Essential Intersection Type Assignment

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. This paper introduces a notion of intersection type assignment on the Lambda Calculus that is a restriction of the BCD-system as presented in [4]. This restricted system is essential in the following sense: it is an almost syntax directed system that satisfies all major properties of the BCD-system. The set of typeable terms can be characterized in the same way, the system is complete with respect to the simple type semantics, and it has the principal type property.

Introduction

In the recent years several notions of type assignment for several (extended) lambda calculi have been studied. The oldest among these is the well understood and well defined notion of type assignment on lambda terms, known as the Curry Type Assignment System [6] which expresses abstraction and application. It is well known that in that system, the problem of typeability

Given a term M, are there basis B and type σ such that $B \vdash M{:}\sigma$

is decidable. Although many of the now existing type assignment systems for functional programming languages are based on (extensions of) Curry's system, it has some drawbacks. In Curry's system it is, for example, not possible to assign a type to the term $(\lambda x.xx)$; moreover, 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. The Intersection Type Discipline as presented in [5] (a more enhanced system was presented in [4]) is an extension of Curry's system that does not have these drawbacks. The extension being made consists mainly of allowing for term variables (and terms) to have more than one type. Intersection types are constructed by adding, next to the type constructor '$\rightarrow$' of Curry's system, the type constructor '$\cap$' and the type constant 'ω'. By introducing this extension a system is obtained that is very powerful: is closed under β-equality: if $B \vdash M{:}\sigma$ and $M =_\beta N$, then $B \vdash N{:}\sigma$. Because of this power, in the intersection system (and even in the system that does not contain ω) type assignment is undecidable.

The type assignment system presented in [4] (the BCD-system) is based on the system as presented in [5]; it defines the set of intersection types $\mathcal{T}$ in a more general way, and is 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 $(\leq)$, as well as the more general treatment of intersection types were introduced mainly to prove completeness of type assignment.

In that paper it was shown that the set of types derivable for a lambda term in the 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 – $[\![M]\!]_\xi$ – is defined in the standard way, and gives a filter lambda model $\mathcal{F}$. The main result of that paper is that, using this model, completeness is proved by proving the statement: $\vdash M{:}\sigma \iff [\![M]\!] \in \upsilon(\sigma)$, where $\upsilon : \mathcal{T} \rightarrow \mathcal{F}$ is a simple type interpretation as defined in [10]. In order to prove the $\Leftarrow$-part of this statement (completeness), the relation $\leq$ is needed. The BCD-system has the principal type property, as was shown in [13]; the set of operations needed for this system consists of substitutions, expansions, and rises. A disadvantage of the BCD-system is that it is too general: in this system there are several ways to deduce a desired result, due to the presence of the derivation rules ($\cap$I), ($\cap$E) and ($\leq$). These rules not only allow of superfluous steps in derivations, but they also make it possible to give essentially different derivations for the same result. Moreover, in [4] the relation $\leq$ induced an equivalence relation $\sim$ on types. Equivalence classes are big (for example: $\omega \sim \sigma \rightarrow \omega$, for all types σ) and type assignment is closed for $\sim$. And although the set $\{<B, \sigma> \mid B \vdash M{:}\sigma\}$ can be generated using the three operations specified in [13], the problem

Given a term M and type σ, is there a B such that $B \vdash M{:}\sigma$?

is complicated. The Essential Type Assignment System as presented in this paper is a true restriction of the system of [4] that satisfies all properties of that system, and is also an extension of Curry's system. We will compare the notion of type assignment presented in this paper with the one defined in [9]; the set of derivable statements of these two systems will turn out to be exactly the same. We will show that, in order to prove a completeness result using intersection types, there is no need to be as general as in [4]; this result can also be obtained for the essential system. The main advantage of the essential system over the intersection system is that the set of types assignable to a term is significantly smaller. An other advantage of the essential system is that derivations are syntax-directed: there is, unlike in the BCD-system, a one-one relationship between terms and skeletons of derivations. These two features are supported by a less complicated type structure.

The system presented here is also an extension of the Strict Type Assignment System as presented in [1]. The major difference is that the essential system will prove to be closed for η-reduction: If $B \vdash_{\mathrm{E}} M{:}\sigma$ and $M \rightarrow_\eta N$, then $B \vdash_{\mathrm{E}} N{:}\sigma$. This does not hold for the strict system.

In this paper, to avoid parentheses in the notation of types, '$\rightarrow$' is assumed to associate to the right; as in logic, '$\cap$' binds stronger than '$\rightarrow$'. I assume the reader to be familiar with the Lambda Calculus, including the definition of the set $\mathcal{N}$ of terms in $\lambda\perp$-normal form (like there, I use the symbol $\perp$ instead of Ω), and the notion of approximant.

1 The Barendregt-Coppo-Dezani type assignment system

In this subsection we give the definitions of the system as presented in [4] that are relevant for the results presented in this paper, together with its major features.

Definition 1 [4]. 1. $\mathcal{T}$, the set of *BCD-types* is inductively defined by: All type-variables φ_0, φ_1, $\ldots \in \mathcal{T}$, $\omega \in \mathcal{T}$, and if σ and $\tau \in \mathcal{T}$, then $(\sigma \rightarrow \tau)$ and $(\sigma \cap \tau) \in \mathcal{T}$.
2. A *BCD-statement* is an expression of the form $M{:}\sigma$ where $M \in \Lambda$ and $\sigma \in \mathcal{T}$.
3. A *BCD-basis* is a finite set of statements over (distinct) term-variables.

Definition 2. 1. On $\mathcal{T}$ the type inclusion relation $\leq$ is defined as the smallest relation satisfying: $\sigma \leq \sigma$, $\sigma \leq \omega$, $\omega \leq \omega \rightarrow \omega$, $\sigma \cap \tau \leq \sigma$, $\sigma \cap \tau \leq \tau$, $(\sigma \rightarrow \tau) \cap (\sigma \rightarrow \rho) \leq \sigma \rightarrow \tau \cap \rho$, $\sigma \leq \tau \leq \rho \;\Rightarrow\; \sigma \leq \rho$, $\sigma \leq \tau \;\&\; \sigma \leq \rho \;\Rightarrow\; \sigma \leq \tau \cap \rho$, and $\rho \leq \sigma \;\&\; \tau \leq \mu \;\Rightarrow\; \sigma \rightarrow \tau \leq \rho \rightarrow \mu$.
2. $B \leq B' \iff \forall\, x{:}\sigma' \in B' \;\exists\, x{:}\sigma \in B \;[\sigma \leq \sigma']$.
3. $\sigma \sim \tau \iff \sigma \leq \tau \leq \sigma$, and $B \sim B' \iff B \leq B' \leq B$.

Definition 3 [4]. 1. *BCD-type assignment* and *BCD-derivations* are defined by the following natural deduction system.

$$(\leq):\ \frac{M{:}\sigma \qquad \sigma \leq \tau}{M{:}\tau} \qquad (\rightarrow\mathrm{E}):\ \frac{M{:}\sigma \rightarrow \tau \qquad N{:}\sigma}{MN{:}\tau} \qquad (\rightarrow\mathrm{I}):\ \frac{\begin{array}{c}[x{:}\sigma]\\ \vdots \\ M{:}\tau\end{array}}{\lambda x.M{:}\sigma \rightarrow \tau}\ \text{(a)}$$

$$(\cap\mathrm{I}):\ \frac{M{:}\sigma \qquad M{:}\tau}{M{:}\sigma \cap \tau} \qquad (\cap\mathrm{E}):\ \frac{M{:}\sigma \cap \tau}{M{:}\sigma} \quad \frac{M{:}\sigma \cap \tau}{M{:}\tau} \qquad (\omega):\ \frac{}{M{:}\omega}$$

(a) If $x{:}\sigma$ is the only statement about x on which $M{:}\tau$ depends.
2. $B \vdash M{:}\sigma$ is used for: $M{:}\sigma$ is derivable from a basis B with a BCD-derivation.

The BCD-type discipline has a great expressive power: all solvable terms have types not equivalent to ω, and a term has a normal form if and only if it has a type without ω occurrences.

The relation between types assignable to a lambda term and those assignable to its approximants is formulated by the following:

Property 4 [13]. *$B \vdash M{:}\sigma \iff \exists\, A \in \mathcal{A}(M)\ [B \vdash A{:}\sigma]$.* ∎

The main result of [4] is the proof for completeness of type assignment. This is achieved by showing that 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 construction of a filter lambda model $\mathcal{F}$ and the definition of a map from types to elements of this model (a *simple type interpretation*) make the proof of completeness possible: if the interpretation of the term M is an element of the interpretation of the type σ, then M is typeable with σ. Filters and the filter λ-model $\mathcal{F}$ are defined by:

Definition 5 [4]. A *BCD-filter* is a subset $d \subseteq \mathcal{T}$ such that: $\omega \in d$, if σ, $\tau \in d$ then $\sigma \cap \tau \in d$, and if $\sigma \geq \tau \in d$ then $\sigma \in d$. $\mathcal{F} = \{d \mid d \text{ is a BCD-filter}\}$, and for d_1, $d_2 \in \mathcal{F}$ define $d_1 \cdot d_2 = \{\tau \in \mathcal{T} \mid \exists\, \sigma \in d_2\ [\sigma \rightarrow \tau \in d_1]\}$.

The following properties are proved in [4]:

- $\forall M \in \Lambda\, [\{\sigma \mid \exists B\, [B \vdash M{:}\sigma]\} \in \mathcal{F}]$.
- For $M \in \Lambda$ define $[\![M]\!]_\xi = \{\sigma \mid \exists B \subseteq \{x{:}\tau \mid \tau \in \xi\,(x)\}\, [B_\xi \vdash M{:}\sigma]\}$, where ξ is a valuation of term-variables in $\mathcal{F}$. Then $<\mathcal{F}, \cdot, [\![\,]\!]>$ is a λ-model.

In constructing a complete system, the semantics of types plays a crucial role. As in [7], [12] and essentially following [9], 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*-, *simple*-, and F type interpretations. These different notions of type interpretations induce of course different notions of semantic satisfiability. In this paper we concentrate on the simple type interpretations.

Definition 6. Let $<\mathcal{D}, \cdot, \varepsilon>$ be a continuous lambda model. A mapping $v{:}\ \mathcal{T} \rightarrow \wp(\mathcal{D}) = \{X \mid X \subseteq \mathcal{D}\}$ is a *simple type interpretation* if and only if: $v(\sigma{\rightarrow}\tau) = \{d \mid \forall e\, [e \in v(\sigma) \Rightarrow d \cdot e \in v(\tau)]\}$, and $v(\sigma\cap\tau) = v(\sigma) \cap v(\tau)$.

This notion of type interpretation leads, naturally, to the following definition for semantic satisfiability (called *simple semantics*).

Definition 7. 1. Let $\mathcal{M} = <\mathcal{D}, \cdot, [\![\,]\!]>$ be a λ-model, and ξ a valuation. Then $[\![M]\!]_\xi{}^{\mathcal{M}} \in \mathcal{D}$ is the interpretation of M in $\mathcal{M}$ via ξ.

2. We define $\models$ by: (where $\mathcal{M}$ is a lambda model, ξ a valuation and v a simple type interpretation)
   (a) $\mathcal{M}, \xi, v \models M{:}\sigma \Leftrightarrow [\![M]\!]_\xi^{\mathcal{M}} \in v(\sigma)$.
   (b) $\mathcal{M}, \xi, v \models B \Leftrightarrow \mathcal{M}, \xi, v \models x{:}\sigma$ for every $x{:}\sigma \in B$.
   (c) $B \models M{:}\sigma \Leftrightarrow \forall \mathcal{M}, \xi, v\, [\mathcal{M}, \xi, v \models B \Rightarrow \mathcal{M}, \xi, v \models M{:}\sigma]$.

The method followed in [4] was to define a simple type interpretation v on the filter lambda model and to use it for the proof of completeness. The main result of that paper was obtained by proving:

Property 8 [4]. *1. Soundness.* $B \vdash M{:}\sigma \Rightarrow B \models M{:}\sigma$.
2. Completeness. $B \models M{:}\sigma \Rightarrow B \vdash M{:}\sigma$. ∎

Since the type interpretation v is simple, the results of [4] in fact show that type assignment in the BCD-system is complete with respect to simple type semantics. (Type assignment in the strict system as presented in [1] was proven in that paper to be complete with respect to the *inference* semantics.)

2 Essential type assignment for the lambda calculus

In this section we present the Essential Type Assignment System, a restricted version of the system presented in [4].

Compared to the BCD-system, the major feature of this system is a restricted version of the derivation rules, as well as a restriction of the set of types. It also forms a slight extension of the Strict Type Assignment System that was presented in [1]; the main difference is that that systems is not closed for η-reduction, whereas the system presented here is.

2.1 Essential type assignment

Strict types are the types that are strictly needed to assign a type to a term in the system as presented in [4] (see also [1]). In the set of strict types intersection type schemes and the type constant ω play a limited role. We will assume that ω is the same as an intersection over zero elements: if $n = 0$, then $\sigma_1 \cap \cdots \cap \sigma_n \equiv \omega$, so ω does not occur in an intersection subtype. Moreover, intersection type schemes (so also ω) occur in strict types only as subtypes at the left hand side of an arrow type scheme.

Definition 9 (cf. [1]). 1. $\mathcal{T}_s$, the set of *strict types*, is inductively defined by:
(a) All type-variables $\varphi_0, \varphi_1, \ldots \in \mathcal{T}_s$.
(b) If $\tau, \sigma_1, \ldots, \sigma_n \in \mathcal{T}_s$ $(n \geq 0)$, then $\sigma_1 \cap \cdots \cap \sigma_n \rightarrow \tau \in \mathcal{T}_s$.
2. $\mathcal{T}_S$ is defined by: If $\sigma_1, \ldots, \sigma_n \in \mathcal{T}_s$ $(n \geq 0)$, then $\sigma_1 \cap \cdots \cap \sigma_n \in \mathcal{T}_S$.
3. On $\mathcal{T}_S$, the relation $\leq_S$ is defined by:
(a) $\forall 1 \leq i \leq n\ (n \geq 1)\ [\ \sigma_1 \cap \cdots \cap \sigma_n \leq_S \sigma_i\ ]$.
(b) $\forall 1 \leq i \leq n\ (n \geq 0)\ [\ \sigma \leq_S \sigma_i\ ] \Rightarrow \sigma \leq_S \sigma_1 \cap \cdots \cap \sigma_n$.
(c) $\sigma \leq_S \tau \leq_S \rho \Rightarrow \sigma \leq_S \rho$.
4. We define $\leq_E$ on $\mathcal{T}_S$ like $\leq_S$, by adding an extra alternative.
(d) $\rho \leq_E \sigma\ \&\ \tau \leq_E \mu \Rightarrow \sigma \rightarrow \tau \leq_E \rho \rightarrow \mu$.
5. On $\mathcal{T}_S$, the relation $\sim_E$ is defined by: $\sigma \sim_E \tau \Leftrightarrow \sigma \leq_E \tau \leq_E \sigma$.

$\mathcal{T}_S$ may be considered modulo $\sim_E$. Then $\leq_E$ becomes a partial order, and in this paper we consider types modulo $\sim_E$.

Unless stated otherwise, if $\sigma_1 \cap \cdots \cap \sigma_n$ is used to denote a type, then all $\sigma_1, \ldots, \sigma_n$ are assumed to be strict. Notice that $\mathcal{T}_s$ is a proper subset of $\mathcal{T}_S$. The relations $\leq_S$, $\leq_E$, and $\sim_E$ are, as with $\leq$ in definition 2, extended to bases.

Definition 10. 1. If $B_1, \ldots, B_n$ are bases, then $\Pi\{B_1, \ldots, B_n\}$ is the basis defined as follows: $x{:}\sigma_1 \cap \cdots \cap \sigma_m \in \Pi\{B_1, \ldots, B_n\}$ if and only if $\{x{:}\sigma_1, \ldots, x{:}\sigma_m\}$ is the set of all statements about x that occur in $B_1 \cup \ldots \cup B_n$.
2. If B is a basis and $\sigma \in \mathcal{T}_S$, then $\mathcal{T}_{<B,\sigma>}$ is the set of all strict subtypes occurring in the pair $<B, \sigma>$.

For the relation $\leq_E$ the following properties hold:

Lemma 11. *1.* $\sigma \leq_S \tau \Rightarrow \sigma \leq_E \tau$.
2. $\varphi \leq_E \sigma \Leftrightarrow \sigma \equiv \varphi$. *So* $\{\sigma \mid \sigma \sim_E \varphi\} = \{\varphi\}$.
3. $\omega \leq_E \sigma \Leftrightarrow \sigma \equiv \omega$. *So* $\{\sigma \mid \sigma \sim_E \omega\} = \{\omega\}$.
4. $\sigma \rightarrow \tau \leq_E \rho \in \mathcal{T}_s \Leftrightarrow \exists \alpha \in \mathcal{T}_S, \beta \in \mathcal{T}_s\ [\rho \equiv \alpha \rightarrow \beta\ \&\ \alpha \leq_E \sigma\ \&\ \tau \leq_E \beta]$.
5. $\sigma_1 \cap \cdots \cap \sigma_n \leq_E \tau \in \mathcal{T}_s \Rightarrow \exists 1 \leq i \leq n\ [\sigma_i \leq_E \tau]$.

Proof. Easy. ∎

Lemma 12. *1.* $\sigma_1 \cap \cdots \cap \sigma_n \leq_E \tau \in \mathcal{T}_S \Rightarrow$
$\exists \tau_1, \ldots, \tau_m\ [\tau \equiv \tau_1 \cap \cdots \cap \tau_m\ \&\ \forall 1 \leq j \leq m\ \exists 1 \leq i \leq n\ [\sigma_i \leq_E \tau_j]]$.
2. $\sigma \leq_E \tau\ \&\ \sigma \in \mathcal{T}_s \Rightarrow \exists \tau_1, \ldots, \tau_m\ [\tau \equiv \tau_1 \cap \cdots \cap \tau_m\ \&\ \forall 1 \leq j \leq m\ [\sigma \leq_E \tau_j]]$.
3. $B' \leq_E B \leq_S \{x{:}\sigma\}\ \&\ \sigma \in \mathcal{T}_s \Rightarrow \exists \sigma' \in \mathcal{T}_s\ [B' \leq_S \{x{:}\sigma'\}\ \&\ \sigma' \leq_E \sigma]$.

Proof. Easy. ∎

The Essential Type Assignment System is constructed from the set of strict types, and a restriction of the derivation rules as in definition 3.1. In this way a syntax directed system is obtained, that satisfies the main properties of the BCD-system.

Definition 13. 1. *Essential type assignment* and *essential derivations* are defined by the following natural deduction system (where all types displayed are strict, except σ in the rules $(\rightarrow\mathrm{I})$ and $(\leq_{\mathrm{E}})$):

$$(\rightarrow\mathrm{I}):\ \frac{\begin{matrix}[x{:}\sigma]\\ \vdots\\ M{:}\tau\end{matrix}}{\lambda x.M{:}\sigma\rightarrow\tau}\ (a) \qquad (\rightarrow\mathrm{E}):\ \frac{M{:}\sigma_1\cap\cdots\cap\sigma_n\rightarrow\tau \quad N{:}\sigma_1 \ldots N{:}\sigma_n}{MN{:}\tau}\ (n\geq 0)$$

$$(\leq_{\mathrm{E}}):\ \frac{x{:}\sigma \quad \sigma\leq_{\mathrm{E}}\tau}{x{:}\tau}$$

(a) If $x{:}\sigma$ is the only statement about x on which $M{:}\tau$ depends.
If $M{:}\sigma$ is derivable from B using an essential derivation, we write $B \vdash_{\mathrm{e}} M{:}\sigma$.

2. We define $\vdash_{\mathrm{E}}$ by: $B \vdash_{\mathrm{E}} M{:}\sigma$ if and only if: there are $\sigma_1, \ldots, \sigma_n$ $(n \geq 0)$ such that $\sigma \equiv \sigma_1\cap\cdots\cap\sigma_n$ and for every $1 \leq i \leq n$ $B \vdash_{\mathrm{e}} M{:}\sigma_i$.

Like with the BCD-type assignment rules, the rules of the Essential Type Assignment System are generalized to terms containing $\perp$ of by allowing for the terms to be elements of that set. Notice that, because essential type assignment is almost syntax directed, if $\perp$ occurs in a term M and $B \vdash_{\mathrm{E}} M{:}\sigma$, then either $\sigma \equiv \omega$, or in the derivation for $M{:}\sigma$, $\perp$ appears in the right hand subterm of an application on which the rule $(\rightarrow\mathrm{E})$ is used with $n = 0$.

For this notion of type assignment, the following properties hold:

Lemma 14. *1.* $B \vdash_{\mathrm{e}} MN{:}\sigma \iff \exists\,\tau \in \mathcal{T}_{\mathrm{S}}\ [B \vdash_{\mathrm{e}} M{:}\tau\rightarrow\sigma\ \&\ B \vdash_{\mathrm{E}} N{:}\tau]$.
2. $B \vdash_{\mathrm{e}} \lambda x.M{:}\sigma \iff \exists\,\rho \in \mathcal{T}_{\mathrm{S}},\ \mu \in \mathcal{T}_{\mathrm{s}}\ [\sigma \equiv \rho\rightarrow\mu\ \&\ B \cup \{x{:}\rho\} \vdash_{\mathrm{e}} M{:}\mu]$.
3. $B \vdash_{\mathrm{E}} M{:}\sigma \iff \exists\,\sigma_1, \ldots, \sigma_n\ [\sigma \equiv \sigma_1\cap\cdots\cap\sigma_n\ \&\ \forall\, 1 \leq i \leq n\ [B \vdash_{\mathrm{e}} M{:}\sigma_i]]$.
4. $B \vdash_{\mathrm{E}} x{:}\sigma \iff \exists\,\rho \in \mathcal{T}_{\mathrm{S}}\ [x{:}\rho \in B\ \&\ \rho \leq_{\mathrm{E}} \sigma]$.
5. $B \vdash_{\mathrm{E}} M{:}\sigma\ \&\ B' \leq_{\mathrm{E}} B \Rightarrow B' \vdash_{\mathrm{E}} M{:}\sigma$.

Proof. Easy. ■

As in [1], it is possible to prove that the essential type assignment system satisfies the main properties of the BCD-system:

Theorem 15. *1.* $B \vdash_{\mathrm{E}} M{:}\sigma\ \&\ M =_{\beta} N \Rightarrow B \vdash_{\mathrm{E}} N{:}\sigma$.
2. $\exists\,B, \sigma\ [B \vdash_{\mathrm{E}} M{:}\sigma\ \&\ B, \sigma\ \omega\text{-free}] \iff M$ *has a normal form.*
3. $\exists\,B, \sigma\ [B \vdash_{\mathrm{E}} M{:}\sigma\ \&\ \sigma \not\equiv \omega] \iff M$ *has a head normal form.* ■

Although the rule $(\leq_{\mathrm{E}})$ is not allowed on all terms, we can prove the following:

Theorem 16. *1. If* $B \vdash_{\mathrm{E}} M{:}\sigma$, *and* $\sigma \leq_{\mathrm{E}} \tau$, *then* $B \vdash_{\mathrm{E}} M{:}\tau$.
2. If $B \vdash_{\mathrm{E}} M{:}\sigma$ *and* $M \rightarrow_{\eta} N$, *then* $B \vdash_{\mathrm{E}} N{:}\sigma$.

Proof. 1. By straightforward induction on $\vdash_{\mathrm{E}}$, using lemmas 11, 12, and 14.
2. The proof is given by induction on the definition of $\rightarrow_{\eta}$, of which only the part $\lambda x.Mx \rightarrow_{\eta} M$, where x does not occur free in M, is interesting; it follows from part (i). The other parts are dealt with by induction.

2.2 Soundness and completeness of essential type assignment

The Essential Type Assignment System is the nucleus of the BCD-system: we will show that, for any derivation in the BCD-system, it is possible to find an equivalent derivation in the essential system.

The proof is based on the fact that for every $\sigma \in \mathcal{T}$ there is a $\sigma^* \in \mathcal{T}_S$ such that $\sigma \sim \sigma^*$, and an approximation theorem.

Property 17 [9, 1]. *For every $\sigma \in \mathcal{T}$ there is a $\sigma^* \in \mathcal{T}_S$ such that $\sigma \sim \sigma^*$.*

The proof for the main theorem of this section is achieved by proving first, for terms in $\mathcal{N}$, if in the derivation the derivation-rule $(\leq)$ is used, then for that term a derivation in the essential system can be built for which basis and type in the conclusion are equivalent, and afterwards generalizing this result to arbitrary lambda terms.

Theorem 18. $B \vdash A{:}\sigma \;\Rightarrow\; \exists\, B', \sigma' \in \mathcal{T}_S\ [B' \vdash_E A{:}\sigma' \;\&\; \sigma' \sim \sigma \;\&\; B' \sim B]$.

Proof. By induction on the structure of terms in $\mathcal{N}$, using property 17. ■

As for the BCD-system, we can formulate the relation between types assignable to a lambda term and those assignable to its approximants as follows:

Theorem 19. $B \vdash_E M{:}\sigma \;\Leftrightarrow\; \exists\, A \in \mathcal{A}(M)\ [B \vdash_E A{:}\sigma]$.

Proof. $\Rightarrow$) By computability.

$\Leftarrow$) If $B \vdash_E A{:}\sigma$, then by the remark made after definition 13, $\perp$ appears only in subterms that are typed by ω. Since $A \in \mathcal{A}(M)$, there is an M' such that $M' =_\beta M$ and A matches M' except for occurrences of $\perp$. Then obviously $B \vdash_E M'{:}\sigma$, and by property 14.1 also $B \vdash_E M{:}\sigma$. ■

The relation between the two different notions of type assignment is formulated as follows:

Theorem 20. $B \vdash M{:}\sigma \;\Rightarrow\; \exists\, B', \sigma'\ [B' \vdash_E M{:}\sigma' \;\&\; \sigma \sim \sigma' \;\&\; B \sim B']$.

Proof. $B \vdash M{:}\sigma \;\Rightarrow (4)\quad \exists\, A \in \mathcal{A}(M)\ [B \vdash A{:}\sigma] \;\Rightarrow (18)$
$\exists\, A \in \mathcal{A}(M), B', \sigma' \in \mathcal{T}_S\ [B' \vdash_E A{:}\sigma' \;\&\; \sigma' \sim \sigma \;\&\; B' \sim B] \;\Rightarrow (19)$
$\exists\, B', \sigma' \in \mathcal{T}_S\ [B' \vdash_E M{:}\sigma' \;\&\; \sigma' \sim \sigma \;\&\; B' \sim B]$. ■
Notice that, in particular, $\sigma' \leq \sigma$ and $B \leq B'$.

For this essential system, it is possible to prove completeness of type assignment with respect to the simple type semantics the same way as done in [4]. Since such a proof would be obtained in exactly the same way as in [4], we will not present it here. Instead, we will prove a completeness result using results proven in [9]. In that paper, some restrictions of the BCD-system were investigated, and one of them proved to be essentially the same as the BCD-system.

Definition 21 [9]. 1. The set $\mathcal{T}_N$ of *normal intersection types* is defined by:

(a) Type-variables and ω are in $\mathcal{T}_N$.
(b) If $\sigma, \tau \in \mathcal{T}_N - \{\omega\}$, then $\sigma \cap \tau \in \mathcal{T}_N$.
(c) If $\sigma \in \mathcal{T}_N$, and $\tau \in \mathcal{T}_N$-$\{\omega$, intersections$\}$, then $\sigma \rightarrow \tau \in \mathcal{T}_N$.

2. On $\mathcal{T}_N$, the relation $\leq_N$ is defined as the restriction of $\leq$ to $\mathcal{T}_N$.

3. The notion of type assignment $\vdash_N$ is defined as $\vdash$, but by adding: All types are in $\mathcal{T}_N$, derivation rule $(\cap\mathrm{E})$ never immediately follows $(\cap\mathrm{I})$, and derivation rules $(\cap\mathrm{E})$ and $(\leq_N)$ are only used with atomic subjects.

It is straightforward to show that $\mathcal{T}_N = \mathcal{T}_{\mathrm{S}}$, and $\leq_N \; = \; \leq_{\mathrm{E}}$.

Proposition 22 (cf. [9]). *1. If $\sigma, \tau \in \mathcal{T}_{\mathrm{S}}$, then $\sigma \leq \tau \iff \sigma \leq_{\mathrm{E}} \tau$.*
2. Let * be defined as in property 17. $B \vdash M{:}\sigma \iff B^* \vdash_N M{:}\sigma^*$. ■

Theorem 23. Conservativity. *$B \vdash M{:}\sigma \Rightarrow B \vdash_{\mathrm{E}} M{:}\sigma$, for $B, \sigma \in \mathcal{T}_{\mathrm{S}}$.*

Proof. $B \vdash M{:}\sigma \Rightarrow (20) \quad \exists\, B', \sigma' \, [B' \vdash_{\mathrm{E}} M{:}\sigma' \,\&\, \sigma' \sim \sigma \,\&\, B \sim B'] \Rightarrow (22.1)$ $\exists\, B', \sigma' \, [B' \vdash_{\mathrm{E}} M{:}\sigma' \,\&\, \sigma' \leq_{\mathrm{E}} \sigma \,\&\, B \leq_{\mathrm{E}} B'] \Rightarrow (14.5 \,\&\, 16) \quad B \vdash_{\mathrm{E}} M{:}\sigma$. ■
The implication in the other direction also holds: If $B \vdash_{\mathrm{E}} M{:}\sigma$, then $B \vdash M{:}\sigma$.

Also, soundness and completeness of essential type assignment are easy to prove.

Theorem 24. *Soundness and completeness of essential type assignment. Let B and σ contain types in $\mathcal{T}_{\mathrm{S}}$. Then $B \vdash_{\mathrm{E}} M{:}\sigma \iff B \models M{:}\sigma$.*

Proof. $\Rightarrow$) $B \vdash_{\mathrm{E}} M{:}\sigma \Rightarrow B \vdash M{:}\sigma \Rightarrow (8.1) \quad B \models M{:}\sigma$.
$\Leftarrow$) $B \models M{:}\sigma \Rightarrow (8.2) \quad B \vdash M{:}\sigma \Rightarrow (23) \quad B \vdash_{\mathrm{E}} M{:}\sigma$. ■

Finally, we show that $\vdash_N$ and $\vdash_{\mathrm{E}}$ are equivalent.

Theorem 25. *$B \vdash_N M{:}\sigma \iff B \vdash_{\mathrm{E}} M{:}\sigma$.*

Proof. $\Rightarrow$) $B \vdash_N M{:}\sigma \Rightarrow (22.2) \quad B \vdash M{:}\sigma \Rightarrow (23) \quad B \vdash_{\mathrm{E}} M{:}\sigma$.
$\Leftarrow$) $B \vdash_{\mathrm{E}} M{:}\sigma \Rightarrow B \vdash M{:}\sigma \Rightarrow (22.2) \quad B \vdash_N M{:}\sigma$. ■

3 Principal type property for the essential system

It is well known that the Curry Type Assignment System also has the principal type property: M is typeable if and only if there are a basis P, and type π, such that $P \vdash M{:}\pi$, and: for every pair $<B, \sigma>$ such that $B \vdash M{:}\sigma$, there exists an operation O (from a specified set of operations) such that $O(<P, \pi>) = <B, \sigma>$. The type π is then called a 'principal type for M'. For Curry's system the operation O consists entirely of substitutions, i.e. operations that replace type-variables by types. Principal type schemes for Curry's system were defined in [8].

Since substitution is an easy operation, the set $\{<B, \sigma> \mid B \vdash M{:}\sigma\}$ can be computed in Curry's system easily from the principal pair for M. In [13] was shown that the BCD-system has the principal type property. This was achieved in that paper by defining a principal pair consisting of basis and type for terms in $\mathcal{N}$. Then three operations on pairs of basis and type – substitution, expansion and rise – were specified that were proved sound and sufficient to generate, for every term in $\mathcal{N}$, all possible pairs from its principal one. Using this technique – which is similar to the one used in [2] – in this section we will sketch the construction of the proof that the Essential Type Assignment System has the principal type property. For each lambda term the principal pair (of basis and type) will be defined. We will define three operations on pairs of basis and types, namely substitution, expansion, and lifting, that are correct and sufficient to generate all derivable pairs for lambda terms. First we will define a notion of principal pairs for terms in $\mathcal{N}$, and later generalize this to arbitrary lambda terms.

Principal pairs for the Essential Type Assignment System are defined by:

Definition 26. 1. Let $A \in \mathcal{N}$. $pp(A)$, the *principal pair* of A, is defined by:
   (a) $pp(\bot) = <\emptyset, \omega>$.
   (b) $pp(x) = <\{x{:}\varphi\}, \varphi>$.
   (c) If $A \neq \bot$, and $pp(A) = <P, \pi>$, then:
      i. If x occurs free in A, and $x{:}\sigma \in P$, then $pp(\lambda x.A) = <P\backslash x, \sigma\rightarrow\pi>$.
      ii. Otherwise $pp(\lambda x.A) = <P, \omega\rightarrow\pi>$.
   (d) If $pp(A_i) = <P_i, \pi_i>$, $1 \leq i \leq n$ (disjoint in pairs), then $pp(xA_1 \ldots A_n) = <\Pi\{P_1, \ldots, P_n, \{x{:}\pi_1\rightarrow\cdots\rightarrow\pi_n\rightarrow\varphi\}\}, \varphi>$, where φ is a type-variable that does not occur in $pp(A_i)$ for $1 \leq i \leq n$.
2. $PP = \{<P, \pi> \mid \exists A \in \mathcal{N}\ [pp(A) = <P, \pi>]\}$.

Notice that, if $<P, \pi> \in PP$, then $\pi \in \mathcal{T}_s$. The notion of principal pairs for terms in $\mathcal{N}$ will be generalized to arbitrary lambda terms in definition 33.

We will now present three different operations on pairs of $<$*basis*, *type*$>$, namely substitution, expansion, and lifting as defined in [2]. Substitution is normally defined on types as the operation that replaces type-variables by types. For strict types this definition would not be correct. For example, the replacement of φ by ω would transform $\sigma\rightarrow\varphi$ (or $\sigma\cap\varphi$) into $\sigma\rightarrow\omega$ ($\sigma\cap\omega$), which is not a strict type. Therefore, for strict types substitution is not defined as an operation that replaces type-variables by types, but as a mapping from types to types.

Definition 27 [2]. 1. The *substitution* $(\varphi \mapsto \alpha) : \mathcal{T}_S \rightarrow \mathcal{T}_S$, where φ is a type-variable and $\alpha \in \mathcal{T}_s \cup \{\omega\}$, is defined by:
   (a) $(\varphi \mapsto \alpha)(\varphi) = \alpha$.
   (b) $(\varphi \mapsto \alpha)(\varphi') = \varphi'$, if $\varphi \not\equiv \varphi'$.
   (c) $(\varphi \mapsto \alpha)(\sigma\rightarrow\tau) = \omega$, if $(\varphi \mapsto \alpha)(\tau) = \omega$.
   (d) $(\varphi \mapsto \alpha)(\sigma\rightarrow\tau) = (\varphi \mapsto \alpha)(\sigma) \rightarrow (\varphi \mapsto \alpha)(\tau)$, if $(\varphi \mapsto \alpha)(\tau) \not\equiv \omega$.
   (e) $(\varphi \mapsto \alpha)(\sigma_1\cap\cdots\cap\sigma_n) = (\varphi \mapsto \alpha)(\sigma_1{}')\cap\cdots\cap(\varphi \mapsto \alpha)(\sigma_m{}')$, where $\{\sigma_1{}', \ldots, \sigma_m{}'\} = \{\sigma_i \in \{\sigma_1, \ldots, \sigma_n\} \mid (\varphi \mapsto \alpha)(\sigma_i) \not\equiv \omega\}$.
2. If S_1 and S_2 are substitutions, then so is $S_1 \circ S_2$, where $S_1 \circ S_2(\sigma) = S_1(S_2(\sigma))$.
3. $S(B) = \{x{:}S(\alpha) \mid x{:}\alpha \in B\ \&\ S(\alpha) \not\equiv \omega\}$.
4. $S(<B, \sigma>) = <S(B), S(\sigma)>$.

The operation of expansion is an operation on types that corresponds to notions of expansion as defined in [13]. It is an operation on types, that deals with the replacement of (sub)types by an intersection of a number of copies of that type. In this process it can be that also other types need to be copied.

It is a complex operation, possibly affecting more types than just the one to be expanded occurs in. (For a clear discussion of the complexity of this operation, see [2].)

Definition 28 [2]. The *last type-variable* of a strict type is defined by:

1. The last type-variable of φ is φ.

2. The last type-variable of $\sigma_1 \cap \cdots \cap \sigma_n \rightarrow \tau$ is the last type-variable of τ.

Definition 29 [2]. For every $\mu \in \mathcal{T}_s$, $n \geq 2$, basis B and $\sigma \in \mathcal{T}_S$, the quadruple $<\mu, n, B, \sigma>$ determines an *expansion* $E_{<\mu,n,B,\sigma>} : \mathcal{T}_S \rightarrow \mathcal{T}_S$, that is constructed as follows.

1. The set of type-variables $\mathcal{V}_\mu(<B, \sigma>)$ is constructed by:
   (a) If φ occurs in μ, then $\varphi \in \mathcal{V}_\mu(<B, \sigma>)$.
   (b) If the last type-variable of $\tau \in \mathcal{T}_{<B,\sigma>}$ is in $\mathcal{V}_\mu(<B, \sigma>)$, then for all type-variables φ that occur in τ: $\varphi \in \mathcal{V}_\mu(<B, \sigma>)$.
2. Suppose $\mathcal{V}_\mu(<B, \sigma>) = \{\varphi_1, \ldots, \varphi_m\}$. Choose $m \times n$ different type-variables $\varphi_1^1, \ldots, \varphi_1^n, \ldots, \varphi_m^1, \ldots, \varphi_m^n$, such that each φ_j^i does not occur in $<B, \sigma>$, for $1 \leq i \leq n$ and $1 \leq j \leq m$. Let S_i be such that $S_i(\varphi_j) = \varphi_j^i$.
3. $E_{<\mu,n,B,\sigma>}(\tau)$ is obtained by traversing τ top-down and replacing every subtype α by $S_1(\alpha) \cap \cdots \cap S_n(\alpha)$, if the last type-variable of α is in $\mathcal{V}_\mu(<B, \sigma>)$.
4. $E_{<\mu,n,B,\sigma>}(B') = \{x{:}E_{<\mu,n,B,\sigma>}(\rho) \mid x{:}\rho \in B'\}$.
5. $E_{<\mu,n,B,\sigma>}(<B', \sigma'>) = <E_{<\mu,n,B,\sigma>}(B'), E_{<\mu,n,B,\sigma>}(\sigma')>$.

The last operation on pairs defined in this subsection is the operation of lifting.

Definition 30 [2]. A *lifting* L is an operation denoted by a pair of pairs $<<B_0, \tau_0>, <B_1, \tau_1>>$ such that $\tau_0 \leq_E \tau_1$ and $B_1 \leq_E B_0$, and is defined by:

1. $L(\sigma) = \tau_1$, if $\sigma \equiv \tau_0$; $L(\sigma) = \sigma$, otherwise.
2. $L(B) = B_1$, if $B = B_0$; $L(B) = B$, otherwise.
3. $L(<B, \sigma>) = <L(B), L(\sigma)>$.

Definition 31 [2]. A *chain* is an object $<O_1, \ldots, O_n>$, where each O_i is an operation of substitution, expansion or lifting, and

1. $<O_1, \ldots, O_n>(\sigma) = O_n(\ldots(O_1(\sigma))\ldots)$.
2. $<O_1, \ldots, O_n>(<B, \sigma>) = O_n(\cdots(O_1(<B, \sigma>))\cdots)$.

It is possible to show that all pairs for a term can be generated by chains that exist of a number of expansions, and that end with one substitution and one lifting (in that order). Moreover, all three operations can be proven to be sound on all pairs. The technique needed for these proofs can be found in [13] and [2]; because of the direct similarities with especially the proofs in the latter paper, proofs will be omitted here.

Theorem 32. *Let $A \in \mathcal{N}$, $<B, \sigma>$ be such that $B \vdash_E A{:}\sigma$, and let O be an operation of substitution, expansion or rise. Then $O(B) \vdash_E A{:}O(\sigma)$.* ■

Definition 33 (cf. [13, 2]). 1. *Linear chains* of operations are defined as chains that start with a number of expansions, followed by a substitution, and that end with a lifting.

2. Let M be a term. Let $\Pi(M)$ be the set of all principal pairs for all approximants of M: $\Pi(M) = \{pp(A) \mid A \in \mathcal{A}(M)\}$.
3. On *PP* it is possible to define the preorder relation $\sqsubseteq_w$ by:
   $<P, \pi> \sqsubseteq_w <P', \pi'> \Leftrightarrow$
   $\exists \varphi_1, \ldots, \varphi_n\, [<P, \pi> = (\varphi_1 := \omega) \circ \cdots \circ (\varphi_n := \omega)(<P', \pi'>)]$,

and PP, $\sqsubseteq_w$ is a meet semilattice isomorphic to $\mathcal{N}$, $\leq$.
4. $\Pi(M)$ is an ideal in PP and therefore:
   (a) If $\Pi(M)$ is finite, then there exists a pair $<P, \pi> = \bigsqcup \Pi(M)$, where $<P, \pi> \in PP$. This pair is then called the principal pair of M.
   (b) If $\Pi(M)$ is infinite, $\bigsqcup \Pi(M)$ does not exist in PP. The principal pair of M is then the infinite set of pairs $\Pi(M)$.

The proof of the principal type property is completed by proving the following:

Theorem 34. *1. Let $A \in \mathcal{N}$ and $<P, \pi>$ be the principal pair for A. For any pair $<B, \sigma>$ such that $B \vdash_{\mathrm{E}} A{:}\sigma$ there exists a linear chain* C *such that* $\mathrm{C}\,(<P, \pi>) = <B, \sigma>$.
2. (a) $\mathcal{A}(M)$ is finite. Let $<P, \pi>$ be the principal pair of M. Then there exists a chain C *such that* $\mathrm{C}\,(<P, \pi>) = <B, \sigma>$.
(b) $\mathcal{A}(M)$ is infinite. Then there exist a pair $<P, \pi> \in \Pi(M)$ and a chain C *such that* $\mathrm{C}\,(<P, \pi>) = <B, \sigma>$.

References

[1] Bakel S. van. Complete restrictions of the Intersection Type Discipline. *Theoretical Computer Science*, 102:135–163, 1992.
[2] Bakel S. van. Principal type schemes for the Strict Type Assignment System. *Logic and Computation*, 1993. To appear.
[3] Barendregt H. *The Lambda Calculus: its Syntax and Semantics*. North-Holland, Amsterdam, revised edition, 1984.
[4] Barendregt H., M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *The Journal of Symbolic Logic*, 48(4):931–940, 1983.
[5] 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.
[6] Curry H.B. and R. Feys. *Combinatory Logic*. volume 1. North-Holland, Amsterdam, 1958.
[7] Dezani-Ciancaglini M. and I. Margaria. A characterisation of F-complete type assignments. *Theoretical Computer Science*, 45:121–157, 1986.
[8] Hindley J.R. The principal type scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60, 1969.
[9] 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.
[10] Hindley J.R. The Completeness Theorem for Typing λ-terms. *Theoretical Computer Science*, 22(1):1–17, 1983.
[11] Hindley R. and G. Longo. Lambda calculus models and extensionality. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 26:289–310, 1980.
[12] Mitchell J.C. Polymorphic Type Inference and Containment. *Information and Computation*, 76:211–249, 1988.
[13] Ronchi della Rocca S. and B. Venneri. Principal type schemes for an extended type theory. *Theoretical Computer Science*, 28:151–169, 1984.

This article was processed using the LaTeX macro package with LLNCS style