrow\tau \;\&\; SN(xM_1 \ldots M_n) \Rightarrow$
$(Comp(B',N,\sigma) \Rightarrow$
$\quad B \vdash_{\omega} xM_1 \ldots M_n:\sigma\rightarrow\tau \;\&\; SN(xM_1 \ldots M_n) \;\&\; B' \vdash_{\omega} N:\sigma \;\&\; SN(N)) \Rightarrow$
$(Comp(B',N,\sigma) \Rightarrow B\cap B' \vdash_{\omega} xM_1 \ldots M_nN:\tau \;\&\; SN(xM_1 \ldots M_n)) \Rightarrow$
$(Comp(B',N,\sigma) \Rightarrow Comp(B\cap B',xM_1 \ldots M_nN,\tau)) \Rightarrow$ (3.27 (*ii*))
$Comp(B,xM_1 \ldots M_n,\sigma\rightarrow\tau)$.

ii) $Comp(B,M,\sigma\rightarrow\tau) \;\&\; z \notin fv(M) \Rightarrow$
$Comp(B,M,\sigma\rightarrow\tau) \;\&\; Comp(\{z{:}\sigma\},z,\sigma) \;\&\; z \notin fv(M) \Rightarrow$ (3.27 (*ii*))
$Comp(B\cup\{z{:}\sigma\},Mz,\tau) \;\&\; z \notin fv(M) \Rightarrow$
$B\cup\{z{:}\sigma\} \vdash_{\omega} Mz:\tau \;\&\; SN(Mz) \;\&\; z \notin fv(M) \Rightarrow$ (3.2 (*iv*))
$B \vdash_{\omega} M:\sigma\rightarrow\tau \;\&\; SN(M)$. ■

Lemma 3.30 $Comp(B,M[N/x]\vec{P},\sigma) \;\&\; Comp(B,N,\rho) \Rightarrow Comp(B,(\lambda x.M)N\vec{P},\sigma)$.

Proof: By induction on the structure of types. We only consider the case that σ is a type variable:

$Comp(B,M[N/x]\vec{P},\varphi) \;\&\; Comp(B,N,\rho) \Rightarrow$ (3.29 (*ii*))
$B \vdash_{\omega} M[N/x]\vec{P}:\varphi \;\&\; SN(M[N/x]\vec{P}) \;\&\; B \vdash_{\omega} N:\rho \;\&\; SN(N) \Rightarrow$ (3.20)
$B \vdash_{\omega} (\lambda x.M)N\vec{P}:\varphi \;\&\; SN((\lambda x.M)N\vec{P}) \Rightarrow$ (3.27 (*i*))
$Comp(B,(\lambda x.M)N\vec{P},\varphi)$. ■

Theorem 3.31 *If $B = \{x_1{:}\mu_1,\ldots,x_n{:}\mu_n\}$ and $Comp(B',N_i,\mu_i)$ and $B \vdash_{\omega} M:\sigma$, then $Comp(B',M[\overrightarrow{N_i/x_i}],\sigma)$.*

Proof: By induction on the structure of derivations. We will only show the non-trivial parts, and assume the substitutions to take place in parellel.

$(\rightarrow I)$: Then $M \equiv \lambda y.M'$, $\sigma = \rho\rightarrow\tau$, and $B\cap\{y{:}\rho\} \vdash_{\omega} M':\tau$.
$B = \{x_1{:}\mu_1,\ldots,x_n{:}\mu_n\} \;\&\; Comp(B',N_i,\mu_i) \;\&\; B\cap\{y{:}\rho\} \vdash_{\omega} M':\tau \Rightarrow$
$(Comp(B',N,\rho) \Rightarrow Comp(B',M'[\overrightarrow{N_i/x_i},N/y],\tau)) \Rightarrow$ (3.30)
$(Comp(B',N,\rho) \Rightarrow Comp(B',(\lambda y.M'[\overrightarrow{N_i/x_i}])N,\tau)) \Rightarrow$ (3.27 (*ii*))
$Comp(B',(\lambda y.M')[\overrightarrow{N_i/x_i}],\rho\rightarrow\tau)$.

$(\to E)$: Then $M \equiv M_1M_2$, $B \vdash_{\omega} M_1 : \rho \to \tau$ and $B \vdash_{\omega} M_2 : \rho$.

$$B = \{x_1{:}\mu_1,\ldots,x_n{:}\mu_n\} \;\&\; Comp(B',N_i,\mu_i) \;\&\; B \vdash_{\omega} M_1 : \rho \to \tau \;\&\; B \vdash_{\omega} M_2 : \rho \Rightarrow$$
$$Comp(B',M_1[\overrightarrow{N_i/x_i}],\rho \to \tau) \;\&\; Comp(B',M_2[\overrightarrow{N_i/x_i}],\rho) \Rightarrow \qquad (3.27\,(ii))$$
$$Comp(B',(M_1M_2)[\overrightarrow{N_i/x_i}],\tau).$$ ■

Theorem 3.32 *If $B \vdash_{\omega} M : \sigma$, then $SN(M)$.*

Proof:
$$B \vdash_{\omega} M : \sigma \Rightarrow \qquad (3.31)$$
$$Comp(B,M,\sigma) \Rightarrow \quad (3.29\,(ii))$$
$$SN(M).$$ ■

We can now prove the main theorem of this subsection.

Theorem 3.33 $\{M \mid M$ *is typeable by means of the derivation rules* $(\cap I)$, $(\cap E)$, $(\to I)$ *and* $(\to E)\} =$ $\{M \mid M$ *is strongly normalisable*$\}$.

Proof: $(\subseteq)$: If M is typeable by means of the derivation rules $(\cap I)$, $(\cap E)$, $(\to I)$ and $(\to E)$, then certainly $B \vdash_{\omega} M : \sigma$. Then by Theorem 3.32, M is strongly normalisable.

$(\supseteq)$: If M is strongly normalisable, then by Corollary 3.26 there are σ and B such that $B \vdash_{\omega} M : \sigma$ and in the derivation the derivation rule $(\leq)$ is not used. ■

Acknowledgements

I would like to thank the people of the Department of Computer Science of the university of Turin, Italy, for their hospitality and support during my stay in the first half of 1988. My very special thanks are for Mariangiola Dezani, for encouragement, support and many helpful suggestions. I would also like to thank one of the referees who gave a lot of fruitful remarks.

References

[1] Barendregt H. *The Lambda Calculus: its Syntax and Semantics*. North-Holland, Amsterdam, revised edition, 1984.

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

[3] Coppo M. and M. Dezani-Ciancaglini. An Extension of the Basic Functionality Theory for the λ-Calculus. *Notre Dame, Journal of Formal Logic*, 21(4):685–693, 1980.

[4] Coppo M., M. Dezani-Ciancaglini, F. Honsell, and G. Longo. Extended type structures and filter λ-models. In G. Lolli, G. Longo, and A. Marcja, editors, *Logic Colloquium 82*, pages 241–262, Amsterdam, 1984. North-Holland.

[5] Coppo M., M. Dezani-Ciancaglini, and B. Venneri. Principal type schemes and λ-calculus semantics. In J. R. Hindley and J. P. Seldin, editors, *To H. B. Curry, Essays in combinatory logic, lambda-calculus and formalism*, pages 535–560. Academic press, New York, 1980.

[6] Coppo M., M. Dezani-Ciancaglini, and B. Venneri. Functional characters of solvable terms. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 27:45–58, 1981.

[7] Coppo M., M. Dezani-Ciancaglini, and M. Zacchi. Type Theories, Normal Forms and D_∞-Lambda-Models. *Information and Computation*, 72(2):85–116, 1987.

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

[9] Curry H.B. and R. Feys. *Combinatory Logic*. volume 1. North-Holland, Amsterdam, 1958.

[10] Dezani-Ciancaglini M., F. Honsell, and S. Ronchi della Rocca. Models for theories of functions strictly depending on all their arguments. Stanford ASL Summer Meeting '85. *Journal of Symbolic Logic*, 51 (2):845–846, 1986. abstract.

[11] Dezani-Ciancaglini M. and I. Margaria. F-semantics for intersection type discipline. In G. R. Kahn, D. B. Macqueen, and G. Plotkin, editors, *Sematics of data types. International symposium Sophia - Antipolis, France*, volume 173 of *Lecture Notes in Computer Science*, pages 279–300. Springer-Verlag, 1984.

[12] Dezani-Ciancaglini M. and I. Margaria. A characterisation of F-complete type assignments. *Theoretical Computer Science*, 45:121–157, 1986.

[13] Hindley J.R. The principal type scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60, 1969.

[14] Hindley J.R. The simple semantics for Coppo-Dezani-Sallé type assignment. In M. Dezani and U. Montanari, editors, *International symposium on programming*, volume 137 of *Lecture Notes in Computer Science*, pages 212–226. Springer-Verlag, 1982.

[15] Hindley J.R. The Completeness Theorem for Typing λ-terms. *Theoretical Computer Science*, 22(1):1–17, 1983.

[16] Honsell F. and S. Ronchi della Rocca. Models for theories of functions strictly depending on all their arguments. Internal report, Department of Computer Science, Turin, Italy, 1984.

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

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

[19] Mitchell J.C. Polymorphic Type Inference and Containment. *Information and Computation*, 76:211–249, 1988.

[20] Plotkin G.D. and M.B. Smyth. The category-theoretic solution of recursive domain equations. DAI Research Report 60, University of Edinburgh, Scotland, 1978.

[21] Pottinger G. A type assignment for the strongly normalisable λ-terms. In J. R. Hindley and J. P. Seldin, editors, *To H. B. Curry, Essays in combinatory logic, lambda-calculus and formalism*, pages 561–577. Academic press, New York, 1980.

[22] Ronchi della Rocca S. Principal type scheme and unification for intersection type discipline. *Theoretical Computer Science*, 59:181–209, 1988.

[23] Ronchi della Rocca S. and B. Venneri. Principal type schemes for an extended type theory. *Theoretical Computer Science*, 28:151–169, 1984.

[24] Sallé P. Une extension de la théorie des types. In G. Ausiello and C. Böhm, editors, *Automata, languages and programming. Fifth Colloquium, Udine, Italy*, volume 62 of *Lecture Notes in Computer Science*, pages 398–410. Springer-Verlag, 1978.

[25] Turner D.A. Miranda: A non-strict functional language with polymorphic types. In *Proceedings of the conference on Functional Programming Languages and Computer Architecture*, volume 201 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 1985.

Appendix A On the correction

The error that is corrected here lies in the proof of subject expansion result in Lemma 3.20. This lemma was used in two others: that *1) all strongly normalisable terms are typeable (Corollary 3.22 in original paper, Theorem 3.24 here),* and that *2) the computability predicate is closed for subject expansion (Lemma 3.26 vs Lemma 3.30 here).*

The original lemma was formulated as

$$\text{If } B \vdash_{\omega} C[M[x := N]] : \sigma \text{ and } B \vdash_{\omega} N : \tau, \text{ then } B \vdash_{\omega} C[(\lambda x.M)N] : \sigma.$$

However, its proof was incorrect, since the basis for the first term $C[M[x := N]]$ changes in the proof. Reformulating this as

$$\text{If } B_1 \vdash_{\omega} C[M[x := N]] : \sigma \text{ and } B_2 \vdash_{\omega} N : \tau, \text{ then } B_1 \cap B_2 \vdash_{\omega} C[(\lambda x.M)N] : \sigma$$

(which seems to be used by various authors) gives no improvement; the problem is that a free variable in N might be bound in the context, which implies that the derived type might change (get bigger in the sense of $\leq$), and that the $B_1 \cap B_2$ is not a correct basis for $C[(\lambda x.M)N]$, since it contains types for bound variables. This suggests the property

$$\begin{array}{c}\text{If } B_1 \vdash_{\omega} C[M[x := N]] : \sigma \text{ and } B_2 \vdash_{\omega} N : \tau, \\ \text{then there exists } B_3 \leq B_1, \rho \geq \sigma \text{ such that } B_3 \vdash_{\omega} C[(\lambda x.M)N] : \rho\end{array}$$

but this is also not achievable, since the use of $\leq$ in the derivation for P in MP (where $MP \rightarrow MQ$) forces a $\geq$ step on M which is not always achievable, which throws the proof for the case of application out of kilter irreparably. Notice that this does not imply that the term $C[(\lambda x.M)N]$ is not typeable *at all*; it might be possible that the following holds:

$$\begin{array}{c}\text{If } B_1 \vdash_{\omega} C[M[x := N]] : \sigma \text{ and } B_2 \vdash_{\omega} N : \tau, \\ \text{then there exists } B_3, \rho \text{ such that } B_3 \vdash_{\omega} C[(\lambda x.M)N] : \rho\end{array}$$

but it is not clear how to prove this, since it specifies no relation between bases and types.

It is, however, possible to show a subject expansion result for *applicative* contexts, where the redex occurs at the left. Since those are exactly the contexts considered in the proof of Lemma 3.30, here Lemma 3.20 is reformulated using those contexts only.

A more general subject expansion result is only really provable for left-most outermost (l.o.) reduction as first shown by Betti Venneri, as is added to this version of the paper (see Definition 3.22, and Lemma 3.23). The correct subject expansion result (with respect to l.o., Lemma 3.23) is now the one used for Theorem 3.24, which now uses induction on the length of the l.o. reduction path.