Complete restrictions of the intersection type discipline
(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 (≤) and it is based on a set of strict types. We
will show that these together give rise to a strict filter λ-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 (≤) 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 normalizable.

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.
Curry’s type assignment system has however drawbacks. It is not capable of assigning a type to
λx.xx, and although the lambda terms λcd.d and (λxyz.xz(yz))(λab.a) are β-equal, the principal

† 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.).
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 ‘→’ it contains a type constructor ‘∩’ 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 \( B \vdash M[x := N]:\sigma \) 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 \( M[x := N] \) and there is no other type \( \rho \) such that \( B \vdash N:\rho \). The intersection types are used for the cases that \( N \) occurs more than once in \( M[x := N] \) and these occurrences were typed in the derivation for \( B \vdash M[x := N]:\sigma \) 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 ‘≤’ on types as well as adding the type assignment rule (≤), and a more general form of the rules concerning intersection. The rule (≤) 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 ≤ (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 λ-model \( \mathcal{F} \). Using this model, completeness is proved. Other interesting use of filter λ-models can be found in [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 (≤)) 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 \( \sigma \) and \( M =_\beta N \), then also \( N \) is typeable by \( \sigma \). 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 (∩I), (∩E) and (≤), 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 ≤ relation and the derivation rule (≤) are no longer present. The elimination of ≤ 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 strict filter λ-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 λ-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

1 Unlike in [2], we will use the notation ‘\( M:\sigma \)’ for the statement ‘\( \sigma \) is a type for \( M \).’
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 $F_S$.

The second restriction presented is a type assignment system without $\omega$. It is not difficult to see that, while building a derivation $B \vdash M:\sigma$ (where $\omega$ does not occur in $\sigma$ and $B$) for a lambda term $M$ that has a normal form, the type $\omega$ 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 $\lambda$I-calculus, the type $\omega$ is not really needed for terms that have a normal form. The type assignment system without $\omega$ yields a $\lambda$I-model and turns out to be complete for the $\lambda$I-calculus with respect to the simple type semantics. The set of terms typeable by this system is just the set of all strongly normalizable 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 $\omega$. 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:

- All type variables $\varphi_0, \varphi_1, \ldots \in \mathcal{T}$.
- $\omega \in \mathcal{T}$.
- If $\sigma$ 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:

- $\sigma \leq \sigma$.
- $\sigma \leq \omega$.
- $\omega \leq \omega \rightarrow \omega$.
- $\sigma \cap \tau \leq \sigma$.
- $\sigma \cap \tau \leq \tau$.
- $(\sigma \rightarrow \tau) \cap (\sigma \rightarrow \rho) \leq \sigma \rightarrow (\tau \cap \rho)$.
- $\sigma \leq \tau \leq \rho \Rightarrow \sigma \leq \rho$.
- $\sigma \leq \tau \& \sigma \leq \rho \Rightarrow \sigma \leq \tau \cap \rho$.
- $\rho \leq \sigma \& \tau \leq \mu \Rightarrow \sigma \rightarrow \tau \leq \rho \rightarrow \mu$. 

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 \( \sigma \in T \). \( M \) is the subject and \( \sigma \) the predicate of \( M:\sigma \).

v) A basis is a set of statements with only variables (not necessarily distinct) as subjects. \( T \) 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.1.): \( \sigma \leq \tau \& \mu \leq \rho \Rightarrow \sigma \cap \mu \leq \tau \cap \rho \).
(h.2.): \( \sigma \leq \sigma \cap \sigma \).

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

Throughout this paper, the symbol \( \phi \) will be a type variable and the symbols \( \mu, \nu, \eta, \rho, \sigma \) and \( \tau \) will range over types.

**Definition 1.2**

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

\[
\begin{align*}
\text{(\rightarrow I):} & \\
& \frac{[x:\sigma]}{\lambda x.M:\sigma \rightarrow \tau} \quad (a) \\
\text{(\rightarrow E):} & \\
& \frac{M:\sigma \rightarrow \tau \quad N:\sigma}{MN:\tau} \\
\text{\text{(\cap I):} } & \\
& \frac{M:\sigma \quad M:\tau}{M:\sigma \cap \tau} \\
\text{\text{(\cap E):} } & \\
& \frac{M:\sigma \cap \tau \quad M:\sigma \cap \tau}{M:\tau} \\
\text{(\leq): } & \\
& \frac{M:\sigma \quad \sigma \leq \tau}{M:\tau} \\
\text{\text{(\omega):} } & \\
& \frac{M:\omega}{M:\omega}
\end{align*}
\]

\((a):\) 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:

i) The set of types derivable for a lambda term is a filter, i.e. a set closed under intersection and right closed for \( \leq \).

ii) The interpretation of a lambda term by the set of types derivable for it, gives a filter \( \lambda \)-model \( F \). Using this model, completeness is proved.

iii) The set of normalizable terms can be characterized in the following way:

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

iv) The set of terms having a head normal form can be characterized 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: \( B \vdash MN:\tau \Rightarrow \exists \sigma \in T \ [ B \vdash M:\sigma \rightarrow \tau \& B \vdash N:\sigma ] \).
ii) [2].2.8.iii: \( B \vdash \lambda x. M : \sigma \rightarrow \tau \iff B \backslash x^2 \cup \{ x : \sigma \} \vdash M : \tau \).  

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

iv) [2].4.13.ii: \( \exists B, \sigma \ [ B \vdash M : \sigma \land B, \sigma \omega \text{-free} ] \iff M \) 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: \( \rho \leq (\tau_1 \cap \cdots \cap \tau_n) \rightarrow \sigma \Rightarrow \rho = (\tau_1^1 \rightarrow \cdots \rightarrow \tau_n^1 \rightarrow \sigma_1) \cap \cdots \cap (\tau_1^n \rightarrow \cdots \rightarrow \tau_n^n \rightarrow \sigma) \cap \rho \), for some \( \tau_1^1, \ldots, \tau_n^s, \sigma_i, \rho \) such that \( \tau_i^j \geq \tau_i \) with \( 1 \leq i \leq n, 1 \leq j \leq s \) and \( \sigma_1 \cap \cdots \cap \sigma_s \leq 1 \).

2 The system without derivation rule (≤)

In this section we will give an extension (without the (≤)-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 2.2.1) and that type assignment in this system is complete (subsection 2.2.3). 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 (≤) is used, the same statement can be derived using a derivation in which the derivation rule (≤) is at the most only used at the very end of the derivation (subsection 2.2.2).

2.1 Strict derivations

In this subsection we present a restricted version of the intersection type assignment system, in which the derivation rule (≤) 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 (≤).

Strict types and strict derivations are closely related. Strict derivations are syntax directed and yield strict types. The type constant \( \omega \) 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 \( \sigma, \sigma_1, \ldots, \sigma_n, \tau \in \mathcal{T}_S \), then \( \sigma \rightarrow \tau, \omega \rightarrow \tau, (\sigma_1 \cap \cdots \cap \sigma_n) \rightarrow \tau \in \mathcal{T}_S \).

d) \( \mathcal{T}_S \) 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) \( \sigma \cap \tau \leq_S \sigma \land \sigma \cap \tau \leq_S \tau \).

d) \( \sigma \leq_S \tau \leq_S \rho \Rightarrow \sigma \leq_S \rho \).

e) \( \sigma \leq_S \rho \land \sigma \leq_S \tau \Rightarrow \sigma \leq_S \rho \cap \tau \).

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 \( \sigma \) the predicate of \( M : \sigma \).

\(^2\) \( B \backslash x \) is the basis obtained from \( B \) by erasing the statements that have \( x \) as subject.
vi) A basis is a set of statements with only variables as subjects. \( \mathcal{T}_S \) 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 \( \sigma \) is an intersection type scheme in which \( \tau \) occurs. Notice moreover that if \( \sigma \sim_S \tau \), then either \( \sigma = \tau \) or \( \sigma \) is an intersection type scheme and \( \tau \) can be obtained from \( \sigma \) 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 \( \sigma \) in rule \((\rightarrow I)):

\[
\begin{align*}
\frac{}{\lambda x.M : \sigma \rightarrow \tau} \quad (\rightarrow I) \\
\frac{[x;\sigma]}{M : \tau} \quad (\rightarrow E) \\
\frac{x : \sigma_1 \cap \cdots \cap \sigma_n}{x : \sigma_j} \quad (\cap E) \\
\frac{N : \sigma_1 \cdots N : \sigma_n}{MN : \tau} \quad (\cap I) \\
\frac{M : \sigma \rightarrow \tau}{M : \omega \rightarrow \tau} \quad (b)
\end{align*}
\]

(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 \((\omega)\) 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 \( \omega \), 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:
Lemma 2.5 For strict filters the following properties hold:

\[ \sigma \Rightarrow \forall \tau \in T_S \{ (B \cup \{x;\sigma\}) \downarrow \Rightarrow B \cup \{x;\sigma\} \downarrow \} \Rightarrow \forall \rho \in T_S \{ B \downarrow \lambda x.M;\rho \downarrow B \downarrow \lambda x.N;\rho \} \].

Proof: Easy. \[ \blacksquare \]

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 \( T_S \) is called a strict filter if and only if:

\begin{itemize}
  \item[a)] \( \omega \in d \).
  \item[b)] \( \sigma, \tau \in d \Rightarrow \sigma \land \tau \in d \).
  \item[c)] \( \tau \in d \land \tau \leq \sigma \Rightarrow \sigma \in d \).
\end{itemize}

ii) If \( V \) is a subset of \( T_S \), then \( \uparrow V \) is the smallest strict filter that contains \( V \), and \( \uparrow \sigma = \uparrow \{ \sigma \} \).

If no confusion is possible, we will omit the subscript on \( \uparrow \).

iii) \( \mathcal{F}_S = \{ d \subseteq T_S \mid d \) is a strict filter \}. We define application on \( \mathcal{F}_S \), \( \cdot : \mathcal{F}_S \times \mathcal{F}_S \rightarrow \mathcal{F}_S \) by:

\[ d \cdot e = \uparrow \{ \tau \mid \exists \sigma \in e \{ \sigma \} \} \rightarrow \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 \( \sigma \rightarrow \tau \in T_S \), \( \tau \) is strict.

\( \prec \mathcal{F}_S, \subseteq \succ \) 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’.** \( \sigma \land \tau \in d \Rightarrow \sigma \in d \) \& \( \tau \in d \).

Notice that a strict filter generated by a finite number of types is finite. Let for example \( \sigma \) be a strict type, then \( \uparrow \sigma = \{ \sigma, \omega \} \) (where by \( \sim_S \) we identify \( \sigma \) and \( \sigma \land \sigma \)). If \( \sigma \) is an intersection of strict types, \( \sigma = \sigma_1 \cap \cdots \cap \sigma_n \), then \( \uparrow \sigma \) 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 \( \mathcal{F}_S \) contains also infinite elements.

**Lemma 2.5** For strict filters the following properties hold:

\begin{itemize}
  \item[i)] \( \sigma \neq \omega \land \sigma \in \uparrow V \land V \subseteq \mathcal{T}_S \Leftrightarrow \exists \sigma_1, \ldots, \sigma_n \{ \sigma = \sigma_1 \cap \cdots \cap \sigma_n \land \forall i \in \{1, \ldots, n\} \{ \sigma_i \in V \} \} \).
  \item[ii)] \( \sigma \in \mathcal{T}_S \land \sigma \in \uparrow V \land V \subseteq \mathcal{T}_S \Rightarrow \sigma \in V \).
  \item[iii)] \( \sigma \in \uparrow \tau \Leftrightarrow \tau \leq \sigma \).
  \item[iv)] \( \sigma \in \uparrow \{ \tau \mid B \downarrow \mathcal{M};\tau \} \Leftrightarrow \sigma \in \{ \tau \mid B \downarrow \mathcal{M};\tau \} \).
  \item[v)] \( \{x;\sigma\} \downarrow x;\tau \Leftrightarrow \sigma \leq \tau \).
\end{itemize}

Proof: Easy. \[ \blacksquare \]

**Theorem 2.6**

i) If \( B \downarrow \mathcal{M};\sigma \land \sigma \leq \tau \), then \( B \downarrow \mathcal{M};\tau \).

ii) \( \{ \sigma \in \mathcal{T}_S \mid B \downarrow \mathcal{M};\sigma \} \subseteq \mathcal{F}_S \).

Proof: i) By induction on \( \leq \mathcal{S} \).
ii) By 2.5.(iv).

**Definition 2.7** We define $F : F_S \to [F_S \to F_S]$ and $G : [F_S \to F_S] \to F_S$ by:

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

ii) $G \ f = \{ \sigma \to \tau \in T_S \mid \tau \in f(\uparrow \sigma) \}$.

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

**Theorem 2.8** $<F_S, \cdot >$ 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 $F \circ G = id_{[F_S \to F_S]}$.

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

$(2.5.(iii))$

$\{ \mu \mid \exists \rho \in d \ [ \mu \in f(\uparrow \rho) \} = f(d)$.

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

**Definition 2.9** Let $\xi$ be a valuation of term variables in $F_S$.

i) $[[M]]_\xi$, the interpretation of terms in $F_S$ via $\xi$ is inductively defined by:

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

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

c) $[[\lambda x.M]]_\xi = G(\lambda x \in F_S. [[M]]_\xi{(v/x)})$.

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

**Theorem 2.10** For all $M, \xi$, $\{ \sigma \in T_S \mid B_\xi \vdash_S M: \sigma \}$.

**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) $[[MN]]_\xi = \{ \tau \mid \exists \sigma \ [ B_\xi \vdash_S N: \sigma \ & B_\xi \vdash_S M: \sigma \to \tau \} = (2.3.(ii))$ $\&$ $(iii)$

$\{ \tau \mid B_\xi \vdash_S MN: \tau \}$

(iii) $[[\lambda x.M]]_\xi = \{ \sigma \to \tau \mid B_\xi(\uparrow \sigma/x) \vdash_S M: \tau \} = (2.3.(iii))$

$\{ \sigma \to \tau \mid B_\xi(\uparrow \sigma/x) \vdash_S M: \tau \} = (2.3.(iii))$

$\{ \rho \mid B_\xi \vdash_S \lambda x. M: \rho \}$

**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 $\vdash_S$:

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

**Proof:** Since $F_S$ is a $\lambda$-model, we know that if $M =_\beta N$, then $[[M]]_\xi = [[N]]_\xi$; so $\{ \sigma \in T_S \mid B_\xi \vdash_S M: \sigma \} = $
Proof: By induction to the structure of lambda terms in normal form.

\[\{\sigma \in T_S \mid B\vDash_S N;\sigma\}.\]

Notice that because of the way in which \(\vDash_S\) is defined, corollary 2.11 also holds if \(\vDash_S\) is replaced by \(\vDash_s\).

Examples: By using 2.3 and 2.5 we can show the following:

i) If \(M\) is a closed term, then for all \(\xi\), \([M]_{\xi} = \{\sigma \in T_S \mid \vDash_S M;\sigma\}\). So for closed terms we can omit the subscript \(\xi\).

ii) \([\lambda xy.\! xy]\) = \(\{\rho \rightarrow \sigma \rightarrow \tau \mid \exists \sigma' \{ \rho \leq_S \sigma' \rightarrow \tau \ & \sigma \leq_S \sigma' \}\}\).

iii) \([\lambda x.\! x]\) = \(\{\sigma \rightarrow \tau \mid \sigma \leq \tau\}\).

iv) \([\lambda x.\! x;\! y]\) = \(\xi(y)\).

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

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

Theorem 2.12 If \(M\) is in normal form, then there are \(B\) and \(\sigma\) such that \(B \vDash_S M;\sigma\), and in this derivation \(\omega\) does not occur.

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

i) \(M \equiv x\). Take \(\sigma\) strict, such that \(\omega\) does not occur in \(\sigma\). Then \(\{x;\sigma\} \vDash_S x;\sigma\).

ii) \(M \equiv \lambda x.M'\), with \(M'\) in normal form. By induction there are \(B\) and \(\tau\) such that \(B \vDash_S M';\tau\) and \(\omega\) does not occur in this derivation. In order to perform the \((\rightarrow)\)-step, \(B\) must contain (whether or not \(x\) is free in \(M'\)) a statement with subject \(x\) and predicate, say, \(\sigma\). But then of course \(B'\! \backslash x \vDash_S \lambda x.M';\sigma \rightarrow \tau\) and \(\omega\) 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 \(i \in \{1, \ldots, n\}\), \(B_i \vDash_M M_i;\sigma_i\) and \(\omega\) does not occur in these derivations. Take \(\tau\) strict, such that \(\omega\) does not occur in \(\tau\), and \(B = \bigcup_{i \in \{1, \ldots, n\}} B_i \cup M;\sigma_1 \rightarrow \ldots \rightarrow \sigma_n \rightarrow \tau\). Then \(B \vDash_S xM_1 \ldots M_n;\tau\) and in this derivation \(\omega\) does not occur.

Theorem 2.13 If \(M\) is in head normal form, then there are \(B\) and \(\sigma\) such that \(B \vDash_S M;\sigma\).

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

i) \(M \equiv x\). Take \(\sigma\) 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 \(\tau\) such that \(B \vDash_S M';\tau\). As in the previous theorem, \(B\) must contain a statement with subject \(x\) and predicate, say, \(\sigma\). But then of course \(B'\! \backslash x \vDash_S \lambda x.M';\sigma \rightarrow \tau\).

iii) \(M \equiv xM_1 \ldots M_n\), with \(M_1, \ldots, M_n\) lambda terms. Take \(\tau\) strict, then also \(\omega \rightarrow \omega \rightarrow \ldots \rightarrow \omega \rightarrow \tau\) is strict, and \(\{x;\omega \rightarrow \omega \rightarrow \ldots \rightarrow \omega \rightarrow \tau\} \vDash_S xM_1 \ldots M_n;\tau\).

Theorem 2.14 \(\exists B, \sigma \{ B \vDash_S M;\sigma \ & B, \sigma \omega\text{-free } \} \Leftrightarrow M\) has a normal form.

Proof: \((\Rightarrow)\): If \(B \vDash_S M;\sigma\) and \(B, \sigma\) \(\omega\)-free, then \(B \vdash M;\sigma\) and \(B, \sigma\) \(\omega\)-free. Then by 1.3.(iv) \(M\) has a normal form.

\((\Leftarrow)\): By 2.12 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 2.12 we can conclude that \(B\) and \(\sigma\) do not contain \(\omega\), but the
property that \( \omega \) does not occur at all in the derivation is, in general, lost.

**Theorem 2.15** \( \exists B, \sigma \ [ B \vdash \_ s M : \sigma ] \iff M \text{ has a head normal form.} \)

*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 2.13 and 2.11.

**Corollary 2.16** i) \( \exists B, \sigma \ [ B \vdash \_ s M : \sigma \ & \ B, \sigma \ \omega\text{-free } ] \iff M \text{ has a normal form.} \)

ii) \( \exists B, \sigma \ [ B \vdash \_ s M : \sigma \ & \ \sigma \neq \omega ] \iff M \text{ has a head normal form.} \)

### 2.2 The relation between \( \vdash \ldots \leq \) 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 \( \sigma \) are elements of \( \mathcal{T}_S \). Then \( B \vdash \_ s M : \sigma \iff B \vdash M : \sigma \).

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

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 2.22 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 2.18; the same result has been stated in [14], §4), and the approximation theorem as given in [23].

**Definition 2.17** i) The set \( N \) of \( \lambda \)-normal forms or approximate normal forms is inductively defined by:

a) All term variables are in \( N \), \( \bot \) is in \( N \).

b) If \( A \) is in \( N \), \( A \neq \bot \), then \( \lambda x.A \) is in \( N \).

c) If \( A_1, \ldots, A_n \) are in \( N \), then \( x A_1 \ldots A_n \) is in \( N \).

ii) \( A \in N \) is a direct approximant of \( M \in \Lambda \) if \( A \) matches \( M \) except for occurrences of \( \bot \).

iii) \( A \in N \) 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) \( A(M) = \{ A \in N \mid A \leq M \} \).

v) The type assignment rules of definition 1.2.(i) and 2.2.(i) are generalized to elements of \( N \) by allowing the terms to be elements of \( \lambda \bot \).

**Lemma 2.18** ([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 \( \sigma \) is a type variable: trivial.

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

a) \( \tau' = \omega \). Take \( \sigma' = \omega \).

b) \( \tau' = \tau_1 \cap \cdot \cdot \cdot \cap \tau_m \), each \( \tau_i \in \mathcal{T}_S \). Take \( \sigma' = (\rho' \rightarrow \tau_1) \cap \cdot \cdot \cdot \cap (\rho' \rightarrow \tau_l) \).

c) \( \tau' \) is strict, then take \( \sigma' = \rho' \rightarrow \tau' \)

iii) \( \sigma = \rho \cap \tau \). By induction there are \( \rho' \) 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 \( \sigma' = \rho' \).
\( \text{Theorem 2.20} \) 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 \( \sigma \neq \omega \).

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 2.18 there is a \( \sigma' \in \mathcal{T}_S \) 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 \( \sigma = (\rho_1 \rightarrow \mu_1) \cap \cdots \cap (\rho_n \rightarrow \mu_n) \).

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 \( \mu_i \) is an element of \( \mathcal{T}_S \). By induction there are \( B_i \) and \( \rho'_i, \mu'_i \in \mathcal{T}_S \) such that \( B_i \cup \{x.\rho_i\} \vdash_S A'.\mu'_i \), \( \mu'_i \leq \mu_i \) and \( B_i \cup \{x.\rho_i\} \geq B \) and \( B_i \geq B \).

We can assume, without loss of generality, that each \( \mu'_i \) is an element of \( \mathcal{T}_S \). Then for all \( i \in \{1, \ldots, n\} \) \( B_i \vdash S \lambda x.A'.\rho'_i \rightarrow \mu'_i \). \( \rho'_i \rightarrow \mu'_i \leq \rho_i \rightarrow \mu_i \) and \( B_i \geq B \).

\[
\bigcup_{i \in \{1, \ldots, n\}} B_i \vdash S \lambda x.A'.(\rho'_1 \rightarrow \mu'_1) \cap \cdots \cap (\rho'_n \rightarrow \mu'_n),
\]

\( (\rho'_1 \rightarrow \mu'_1) \cap \cdots \cap (\rho'_n \rightarrow \mu'_n) \leq \sigma \) and \( \bigcup_{i \in \{1, \ldots, n\}} B_i \geq B \).

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 \( B \vdash x.\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \sigma \), \( B \vdash A_1.\tau_1 \), \ldots, and \( B \vdash A_n.\tau_n \). By induction \( B_i \vdash_S A_i.\tau'_i \), \( \tau' \leq \tau \) and \( B_i \geq B \). Take \( B' = \bigcup_{i \in \{1, \ldots, n\}} B_i \), then \( B' \geq B \). By lemma 2.18 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 \( \tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \sigma \leq (\tau'_1 \rightarrow \cdots \rightarrow \tau'_n \rightarrow \sigma_1) \cap \cdots \cap (\tau'_1 \rightarrow \cdots \rightarrow \tau'_n \rightarrow \sigma_k) \) and 1.3.(v), we have \( B' \cup \{x.:(\tau'_1 \rightarrow \cdots \rightarrow \tau'_n \rightarrow \sigma_1) \cap \cdots \cap (\tau'_1 \rightarrow \cdots \rightarrow \tau'_n \rightarrow \sigma_k)\} \geq B \). Also for every \( f \in \{1, \ldots, k\} \)

\[
B' \cup \{x.:(\tau'_1 \rightarrow \cdots \rightarrow \tau'_n \rightarrow \sigma_f)\} \vdash_S xA_1 \ldots A_n.\sigma_f.
\]

\section*{Theorem 2.21}

i) \( B \vdash M.\sigma \iff \exists A \in A(M) \ [ \ B \vdash A.\sigma \] \).

ii) \( B \vdash_S M.\sigma \iff \exists A \in A(M) \ [ \ B \vdash_S A.\sigma \] \).

Proof: i) [23].2.13.

ii) The structure of the proof of [23].2.13 is same as the structure of the proof for this part.
Theorem 2.22 If $B \vdash M: \sigma$ then there are $B', \sigma' \in T_S$ such that $B' \vdash_S M: \sigma'$, $\sigma' \leq \sigma$ and $B' \geq B$.

Proof: If $B \vdash M: \sigma$ then, by 2.21.i there is an $A \in A(M)$ such that $B \vdash A : \sigma$. Then by definition 2.17 there is an $M'$ such that $M' = S M$ and $A$ is a direct approximant of $M'$. By theorem 2.20 there are $B', \sigma' \in T_S$ such that $B' \vdash_S A : \sigma'$, $\sigma' \leq \sigma$ and $B' \geq B$. Then by theorem 2.21.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 2.2.1. 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 $T_S$ to $\phi(F_S)$ 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 2.23 i) Let $M = <D, \cdot, \epsilon>$ be a continuous $\lambda$-model. A mapping $v: T \to \phi(D)$ is a type interpretation if and only if:

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

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

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:

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

iii) On the other hand, a type interpretation is called an $F$ type interpretation if it satisfies:

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

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 satisfaction (called respectively inference, simple and $F$-semantics).

Definition 2.24 We define $\vdash$ by: (where $M$ is a $\lambda$-model, $\xi$ a valuation and $v$ a type interpretation)

i) $M, \xi, v \vdash M: \sigma \iff [M]_v^M \in v(\sigma)$.

ii) $M, \xi, v \vdash B \iff M, \xi, v \vdash x: \sigma$ for every $x: \sigma \in B$.

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

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

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

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

Theorem 2.25 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 $\lambda x. x : (\sigma \to \sigma) \to (\sigma \cap \tau) \to \sigma$. This statement is derivable in the system $\vdash$, but it is not valid in the strict filter $\lambda$-model.
Definition 2.26  i) We define a map \( \nu_0 : \mathcal{T}_S \rightarrow \wp(\mathcal{F}_S) \) by \( \nu_0(\sigma) = \{ d \in \mathcal{F}_S \mid \sigma \in d \} \).

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

Theorem 2.27  The map \( \nu_0 \) is a type interpretation.

Proof: We check the conditions of 2.23.i.

i) \( \forall e \mid e \in \nu_0(\sigma) \Rightarrow d \cdot e \in \nu_0(\tau) \) \( \Rightarrow \)

\[ \forall e \mid e \in \nu_0(\sigma) \Rightarrow e \cdot d \cdot e \in \nu_0(\tau) \] \( \Rightarrow \) (take \( e = \uparrow \sigma \))

\[ \tau \in \varepsilon \cdot d \cdot \tau \Rightarrow (2.27.iii) \]

\[ \exists \rho \in \varepsilon \sigma, v \in d, \eta \mid v \leq \eta \rightarrow \tau \land \rho \leq \eta \] \( \Rightarrow \) (2.27.iii)

\[ \exists v \in d, \eta \mid v \leq \eta \rightarrow \tau \land \sigma \leq \eta \Rightarrow \]

\[ \sigma \rightarrow \tau \in \uparrow \{ \rho \rightarrow \mu \mid \exists v \in d, \eta \mid v \leq \eta \rightarrow \mu \land \rho \leq \eta \} \Rightarrow \]

\[ e \cdot d \in \nu_0(\sigma \rightarrow \tau) \].

ii) Easy.

iii) Trivial.

Notice that although \( \nu_0(\sigma \rightarrow \tau) = \nu_0(\tau \rightarrow \sigma) \), the sets \( \nu_0((\sigma \rightarrow \tau) \rightarrow \sigma) \) and \( \nu_0((\tau \rightarrow \sigma) \rightarrow \sigma) \) are incompatible. We can only show that both contain \( \{ \varepsilon \cdot d \mid \forall e \mid e \in \nu_0(\sigma) \land \nu_0(\tau) \Rightarrow d \cdot e \in \nu_0(\sigma) \} \) and are both contained in \( \{ d \mid \forall e \mid e \in \nu_0(\sigma) \land \nu_0(\tau) \Rightarrow d \cdot e \in \nu_0(\sigma) \} \). However, it is not difficult to prove that \( \varepsilon \cdot \uparrow((\sigma \rightarrow \tau) \rightarrow \sigma) = \varepsilon \cdot \uparrow((\tau \rightarrow \sigma) \rightarrow \sigma) \), so the filters \( \uparrow((\sigma \rightarrow \tau) \rightarrow \sigma) \) and \( \uparrow((\tau \rightarrow \sigma) \rightarrow \sigma) \) represent the same function.

Lemma 2.28  i) \( B \vdash_S M : \sigma \) if and only if \( B_{\xi_B} \vdash_S M : \sigma \).

ii) \( \mathcal{F}_S, \xi_B, \nu_0 \models B \).

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

ii) \( x : \sigma \in B \models (i) \sigma \in \{ \tau \mid B_{\xi_B} \vdash_S x : \tau \} \Rightarrow \sigma \in \llbracket x \rrbracket_{\xi_B} \). So \( \llbracket x \rrbracket_{\xi_B} \in \{ d \in \mathcal{F}_S \mid \sigma \in d \} = \nu_0(\sigma) \).}

The system \( \vdash_S \) of [2] has been proved complete with respect to the simple type semantics. The system \( \vdash_S \) however is not complete in this semantics. This is due to the fact that if we take \( \nu \) to be a type interpretation from \( \mathcal{T}_S \) to \( \wp(\mathcal{F}_S) \), the set \( \{ d \mid \forall e \mid e \in \nu(\sigma) \land \nu(\tau) \Rightarrow d \cdot e \in \nu(\sigma) \} \) is not contained in \( \nu(\sigma \rightarrow \tau) \), since we don’t allow \( \omega \) 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 2.23.i, because of theorem 2.27 completeness can be proved.

Theorem 2.29  Completeness: Let \( \sigma \in \mathcal{T}_S \), then \( B \models M : \sigma \Rightarrow B \vdash_S M : \sigma \).

Proof: \( B \models M : \sigma \Rightarrow (2.24.iii.a, 2.28.ii \& 2.27) \)

\[ B \vdash_M M : \sigma \Rightarrow (2.24.i \& 2.27) \]

\[ \llbracket M \rrbracket_{\xi_B} \in \nu_0(\sigma) \Rightarrow (2.26.i) \]

\[ \sigma \in \llbracket M \rrbracket_{\xi_B} \Rightarrow (2.10) \]

\( B \vdash_S M : \sigma \).

3 The system without \( \omega \)

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 \( \omega \). We will show that the intersection type assignment system without \( \omega \) yields a filter model for the \( \lambda I \)-calculus (subsection 3.3.1), show that for the \( \lambda I \)-calculus the intersection type assignment is conservative over the one without \( \omega \) (subsection 3.3.2) and prove that this type assignment is complete for
the λI-calculus with respect to the simple type semantics (subsection 3.3.3). Furthermore we will prove that each term typeable by the system without $\omega$ is strongly normalizable (subsection 3.3.4).

While obtaining these results we could of course use the result of the previous section and look at the system without $\leq$ and without $\omega$, 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 3.3.4 are greatly facilitated by the presence of derivation rule ($\leq$). In fact, the strong normalization property for the system without $\leq$ and $\omega$ follows immediately from the results of 3.3.4. Moreover we could prove a completeness result for this system with respect to the inference semantics.

3.1 $\omega$-free derivations

In this subsection we present a restriction of the intersection type assignment system in which the type $\omega$ is removed. This system yields a filter $\lambda$I-model.

Definition 3.1  i) $\mathcal{T}_{-\omega}$, the set of $\omega$-free types is inductively defined by:

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

b) If $\sigma, \tau \in \mathcal{T}_{-\omega}$, then $\sigma \cap \tau, \sigma \rightarrow \tau \in \mathcal{T}_{-\omega}$.

ii) On $\mathcal{T}_{-\omega}$ 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 $\omega$-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 $B \vdash_{-\omega} M: \sigma$.

Lemma 3.2  i) If $B \vdash_{-\omega} MN: \sigma \iff \exists \tau \{ B \vdash_{-\omega} M: \tau \rightarrow \sigma \land B \vdash_{-\omega} N: \tau \}$.

ii) $B \vdash_{-\omega} \lambda x.M: \sigma \rightarrow \tau \iff B \backslash \{ x: \sigma \} \vdash_{-\omega} M: \tau$.

iii) If $B \vdash_{-\omega} \lambda x.M: \rho$, then there are $\sigma_1, \ldots, \sigma_n, \tau_1, \ldots, \tau_n$ such that $\rho = (\sigma_1 \rightarrow \tau_1) \cap \ldots \cap (\sigma_n \rightarrow \tau_n)$.

iv) If $B \backslash z \cup \{ z: \sigma \} \vdash_{-\omega} Mz: \tau$ and $z \notin \text{FV}(M)$, then $B \vdash_{-\omega} M: \sigma \rightarrow \tau$.

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 $\omega$.

Definition 3.3  i) A subset $d$ of $\mathcal{T}_{-\omega}$ 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) $\mathcal{F}_{-\omega} = \{ d \subseteq \mathcal{T}_{-\omega} \mid d \text{ is an I-filter} \}$. 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 $\mathcal{T}_{-\omega}$, 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 $\rho, \mu \in \mathcal{T}_{-\omega}$: 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 $\langle D, \leq \rangle$ be a cpo with least element $\bot$. The set of strict functions is defined as usual, i.e. as the set of continuous functions that at least map $\bot$ onto $\bot$. We denote by $[D \rightarrow \bot, D]$ the set of strict functions from $D$ to $D$. 

**Theorem 3.10** Proof: By induction on the structure of lambda terms.

### Definition 3.5
We define \( F: \mathcal{F}_{\omega} \rightarrow [\mathcal{F}_{\omega} \rightarrow \bot \mathcal{F}_{\omega}] \) and \( G: [\mathcal{F}_{\omega} \rightarrow \bot \mathcal{F}_{\omega}] \rightarrow \mathcal{F}_{\omega} \) by:

1. \( F \ d \ e = d \cdot e \).
2. \( G \ f = \uparrow \{ \sigma \rightarrow \tau \in \mathcal{T}_{\omega} \mid \tau \in f(\uparrow \sigma) \} \).

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 \( < \mathcal{D}, \cdot, \varepsilon > \) is called a \( \lambda I \)-model if and only if in \( \mathcal{D} \) there are five elements \( i, b, c, s \) and \( \varepsilon \) that satisfy the following conditions:

1. \( i \cdot d = d \).
2. \((b \cdot d) \cdot e \cdot f = d \cdot (e \cdot f)\).
3. \((c \cdot d) \cdot e \cdot f = (d \cdot f) \cdot e \).
4. \((s \cdot d) \cdot e \cdot f = (d \cdot f) \cdot (e \cdot f)\).
5. \((\varepsilon \cdot d) \cdot e = d \cdot e \) and \( \forall d \in \mathcal{D} \) \( e \cdot d = f \cdot d \Rightarrow \varepsilon \cdot e = \varepsilon \cdot f \) and \( \varepsilon \cdot \varepsilon = \varepsilon \).

Moreover, in [10] the following is stated:

### Proposition 3.7 ([10])
If \( < \mathcal{D}, \leq > \) is a cpo and there are continuous maps \( F: \mathcal{D} \rightarrow [\mathcal{D} \rightarrow \bot \mathcal{D}] \) and \( G: [\mathcal{D} \rightarrow \bot \mathcal{D}] \rightarrow \mathcal{D} \) such that:

1. \( F \circ G = \text{id}_{[\mathcal{D} \rightarrow \bot \mathcal{D}]} \).
2. \( G \circ F \in [\mathcal{D} \rightarrow \bot \mathcal{D}] \).

Then \( \mathcal{D} \) is a \( \lambda I \)-model.

### Theorem 3.8
\( F \) and \( G \) as defined in 3.5 yield a \( \lambda I \)-model.

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

1. \( (F \circ G)(f) \) \( d = \{ \mu \mid \exists \rho \in d \ [ \rho \rightarrow \mu \} \uparrow (\{ \sigma \rightarrow \tau \mid \tau \in f(\uparrow \sigma) \}) \} = (3.4) \)

\( \{ \mu \mid \exists \rho \in d \ [ \mu \in f(\uparrow \rho) ] \} = f(d) \).

2. \( G \circ F (\emptyset) = \{ \rho \rightarrow \mu \mid \mu \in \{ \sigma \mid \exists \tau \in \uparrow \rho [ \tau \rightarrow \sigma \in \emptyset ] \} \} = \emptyset \).

That the type discipline without \( \omega \) gives rise to a model for the \( \lambda I \)-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 \( \mathcal{D} \simeq \mathcal{P}_\omega \times [\mathcal{D} \rightarrow \bot \mathcal{D}] \), with \( \mathcal{D} = \mathcal{P}_\omega \) (where \( \mathcal{P}_\omega \) is the powerset of natural numbers) and \( i: \mathcal{D} \rightarrow \mathcal{P}_\omega \times [\mathcal{D} \rightarrow \bot \mathcal{D}] \) is defined by \( i(d) = < d, \lambda x.\bot > \) (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 \( \xi \) be a valuation of term variables in \( \mathcal{F}_{\omega} \).

1. \( [M]_\xi, \) the interpretation of \( \lambda I \)-terms in \( \mathcal{F}_{\omega} \) via \( \xi \) is inductively defined by:
   a) \( [x]_\xi = \xi(x) \).
   b) \( [MN]_\xi = F ([M]_\xi) [N]_\xi \).
   c) \( [\lambda x.M]_\xi = G (\lambda v \in \mathcal{F}_{\omega}. [M]_{\xi(v/x)} ) \).
2. \( B_\xi = \{ x ; \sigma \mid \sigma \in \xi(x) \} \).

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

### Theorem 3.10
For all \( M \in \lambda I, \xi; [M]_\xi = \{ \sigma \mid \lambda v \in \mathcal{F}_{\omega}. [M]_{\xi(v/x)} \emptyset \} \).

**Proof:** By induction on the structure of lambda terms.
Lemma 3.11 If $M$ is in normal form and $B \vdash \omega \lambda \sigma$ reduces to $\omega \lambda \sigma$ and $B \vdash \omega \lambda \sigma$. Then clearly $0 (D D) = \beta I$ and $\vdash \omega \lambda \sigma$ but we cannot give a derivation without $\omega$ for $0 (D D) : \sigma \rightarrow \sigma$.

3.2 The relation between $\vdash \omega \lambda$ and $\vdash$

Type assignment in the intersection type assignment system is not fully conservative over the type assignment without $\omega$. If for example we have $B \vdash \omega \lambda \sigma$ such that $B$ and $\sigma$ are $\omega$-free, but $M$ contains a sub-term that has no normal form, $\omega$ 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 $B \vdash \omega \lambda \sigma$ with $B$ and $\sigma$ $\omega$-free, there is an $M'$ such that $M$ reduces to $M'$ and $B \vdash \omega \lambda \sigma$. We will show this by proving for terms in normal form that each $\omega$-free predicate, starting from a $\omega$-free basis, can be derived in $\vdash \omega \lambda$, 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 $B \vdash \omega \lambda \sigma$ such that $B$ and $\sigma$ are $\omega$-free, then $B \vdash \omega \lambda \sigma$.

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 $x: \sigma$, ..., $x: \sigma$ in $B$ such that $\sigma \cap \cdots \cap \sigma \leq \sigma$. Then obviously $B \vdash \omega x: \sigma$.

ii) $B \vdash \lambda x. M: \sigma$. Then $\sigma \equiv (\rho_1 \cap \cdots \cap \rho_n) \cap \cdots \cap (\rho_1 \cap \cdots \cap \rho_n)$ for some $n \geq 1$, and by 1.3.(ii) for every $i \in \{1, \ldots, n\}$: $B \cup \{x: \sigma_i\} \vdash \omega \lambda M: \sigma_i$. By induction for every $i \in \{1, \ldots, n\}$: $B \cup \{x: \sigma_i\} \vdash \omega \lambda M: \sigma_i$. So $B \vdash \omega \lambda M: \sigma$.

iii) $B \vdash x M_1 \ldots M_n: \sigma$. By 1.3.(i) there are $\tau_1, \ldots, \tau_n$ such that $B \vdash x: \tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \sigma$, $B \vdash M_1: \tau_1$, ..., and $B \vdash M_n: \tau_n$. We have $x: \rho_1, \ldots, x: \rho_n$ in $B$ such that $\rho_1 \cap \cdots \cap \rho_n \leq \tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \sigma$. By 1.3.(vi) this implies $\rho_1 \cap \cdots \cap \rho_n \leq (\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \sigma) \cap \cdots \cap (\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \sigma)$ for $\tau_1, \ldots, \tau_n$. Thus such that $\tau_i \geq \tau_j$ with $1 \leq i \leq n$, $1 \leq j \leq s$ and $\sigma \cap \cdots \cap \sigma \leq \sigma$. Then by (s) and (n) for every $i \in \{1, \ldots, n\}$ we have $B \vdash x: \tau_i \cap \cdots \cap \tau_i$.

Since each $\tau_i$ occurs in a statement in the basis, the induction hypothesis is applicable and for every $i \in \{1, \ldots, n\}$ we have $B \vdash \omega \lambda M_i: \tau_i \cap \cdots \cap \tau_i$. Also

\[
(\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \sigma) \cap \cdots \cap (\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \sigma) \leq \\
(\tau_1 \cap \cdots \cap \tau_n) \rightarrow (\sigma \cap \cdots \cap \sigma),
\]

so $B \vdash x: (\tau_1 \cap \cdots \cap \tau_n) \rightarrow (\sigma \cap \cdots \cap \sigma)$ and by part i

\[
B \vdash \omega x: (\tau_1 \cap \cdots \cap \tau_n) \rightarrow (\sigma \cap \cdots \cap \sigma).
\]

But then by $(\rightarrow E)$ $B \vdash \omega x M_1 \ldots M_n: \sigma$ and by $(\leq) B \vdash \omega x M_1 \ldots M_n: \sigma$.  

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

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

As remarked in the introduction, if we are interested in deriving types without $\omega$ occurrences, the type constant $\omega$ 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 $\rho$ such that $B \vdash_{-\omega} N : \rho$, then even for this $N$ we would not need to use $\omega$. Unfortunately there are lambda terms $M$ that contain a sub-term $N$ that must be typed with $\omega$ in $B \vdash M : \sigma$, even if $\omega$ does not occur in $B$ and $\sigma$. We can even find strongly normalizable 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 $\lambda I$-term and $B \vdash M : \sigma$ where $\omega$ does not occur in $B$ and $\sigma$, then $B \vdash_{-\omega} M : \sigma$.

Proof: If $B \vdash M : \sigma$ and $\sigma$ are $\omega$-free, then by 1.3.(iii), $M$ has a normal form $M'$. Then also $B \vdash M' : \sigma$. By lemma 3.11 we have $B \vdash_{-\omega} M' : \sigma$. Because $M$ and $M'$ are $\lambda I$-terms, by corollary 3.21 we obtain $B \vdash_{-\omega} M : \sigma$.

3.3 The type assignment without $\omega$ is complete for the $\lambda I$-calculus

In this subsection completeness of type assignment without $\omega$ for the $\lambda I$-calculus is proved using the method of [2]. The notions of type interpretation as defined in 2.23 lead also in the case of the $\lambda I$-calculus in a natural way to the following definitions for semantic satisfiability.

Definition 3.14 As in 2.24 we define $\vdash$ by: (where $M$ is a $\lambda I$-model, $\xi$ a valuation and $\nu$ a type interpretation)

i) $M, \xi, \nu \vdash M : \sigma \iff \llbracket M \rrbracket^\sigma_\xi \in \nu(\sigma)$.

ii) $M, \xi, \nu \vdash B \iff M, \xi, \nu \vdash x : \sigma$ for every $x : \sigma \in B$.

iii) a) $B \vdash M : \sigma \iff \forall M, \xi, \nu \ [ M, \xi, \nu \vdash B \Rightarrow M, \xi, \nu \vdash M : \sigma ]$.

b) $B \vdash_{s} M : \sigma \iff \forall M, \xi$, simple type interpretations $\nu \ [ M, \xi, \nu \vdash B \Rightarrow M, \xi, \nu \vdash M : \sigma ]$.

c) $B \vdash_{F} M : \sigma \iff \forall M, \xi, F$ type interpretations $\nu \ [ M, \xi, \nu \vdash B \Rightarrow M, \xi, \nu \vdash M : \sigma ]$.

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

Theorem 3.15 Soundness: If $B \vdash_{-\omega} M : \sigma$ then $B \vdash_{s} M : \sigma$.

Proof: By induction on the structure of derivations.

Definition 3.16 i) We define a map $v_0 : \mathcal{T}_{-\omega} \rightarrow \mathcal{F}_{-\omega}$ by $v_0(\sigma) = \{ d \in \mathcal{F}_{-\omega} \mid \sigma \in d \}$.

ii) $\xi_B(x) = \{ \sigma \in \mathcal{T}_{-\omega} \mid B \vdash_{-\omega} x : \sigma \}$

Theorem 3.17 i) The map $v_0$ is a simple type interpretation.

ii) $B \vdash_{-\omega} M : \sigma$ if and only if $B_{\xi_B} \vdash_{-\omega} M : \sigma$.

iii) $\mathcal{F}_{-\omega}, \xi_B, v_0 \vdash_{s} B$.
Proof: i) Easy.

ii) Because for every $x$, $\xi_B(x)$ is an I-filter.

iii) $x: \sigma \in B \Rightarrow \sigma \in \{ \tau | B_{\xi_B} \vdash \omega x: \tau \} \Rightarrow \sigma \in [x]_{\xi_B}$. So $[x]_{\xi_B} \in \{ d | d \in \mathcal{F}_{-\omega} | \sigma \in d \}$. 

\[ \square \]

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

Proof: $B \vdash_s M: \sigma \Rightarrow (3.14.iii, 3.17.i \& 3.17.iii)$

$\mathcal{F}_{-\omega}, \xi_B, v_0 \vdash_s M: \sigma \Rightarrow (3.14.i \& 3.17.i)$

$\|M\|_{\xi_B} \in v_0(\sigma) \Rightarrow (3.16.i)$

$\sigma \in [M]_{\xi_B} \Rightarrow (3.10)$

$B_{\xi_B} \vdash_{-\omega} M: \sigma \Rightarrow (3.17.ii)$

$B \vdash_{-\omega} M: \sigma$.

\[ \square \]

### 3.4 The set of lambda terms typeable by means of the derivation rules ($\cap I$), ($\cap E$), ($\rightarrow I$) and ($\rightarrow E$) is exactly the set of strongly normalizable 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: $B \vdash_{-\omega} M: \sigma \Leftrightarrow M$ is strongly normalizable.

To prove that each term typeable by the rules ($\cap I$), ($\cap E$), ($\rightarrow I$) and ($\rightarrow E$) is strongly normalizable, we will prove even more: we will show that if $B \vdash_{-\omega} M: \sigma$ (i.e. using also rule ($\leq$)), then $M$ is strongly normalizable. 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 normalizable terms. In this subsection we present a proof for the same result, different from the one given in [22]. The proof that all strongly normalizable terms are typeable in the system without $\omega$ and ($\leq$) is given in corollary 3.22.

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 $\lambda I$-calculus exactly the same as ‘not having a normal form’, as well as that ‘normalizable’ and ‘strongly normalizable’ coincide) onto the empty filter.

Notice also that we no longer restrict ourselves to the $\lambda I$-terms, but prove the statement for the full $\lambda K$-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 normalizable, then so is $xM_1 \ldots M_n N$.

ii) If $Mz$ is strongly normalizable (where $z$ does not occur free in $M$), then so is $M$.

iii) If $M[x := N]$ and $N$ are strongly normalizable, then so is $(\lambda x.M)N$.

\[ \square \]

**Lemma 3.20** If $B \vdash_{-\omega} C[M[x := N]]:\tau$ and $B \vdash_{-\omega} N:\rho$, then $B \vdash_{-\omega} C[(\lambda x.M)N]:\tau$, where $C[\cdots]$ is the notation for a context.

Proof: By induction on the structure of contexts. We omit the case that the context is an application, since it is trivial.

i) $C[M[x := N]] = M[x := N]$. We can assume that $x$ does not occur in $B$.

a) $N$ occurs $n$ times in $M[x := N]$, each time typed by, say, $\sigma_i$.

$B \vdash_{-\omega} M[x := N]:\tau \Rightarrow B \cup \{ x: \sigma_1 \cap \cdots \cap \sigma_n \} \vdash_{-\omega} M: \tau \& B \vdash_{-\omega} N: \sigma_1 \cap \cdots \cap \sigma_n \Rightarrow$
Lemma 3.24 Let $\sigma$ reduces to the latter. We know that $\{\text{derivation the derivation rule (Definition 3.23)}\}$

\[ \begin{align*}
B & \vdash_\omega \lambda x. M : (\sigma_1 \cap \cdots \cap \sigma_n) \rightarrow \tau \\
& \quad \land B \vdash_\omega N : \sigma_1 \cap \cdots \cap \sigma_n \Rightarrow \\
B & \vdash_\omega (\lambda x. M) N : \tau. 
\end{align*} \]

b) $N$ does not occur in $M[x := N]$, so $x \notin \text{FV}(M)$.

$B \vdash_\omega M : \tau \land B \vdash_\omega N : \rho \Rightarrow (x \notin \text{FV}(M))$

$B \cup \{x: \rho\} \vdash_\omega M : \tau \land B \vdash_\omega N : \rho \Rightarrow \\
B \vdash_\omega \lambda x. M : \rho \rightarrow \tau \land B \vdash_\omega N : \rho \Rightarrow \\
B \vdash_\omega (\lambda x. M) N : \tau.$

\[ \begin{align*}
\forall 1 \leq i \leq n & \{ B \cup \{y: \rho_i\} \vdash_\omega C[M[x := N] : \mu_i ] \land B \vdash_\omega N : \rho \Rightarrow \\
& \forall 1 \leq i \leq n \{ B \cup \{y: \rho_i\} \vdash_\omega C[(\lambda x. M) N : \mu_i ] \Rightarrow \\
B & \vdash_\omega \lambda y. C[(\lambda x. M) N] : \tau. 
\end{align*} \]

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

Notice that for the $\lambda$-calculus part 3.20.i.b is not applicable and the condition $B \vdash_\omega N : \rho$ is not needed. So the following is an immediate result:

**Corollary 3.21** Let $M$ and $M'$ be $\lambda$-I-terms, such that $M$ reduces to $M'$ and there are $B$ and $\sigma$ such that $B \vdash_\omega M : \sigma$. Then $B \vdash_\omega M' : \sigma$. \hfill \[ \blacksquare \]

Lemma 3.20 is also essentially the proof for the statement that each strongly normalizable term can be typed in the system $\vdash_\omega$. \hfill \[ \blacksquare \]

**Corollary 3.22** If $M$ is strongly normalizable, then there are $\sigma$ and $B$ such that $B \vdash_\omega M : \sigma$ 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 ([1],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 2.12. \hfill \[ \blacksquare \]

In order to prove that each term typeable in $\vdash_\omega$ is strongly normalizable we introduce a notion of computability. From now on, we will abbreviate ‘$M$ is strongly normalizable’ by SN($M$).

**Definition 3.23** (cf. [21]) Comp($B$, $M$, $\rho$) is inductively defined by:

i) Comp($B$, $M$, $\phi$) $\iff$ $B \vdash_\omega M : \phi$ \& SN($M$).

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

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

**Lemma 3.24** Let $\sigma$ and $\tau$ 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 3.25  

i) If $B \vdash_{\omega} xM_1 \ldots M_n; \rho$ and $\text{SN}(xM_1 \ldots M_n)$, then $\text{Comp}(B, xM_1 \ldots M_n, \rho)$.

ii) If $\text{Comp}(B, M, \rho)$, then $B \vdash_{\omega} M; \rho$ and $\text{SN}(M)$.

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) $B \vdash_{\omega} xM_1 \ldots M_n; \sigma \rightarrow \tau$ and $\text{SN}(xM_1 \ldots M_n)$ \Rightarrow

$\text{Comp}(B', \sigma, \tau) \Rightarrow$

$B \vdash_{\omega} xM_1 \ldots M_n; \sigma \rightarrow \tau$ and $\text{SN}(xM_1 \ldots M_n)$ and $B' \vdash_{\omega} N; \sigma$ and $\text{SN}(N)$ \Rightarrow

$\text{Comp}(B', \sigma, \tau) \Rightarrow B \cup B' \vdash_{\omega} xM_1 \ldots M_n; \tau$ and $\text{SN}(xM_1 \ldots M_n)$ \Rightarrow

$\text{Comp}(B', \sigma, \tau) \Rightarrow B \cup B' \vdash_{\omega} xM_1 \ldots M_n, \tau) \Rightarrow (3.23.(ii))$

$\text{Comp}(B, xM_1 \ldots M_n, \sigma \rightarrow \tau)$.

ii) $\text{Comp}(B, M, \sigma \rightarrow \tau)$ and $z \notin \text{FV}(M) \Rightarrow$

$\text{Comp}(B, M, \sigma \rightarrow \tau)$ and $\text{Comp}(\{z; \sigma\}, z, \sigma)$ and $z \notin \text{FV}(M) \Rightarrow (3.23.(ii))$

$\text{Comp}(B \cup \{z; \sigma\}, \sigma)$ and $\text{Comp}(B \cup \{z; \sigma\}, z, \sigma)$ and $z \notin \text{FV}(M) \Rightarrow (3.23.(ii))$

$B \vdash_{\omega} M; \sigma \rightarrow \tau$ and $\text{SN}(M)$.

Lemma 3.26  

$\text{Comp}(B \cup B', C[M[x := N]], \sigma)$ and $\text{Comp}(B', N, \rho)$ \Rightarrow $\text{Comp}(B \cup B', C[(\lambda x.M)N], \sigma)$.

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

$\text{Comp}(B \cup B', C[M[x := N]], \phi)$ and $\text{Comp}(B', N, \rho) \Rightarrow (3.25.(ii))$

$B \vdash_{\omega} C[M[x := N]]; \phi$ and $\text{SN}(C[M[x := N]])$ and $B \vdash_{\omega} N; \rho$ and $\text{SN}(N) \Rightarrow (3.20)$

$B \vdash_{\omega} C[(\lambda x.M)N]; \phi$ and $\text{SN}(C[(\lambda x.M)N]) \Rightarrow (3.25.(ii))$

$\text{Comp}(B \cup B', C[(\lambda x.M)N], \phi)$.

Theorem 3.27  

If $B = \{x_1; \mu_1, \ldots, x_n; \mu_n\}$ and $\text{Comp}(B_\nu, N_\nu, \mu_\nu)$ and $B \vdash_{\omega} M; \sigma$, then $\text{Comp}(B_1 \cup \cdots \cup B_n, M[x_1 := N_1, \ldots, x_n := N_n], \sigma)$.

Proof: By induction on the structure of derivations. We will only show the non-trivial parts.

i) $(\rightarrow I)$. Then $M \equiv \lambda y.M'$, $\sigma = \rho \rightarrow \tau$, and $B \cup \{y; \rho\} \vdash_{\omega} M'; \tau$.

$B = \{x_1; \mu_1, \ldots, x_n; \mu_n\}$ and $\text{Comp}(B_\nu, N_\nu, \mu_\nu)$ and $B \cup \{y; \rho\} \vdash_{\omega} M'; \tau$ \Rightarrow

$\text{Comp}(B', \nu, \rho) \Rightarrow$

$\text{Comp}(B_1 \cup \cdots \cup B_n \cup B', M'[x_1 := N_1, \ldots, x_n := N_n], y := N] \Rightarrow (3.26)$

$\text{Comp}(B', \nu, \rho) \Rightarrow$

$\text{Comp}(B_1 \cup \cdots \cup B_n \cup B', (\lambda y.M'[x_1 := N_1, \ldots, x_n := N_n])N, \tau) \Rightarrow (3.23.(ii))$

$\text{Comp}(B_1 \cup \cdots \cup B_n, (\lambda y.M'[x_1 := N_1, \ldots, x_n := N_n], \rho \rightarrow \tau)$.

ii) $(\rightarrow E)$. Then $M \equiv M_1 M_2$, $B \vdash_{\omega} M_1; \rho \rightarrow \tau$ and $B \vdash_{\omega} M_2; \rho$.

$B = \{x_1; \mu_1, \ldots, x_n; \mu_n\}$ and $\text{Comp}(B_\nu, N_\nu, \mu_\nu)$ and $B \vdash_{\omega} M_1; \rho \rightarrow \tau$ and $B \vdash_{\omega} M_2; \rho$ \Rightarrow

$\text{Comp}(B' \cup \cdots \cup B_n, M_1[x_1 := N_1, \ldots, x_n := N_n], \rho \rightarrow \tau)$ and

$\text{Comp}(B_1 \cup \cdots \cup B_n, M_2[x_1 := N_1, \ldots, x_n := N_n], \rho \rightarrow \tau) \Rightarrow (3.23.(ii))$

$\text{Comp}(B_1 \cup \cdots \cup B_n, (M_1 M_2)[x_1 := N_1, \ldots, x_n := N_n], \tau)$.

Theorem 3.28  

If $B \vdash_{\omega} M; \sigma$, then $\text{SN}(M)$.

Proof: $B \vdash_{\omega} M; \sigma \Rightarrow (3.27)$ $\text{Comp}(B, M, \sigma) \Rightarrow (3.25.(ii))$ $\text{SN}(M)$.

We can now prove the main theorem of this subsection.

Theorem 3.29  

$\{M \mid M$ is typeable by means of the derivation rules $(\cap I)$, $(\cap E)$, $(\rightarrow I)$ and $(\rightarrow E)$ \} =
\{ M \mid M \text{ is strongly normalizable} \}.

Proof: ( ⊆ ) : If \( M \) is typeable by means of the derivation rules (\( \land \)I), (\( \land \)E), (\( \rightarrow \)I) and (\( \rightarrow \)E), then certainly \( B \vdash_{\omega} M : \sigma \). Then by theorem 3.28, \( M \) is strongly normalizable.

( ⊇ ) : If \( M \) is strongly normalizable, then by corollary 3.22 there are \( \sigma \) 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


