Principal Type Schemes for the Strict Type Assignment System

STEFFEN VAN BAKEL, *Department of Informatics, Faculty of Mathematics and Informatics, University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands. E-mail: steffen@cs.kun.nl*

Abstract

We study the strict type assignment system, a restriction of the intersection type discipline [6], and prove that it has the principal type property. We define, for a term M, the principal pair (of basis and type). We specify three operations on pairs, and prove that all pairs deducible for M can be obtained from the principal one by these operations, and that these map deducible pairs to deducible pairs.

Keywords: Lambda Calculus, type assignment systems, principal type scheme, intersection types, approximate normal forms

1 Introduction

There are various ways to deal with the problem of handling types in programming languages, that can roughly be divided in the 'typed'- and 'untyped'-approaches. The 'typed'-approach can be found in programming languages that have explicit typing: objects in a program have types that are provided by the programmer, and the type-algorithm incorporated in the compiler for the language checks if these are used consistently. The 'untyped'-approach is used in languages that allow programmers to write programs without any type-specification at all, and it is the task of the type-algorithm to infer types for objects and to check consistency. Many of the now existing type assignment systems for functional programming languages – that use the 'untyped'-approach – are based on (extensions of) the Curry type assignment system [11, 12] for the pure, untyped lambda calculus [5]. For example, the functional programming language ML [19] is in fact an extended lambda calculus and its type system is based on Curry's system.

It is well known that in Curry's system, the problem of typeability

> Given a term M, are there B and σ such that $B \vdash M{:}\sigma$ (i.e. M can be typed with the type σ starting from a basis B)?

is decidable. This system also has the principal type property: M is typeable if and only if there is a pair $\langle P, \pi\rangle$ of basis and type, called the principal pair for M, such that:

1. $P \vdash M{:}\pi$, and
2. for every pair $\langle B, \sigma\rangle$ such that $B \vdash M{:}\sigma$, there exists an operation O (from a specified set of operations) such that $O(\langle P, \pi\rangle) = \langle B, \sigma\rangle$.

The type π is then called the 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 are defined in [15].) The existence of a principal type for a typeable lambda term M shows an internal coherence between all types that can be assigned to M. Since substitution is an easy operation, in Curry's system the set

$$\{\langle B, \sigma\rangle \mid B \vdash M{:}\sigma\}$$

can be computed in a simple way from the principal pair for M.

The principal type property plays an important role in the 'untyped'-approach, since, in an implementation, it allows to use the principal type of M to find the right types for the various occurrences of M without deriving the type for M over and over again; this property forms the basis for the notion of 'polymorphic functions' in programming languages like ML. There exist type assignment systems for which it is uncertain whether or not this property holds. For example, in the polymorphic type discipline [14], there is no known way to obtain the types $(\forall\varphi.\varphi)\rightarrow(\forall\varphi.\varphi)$ and $(\forall\varphi.\varphi\rightarrow\varphi)\rightarrow(\forall\varphi.\varphi\rightarrow\varphi)$ – both types for the lambda term $\lambda x.xx$ – from a unique type. (See [13].) Moreover, there exists no type σ derivable for $\lambda x.xx$ such that both types can be obtained from σ by substitution.

Although frequently used as a basis for type assignment algorithms for functional programming languages, the Curry type assignment system has drawbacks. It is, in this system, for example not possible to assign a type to the term $(\lambda x.xx)$, and although the lambda terms $(\lambda cd.d)$ and $((\lambda xyz.xz(yz))(\lambda ab.a))$ are β-equal, the principal type schemes for these terms are different.

The intersection type discipline as presented in [9] (see also [6], and [1]) is an extension of Curry's system that does not have these drawbacks. The extension to Curry's system is essentially that terms and term-variables are allowed to have more than one type. Intersection types are constructed by adding, next to the type constructor '$\rightarrow$', the type constructor '$\cap$' and the type constant 'ω'. By introducing this extension a system is obtained that is closed under β-equality: if $B \vdash M{:}\sigma$ and $M =_\beta N$, then $B \vdash N{:}\sigma$. The first type assignment system with intersection types was presented in [7]; the CDV-system with intersection types and ω is introduced in [9] and in [23]. The best-known intersection type assignment system is the BCD-system, as presented in [6], that is an extension of the CDV-system: there it 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 $(\leq)$ is introduced mainly to prove completeness of type assignment.

In [1] two independent restrictions of the BCD-system are presented. The first is the strict type assignment system, a type assignment system in which the $\leq$-relation and the derivation rule $(\leq)$ are no longer present. (The second system is the type assignment system without ω, and the main result for that system is that a lambda term is typeable in it, if and only if that term is strongly normalizable.) The elimination of $\leq$ induces a set of strict types, a restriction of the set of types used in the BCD-system. The strict type assignment system is constructed from the set of strict types and a slight extension of the derivation rules as defined in [9]. This strict system is essentially equivalent to the BCD-system, so it satisfies the main properties of that system: type assignment is closed under β-equality, the set of terms typeable with type σ from a basis B such that ω does not occur in B and σ is the set of normalizable terms, and the set of terms typeable with type $\sigma \neq \omega$ is the set of terms having a head normal form. In [1] it is shown that the strict system is the kernel of the BCD-system: strict types are the types that are strictly needed to assign a type to a term in that system. The most significant difference between the BCD-system and the strict one is that the former is closed for η-reduction, whereas the latter is not. The main result proved for the strict system in [1] is that of completeness of type assignment without the $\leq$-relation, by using inference type semantics as defined in [20] instead of the simple type semantics as defined in [16].

As stated above, if for the construction of a type inference system for an untyped functional programming language instead of Curry's system a type assignment system with intersection types is to be used, then such a system should at least have the principal pair property. There exists two of those systems for which this property is proved.

In [8] principal type schemes are defined for a type assignment system that is a restriction of the system as presented in [9]. This system has as a disadvantage that it is not an extension of Curry's system: if $B \vdash M{:}\tau$, and the term-variable x does not occur in B, then for $\lambda x.M$ only the type $\omega\rightarrow\tau$ can be derived. Therefore, in this system it is impossible to derive $\varphi_0\rightarrow\varphi_1\rightarrow\varphi_0$ for the lambda term $\lambda ab.a$. That type is

derivable for that term in Curry's system. (There is one important remark to be made. On page 540 of [8] it is suggested that it is straightforward to extend the results of that paper to an intersection type assignment system that is a true extension of Curry's system. The results presented in this paper show that this is not the case.)

For the system as defined in [6], principal type schemes can be defined as in [22]. There three operations are provided – substitution, expansion, and rise – that are sound and sufficient to generate all suitable pairs for a term M from its principal pair. The BCD-system, however, has as a disadvantage 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 superfluous steps in derivations, but it is also possible to give essentially different derivations for the same result. In the strict type assignment system these rules are not present and there is a one-one relationship between terms and skeletons of derivations: that system is syntax directed.

Moreover, in [6] the relation $\leq$ induces an equivalence relation $\sim$ on types. Since equivalence classes are big (for example: $\omega \sim \sigma \to \omega$, for all types σ), and type assignment is closed for $\sim$, although for every M the set $\{\langle B, \sigma\rangle \mid B \vdash M{:}\sigma\}$ can be generated using the three specified operations, the problem of type-checking

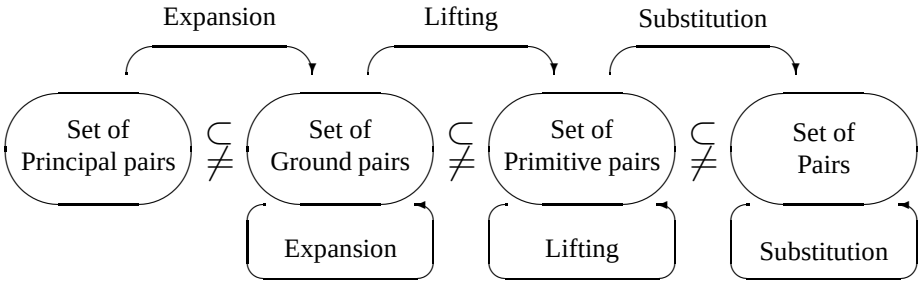
Given a term M and type σ, is there a B such that $B \vdash M{:}\sigma$?

is complicated.

The set of types assignable to a term M in the strict system is significantly smaller than the set of types assignable to M in the BCD-system. In fact, the results of [1] show that the strict type assignment system is the 'minimal' intersection type assignment system. In particular, the problem of type checking for the strict system is less complicated than for the BCD-system. As shown in Section 3 of this paper, the results of [8] and [22] do not provide principal types for the strict system. The main result of this paper – the proof of the principal type property for the strict system – is then, that if one wants to use intersection types in a type-checker, there is no need to restrict the types as in [8], or to extend the types as in [6]; one can use the strict types. The results of this paper can be used as a guideline to develop type-inferencing (and type-checking) algorithms using strict types in functional programming languages. Since the strict system is closed under β-equality, type assignment in this system is not decidable, but semi-decidable. This means of course that in an implementation of the strict system some restrictions have to be made, as is done for example in [10] and [2]. Limiting the notion of type assignment by restricting the situations in which the derivation rules can be applied (the technique used in [10]), or by limiting the structure of types (as is done in [2]) changes the notion of type assignment, and, therefore, also the principal types for these systems differ.

The general outline of this paper is as follows: in Section 2 we present the strict type assignment system. In Section 4 we define three operations on pairs of basis and types: substitution (Subsection 4.1), expansion (Subsection 4.2), and lifting (Subsection 4.3). The operation of lifting is new and forms the strict counterpart of the operation of rise as defined in [22]. The operation of substitution is a modification of the one normally used. The operation of expansion coincides with the one given in [8, 22].

In order to prove that the operations defined are sufficient, we define three subsets of the set of all pairs of basis and type, namely: principal pairs (Section 3), ground pairs (Subsection 4.2) and primitive pairs (Subsection 4.3). (The definition of ground pairs coincides with the one given in [8].) We show that these form a true hierarchy, that the set of admissible ground pairs for a term is closed under the operation of expansion, that the set of primitive pairs is closed under the operation of lifting, and that the set of pairs is closed for substitution. In a diagram, this construction looks like:



There is an arrow from one set of pairs to another, if the operation mentioned above the arrow maps a pair in the set at the beginning of the arrow to a pair in the set at the end. When these sets are different, in general the result of applying the operation is not a pair in the set at the beginning of the arrow. The symbols between the sets indicate that the set on the left of them is a true subset of the set on the right.

In Subsection 5.1 the main result of this paper is reached by showing that the three operations defined in Section 4 are complete: if $\langle B, \sigma\rangle$ is a suitable pair for a term A in $\lambda\perp$-normal form, and $\langle P, \pi\rangle$ is the principal pair for A, then there are an operation of lifting L, and sequences of operations of expansion $\vec{E}$, and substitution $\vec{S}$, such that

$$\langle B, \sigma\rangle = \vec{S}(L(\vec{E}(\langle P, \pi\rangle))).$$

Finally, in Subsection 5.2 we generalize this result to arbitrary lambda terms.

We assume the reader to be familiar with the lambda calculus. For definitions and notions used here, see [5]. In this paper, the symbol φ (often indexed, like in φ_i) will be a type-variable and the greek symbols like μ, ν, η, ρ, σ, and τ will range over types. To avoid parentheses in the notation of types, we will assume that '$\rightarrow$' associates to the right, so right-most, outer-most brackets will be omitted. We will also assume that, as in logic, '$\cap$' binds stronger than '$\rightarrow$', so $\alpha\cap\beta\rightarrow\gamma\cap\delta\rightarrow\eta$ means $((\alpha\cap\beta)\rightarrow((\gamma\cap\delta)\rightarrow\eta))$. In this paper we use the word 'subtype' for a type that is a syntactic component of another type.

2 The Strict Type Assignment System

In this section we present the strict type assignment system, as defined in [1]. It is an extension of Curry's system, and a restricted version of the BCD-system as presented in [6], in which the derivation rule ($\leq$) is no longer present, together with a restricted set of types. It is also a generalization of the CDV-system presented in [8].

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

In the rest of this paper we define that the type ω is the same as an intersection over zero elements: if $n = 0$, then $\sigma_1\cap\cdots\cap\sigma_n = \omega$. We could have omitted the type constant ω completely from the presentation of the system, because we can always assume that $n = 0$ in $\sigma_1\cap\cdots\cap\sigma_n$, but some of the definitions and the results we obtain are more clear when ω is dealt with explicitly.

DEFINITION 2.1

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 $\sigma_1 \cap \cdots \cap \sigma_n \to \tau \in \mathcal{T}_s$.

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) $\sigma \leq_S \sigma$.
   (b) $\sigma \leq_S \omega$.
   (c) $\sigma \cap \tau \leq_S \sigma \ \& \ \sigma \cap \tau \leq_S \tau$.
   (d) $\sigma \leq_S \tau \leq_S \rho \Rightarrow \sigma \leq_S \rho$.
   (e) $\sigma \leq_S \rho \ \& \ \sigma \leq_S \tau \Rightarrow \sigma \leq_S \rho \cap \tau$.
4. On $\mathcal{T}_S$, the relation $\sim_S$ is defined by:
   (a) $\sigma \leq_S \tau \leq_S \sigma \Rightarrow \sigma \sim_S \tau$.
   (b) $\sigma \sim_S \rho \ \& \ \tau \sim_S \mu \Rightarrow \sigma \to \tau \sim_S \rho \to \mu$.
5. A *statement* is an expression of the form $M{:}\sigma$, where M is a lambda term (notation: $M \in \Lambda$) and $\sigma \in \mathcal{T}_S$. M is the *subject* and σ the *predicate* of $M{:}\sigma$.
6. A *basis* is a set of statements with only distinct term-variables as subjects; if $\sigma_1 \cap \cdots \cap \sigma_n$ is a predicate in a basis, then $n \geq 1$.
7. We say that two types (bases, pairs of basis and type) are *disjoint* if and only if they have no type-variables in common.

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

In this paper we consider types modulo $\sim_S$, so for example $\rho \cap (\sigma \cap \tau) = (\rho \cap \sigma) \cap \tau$. 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 $\mathcal{T}_S$. We could have taken another approach for the representation of types and write $\{\sigma_1, \ldots, \sigma_n\}$ for the type $\sigma_1 \cap \cdots \cap \sigma_n$ ($\emptyset$ for ω). But that would mean that reasoning about types would become more complicated, since there would be an extra syntactical construct that has to be dealt with. Moreover, the strict system is defined as a restricted version of the BCD-system; by using the notations given above it is more clear that the set of strict types $\mathcal{T}_s$ (as well as $\mathcal{T}_S$) is a true subset of the set of types in the BCD-system.

DEFINITION 2.2

1. *Strict type assignment* and *strict derivations* are defined by the following natural deduction system (where all types displayed are strict, except σ in the derivation rule (→I)):

$$(\to\text{I}): \quad \frac{\begin{array}{c}[x{:}\sigma] \qquad (\sigma \in \mathcal{T}_S)\\ \vdots \\ M{:}\tau\end{array}}{\lambda x.M{:}\sigma \to \tau}\ \text{(a)} \qquad\qquad (\cap\text{E}): \quad \frac{x{:}\sigma_1 \cap \cdots \cap \sigma_n}{x{:}\sigma_i} \quad (n \geq 2)$$

$$(\to\text{E}): \quad \frac{M{:}\sigma_1 \cap \cdots \cap \sigma_n \to \tau \qquad N{:}\sigma_1 \ \ldots \ N{:}\sigma_n}{MN{:}\tau} \quad (n \geq 0)$$

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

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

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

If $B \vdash_S M{:}\sigma$, and $B = \emptyset$, we write $\vdash_S M{:}\sigma$.

In $B \vdash_{\mathrm{s}} M{:}\sigma$ the basis can contain types that are not strict, but are intersections of strict types. Therefore, if a statement $x{:}\sigma$ is used to derive $M{:}\tau$, then σ in the ($\rightarrow$I)-rule is a strict type, or an intersection of strict types. If no statement with subject x is used to derive $M{:}\tau$, then σ can be any element of $\mathcal{T}_{\mathrm{S}}$, including ω.

The introduction of two different notions of derivability seems somewhat superfluous. Notice that we could have limited ourselves to one, by omitting the definition of $\mathcal{T}_{\mathrm{S}}$ and stating:

> We define $\vdash_{\mathrm{s}}$ by: $B \vdash_{\mathrm{s}} M{:}\sigma$ if and only if there are $\sigma_1, \ldots, \sigma_n$ $(n \geq 0)$ such that $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$ and for every $1 \leq i \leq n$, $M{:}\sigma_i$ 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. Notice, moreover, that the notion $\vdash_{\mathrm{S}}$ can be seen as a short-hand notation for any number of derivations in $\vdash_{\mathrm{s}}$; this aspect is very convenient in reasoning about possible type assignments for, for example, applications. For these reasons, we prefer two different notions of derivability.

The difference between the strict system and the BCD-system is essentially this: in the BCD-system, the derivation rule ($\cap$E) is allowed on all terms, whereas in the strict system it is only performed on term-variables. Moreover, the BCD-system has the derivation rules (ω) and ($\cap$I), also allowed on all terms, which are implicitly present in the derivation rule ($\rightarrow$E) of the strict system.

This difference in use and definitions of type assignment rules has no effect on the set of typeable terms: for the strict and for the full system these are the same. The most significant difference is that the BCD-system is closed for η-reduction, i.e. if $B \vdash M{:}\sigma$ and $M \rightarrow_\eta N$, then $B \vdash N{:}\sigma$, whereas this property does not hold for the strict system. As a counter example, take the lambda term $\lambda xy.xy$, and notice that $\lambda xy.xy \rightarrow_\eta \lambda x.x$. For the first term we can derive $\vdash_{\mathrm{s}} \lambda xy.xy{:}(\sigma \rightarrow \tau) \rightarrow \sigma \cap \rho \rightarrow \tau$. That type cannot be derived for $\lambda x.x$ in the strict system. (This difference plays an important role in Subsection 4.3.)

We give some of the most important properties of the strict system. They are given as an illustration of the power of the system but do not play a role in this paper.

PROPERTY 2.3 ([1])

1. If $M =_\beta N$ and $B \vdash_{\mathrm{s}} M{:}\sigma$, then $B \vdash_{\mathrm{s}} N{:}\sigma$, so the following rule is a derived rule in $\vdash_{\mathrm{s}}$:

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

2. $\exists\, B, \sigma\, [\, B \vdash_{\mathrm{s}} M{:}\sigma \;\&\; \omega \text{ not in } B, \sigma\,] \iff M$ has a normal form.
3. $\exists\, B, \sigma\, [\, B \vdash_{\mathrm{s}} M{:}\sigma\,] \iff M$ has a head normal form.

For the notions of type assignment as defined in Definition 2.2, the following properties hold:

LEMMA 2.4 ([1])

1. If $B \vdash_{\mathrm{S}} M{:}\sigma$ and $\sigma \leq_{\mathrm{S}} \tau$, then $B \vdash_{\mathrm{S}} M{:}\tau$.
2. $B \vdash_{\mathrm{s}} MN{:}\sigma \iff \exists\, \tau\, [\, B \vdash_{\mathrm{s}} M{:}\tau \rightarrow \sigma \;\&\; B \vdash_{\mathrm{S}} N{:}\tau\,]$.
3. $B \vdash_{\mathrm{s}} M{:}\sigma \iff B \vdash_{\mathrm{S}} M{:}\sigma \;\&\; \sigma \in \mathcal{T}_{\mathrm{s}}$.
4. $B \vdash_{\mathrm{s}} \lambda x.M{:}\sigma \iff \exists\, \rho \in \mathcal{T}_{\mathrm{S}}, \mu \in \mathcal{T}_{\mathrm{s}}\, [\, \sigma = \rho \rightarrow \mu \;\&\; B \cup \{x{:}\rho\} \vdash_{\mathrm{s}} M{:}\mu\,]$.

The following definition introduces some terminology and notations for bases.

DEFINITION 2.5

1. We extend the relation $\leq_{\mathrm{S}}$ to bases by: $B \leq_{\mathrm{S}} B'$ if and only if for every $x{:}\sigma' \in B'$ there is an $x{:}\sigma \in B$ such that $\sigma \leq_{\mathrm{S}} \sigma'$.
2. If a basis is denoted as $B \cup \{x{:}\sigma\}$, then B is a basis, and x does not occur in B.
3. $B \backslash x$ is the basis defined by: if $x{:}\sigma \in B$, take $B \backslash \{x{:}\sigma\}$, otherwise take B.

4. 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 whose subject is x that occur in $B_1 \cup \cdots \cup B_n$.
5. If B is a basis and $\sigma \in \mathcal{T}_S$, then $\mathcal{T}_{\langle B,\sigma\rangle}$ is the set of all strict subtypes occurring in the pair $\langle B, \sigma\rangle$.

Notice that if $n = 0$, then $\Pi\{B_1, \ldots, B_n\} = \emptyset$.

By abuse of notation, we sometimes write a basis as $B \cup \{x{:}\sigma\}$, where $\sigma = \omega$. We then assume that $B \cup \{x{:}\sigma\} = B$.

LEMMA 2.6
If $B \vdash_S M{:}\sigma$ and $B' \leq_S B$, then $B' \vdash_S M{:}\sigma$.

PROOF. Since by Lemma 2.4 (1), for every $x{:}\tau \in B$: if $B' \leq_S B$, then $B' \vdash_S x{:}\tau$. ■

In the following definition we introduce the notion of a basis used for a statement. The idea is that in an average derivation, some types attached to term-variables in the basis are not needed in the derivation at all: there is for example no ($\cap$E)-rule that selects these types. In constructing a used basis of a derivation, we collect all (and nothing but) the types that are actually used in the derivation.

DEFINITION 2.7
1. The set of *used bases of* $B \vdash_S M{:}\sigma$ is inductively defined by:
   (a) If $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$ ($n \geq 0$), then for every $1 \leq i \leq n$, $B \vdash_s M{:}\sigma_i$. Let for every $1 \leq i \leq n$, $\mathcal{B}_i$ be the set of used bases of $B \vdash_s M{:}\sigma_i$.
   Take $\{\Pi\{B_1, \ldots, B_n\} \mid \forall\, 1 \leq i \leq n\, [\, B_i \in \mathcal{B}_i\,]\}$.
   (b) $\sigma \in \mathcal{T}_s$.
      i. $M \equiv x$. Take $\{\{x{:}\sigma\}\}$.
      ii. $M \equiv \lambda x.M'$. Then $\sigma = \alpha{\to}\beta$, and $B \cup \{x{:}\alpha\} \vdash_S M'{:}\beta$. Let $\mathcal{B}$ be the set of used bases of $B \cup \{x{:}\alpha\} \vdash_S M'{:}\beta$. Take $\{B\backslash x \mid B \in \mathcal{B}\}$.
      iii. $M \equiv M_1 M_2$. Then there is a τ such that $B \vdash_s M_1{:}\tau{\to}\sigma$ and $B \vdash_S M_2{:}\tau$. Take $\{\Pi\{B_1, B_2\} \mid \exists\, \tau \in \mathcal{T}_S\, [\, B_1$ is a used basis of $B \vdash_s M_1{:}\tau{\to}\sigma$ and B_2 is a used basis of $B \vdash_S M_2{:}\tau\,]\}$.
2. A basis B is *used for* $M{:}\sigma$ if and only if there is a basis B' such that $B' \vdash_S M{:}\sigma$ and B is a used basis of $B' \vdash_S M{:}\sigma$.

Notice that in part (1.a), if $n = 0$, then $\sigma = \omega$, and $\Pi\{B_1, \ldots, B_n\} = \emptyset$.

Notice that constructing a used basis from a derivation is not the same as constructing the minimal basis needed to derive the conclusion of the derivation. Take for example the derivation $\{x{:}\sigma\cap\rho,\, z{:}\tau\} \vdash_s (\lambda y.x)z{:}\sigma$. A used basis for this derivation is $\{x{:}\sigma,\, z{:}\tau\}$, whereas the minimal one is $\{x{:}\sigma\}$. We need to collect all types that were essential in the derivation, not merely the minimal set possible.

Notice that $\{x{:}\sigma\}$ is also a used basis for this derivation, so a used basis for a derivable statement $M{:}\sigma$ is not unique, but depends on the derivation used. This is caused by the fact that in part (1.b.iii) the type τ is not fixed. Choosing another suitable τ in general gives another used basis. The results of this paper we present with used bases however do not depend on the actual structure of a used basis, only on its existence.

For used bases, the following properties hold.

LEMMA 2.8
1. If B is used for $M{:}\sigma$, then $B \vdash_S M{:}\sigma$.
2. $B \vdash_S M{:}\sigma \;\Leftrightarrow\; \exists\, B'\, [\, B \leq_S B' \;\&\; B'$ is used for $M{:}\sigma\,]$.
3. B is used for $\lambda x.M{:}\sigma{\to}\tau \;\Leftrightarrow\; \exists\, \sigma'\, [\, \sigma \leq_S \sigma' \;\&\; B \cup \{x{:}\sigma'\}$ is used for $M{:}\tau\,]$.

4. B is used for $xM_1 \dots M_n{:}\sigma \iff \exists\, B_1, \dots, B_n, \sigma_1, \dots, \sigma_n$ [$\forall\, 1 \leq i \leq n$ [B_i is used for $M_i{:}\sigma_i$] & $B = \Pi\{B_1, \dots, B_n, \{x{:}\sigma_1 \to \cdots \to \sigma_n \to \sigma\}\}$].

PROOF. By Lemmas 2.4 and 2.6, and Definition 2.7. ■

3 Principal pairs for terms in $\lambda\perp$-normal form

In [8] principal pairs are defined for a type assignment system similar to the strict system. In [22] principal pairs are defined for the BCD-system. In order to understand the necessity of defining principal pairs for the strict system, we focus on the differences between the latter and the other two.

In the CDV-system as presented in [8], instead of using intersections in bases, it is allowed to let a basis contain several statements whose subject is a term-variable. If $B \vdash M{:}\tau$ and x occurs in B, the ($\to$I)-rule of that system collects all types that are predicates for x and that are used in the derivation of $M{:}\tau$. This system does not contain an ($\cap$E)-rule of any kind, so in that system it is impossible to derive the type $(\varphi_0 \cap \varphi_1) \to \varphi_0$ for the lambda term $\lambda x.x$. That type is derivable for that term in the strict system. Moreover, the CDV-system is not a true extension of Curry's type assignment system: if $B \vdash M{:}\tau$, and the term-variable x does not occur in B, then for $\lambda x.M$ only the type $\omega \to \tau$ can be derived. Therefore, in this system it is impossible to derive $\varphi_0 \to \varphi_1 \to \varphi_0$ for the lambda term $\lambda ab.a$. That type is derivable for that term in the strict system.

The restriction being made from the BCD-system to the strict system consists of eliminating the derivation rule ($\leq$). This rule plays an important part in [22], where the derivation rule ($\cap$E) is left out because it is in fact derivable from ($\leq$). In that system the statement $\lambda x.x{:}\varphi_0 \cap \varphi_1 \to \varphi_0$ can be derived by applying the ($\leq$)-rule to the derivation for

$$\vdash \lambda x.x{:}\varphi_0 \to \varphi_0$$

(allowed since $\varphi_0 \to \varphi_0 \leq \varphi_0 \cap \varphi_1 \to \varphi_0$). Similarly, the statement $\lambda ab.a{:}\varphi_0 \to \varphi_1 \to \varphi_0$ can be derived by applying the ($\leq$)-rule to the derivation for

$$\vdash \lambda ab.a{:}\varphi_0 \to \omega \to \varphi_0$$

(since $\varphi_0 \to \omega \to \varphi_0 \leq \varphi_0 \to \varphi_1 \to \varphi_0$).

Although the ($\leq$)-rule is not allowed in the strict system, it could of course be that the operation of rise, which works on pairs and produces pairs, is sound for strict pairs. This, however, is not true. Take for example the rise that maps the pair

$$\langle \{x{:}\varphi_0 \to \omega \to \varphi_0\},\, \varphi_0 \to \omega \to \varphi_0 \rangle$$

onto the pair

$$\langle \{x{:}\varphi_0 \to \omega \to \varphi_0\},\, \varphi_0 \to \varphi_1 \to \varphi_0 \rangle.$$

It is not possible to derive $x{:}\varphi_0 \to \varphi_1 \to \varphi_0$ from the basis $\{x{:}\varphi_0 \to \omega \to \varphi_0\}$ in the strict system.

So the results of [8] and [22] do not provide principal types for the strict type assignment system.

In proving that the strict type assignment system has the principal pair property, we could have used the same technique as [18]. In that paper principal typings for the type assignment system as presented in [17] are studied. That system is a combination of the BCD-system and the polymorphic type discipline as presented in [14]. This combination can be seen as an extension of the BCD-system by adding the type constructor '$\forall$': if φ is a type-variable and σ is a type, then $\forall\varphi.\sigma$ is a type. Also the type inclusion relation $\leq$ is extended in a natural way. The type assignment rules consist of ($\to$I), ($\to$E), ($\forall$I), ($\cap$I), ($\leq$) and (ω) – the rules ($\cap$E) and ($\forall$E) are omitted since they can be derived from ($\leq$).

The technique used in [18] is the following: For every A in $\lambda\perp$-normal form (see Definition 3.1) a relation

$\subseteq_A$ is defined on the inductively defined set of pairs $\langle B, \sigma\rangle$ admissible for A (i.e. such that $B \vdash A{:}\sigma$). This relation satisfies:

If $\langle B_1, \sigma_1\rangle \subseteq_A \langle B_2, \sigma_2\rangle$, then both pairs are admissible for A.

The principal pairs of terms in $\lambda\bot$-normal form are defined by induction on the structure of such terms (similar to Definition 3.2 of this paper). The proof is completed by showing that if the pair $\langle B, \sigma\rangle$ is admissible for A, and $\langle P, \pi\rangle$ is the principal pair of A, then $\langle P, \pi\rangle \subseteq_A \langle B, \sigma\rangle$.

A major difference between that technique and the one used in this paper (which is similar to the one used in [15], [8], and [22]), is that the latter provides, given a pair $\langle B, \sigma\rangle$ suitable for A, a sequence of operations that will transform the principal pair of A into $\langle B, \sigma\rangle$. The technique of [18] is sufficient to show that for every term in $\lambda\bot$-normal form there exists a principal type, but not to provide the way of creating a suitable pair from the principal one, as is done in this paper.

As in [8, 22], in this section we define, for each term in $\lambda\bot$-normal form, a principal pair (basis and type) in the strict type assignment system. (We will generalize these to arbitrary lambda terms in Subsection 5.2.) As in [5], we use here the symbol $\bot$ instead of Ω.

DEFINITION 3.1 ([5])

1. The set of $\Lambda\bot$-*terms* is defined as the set of lambda terms, extended by $\bot \in \Lambda\bot$.
2. The notion of reduction $\rightarrow_{\beta\bot}$ is defined as $\rightarrow_\beta$, extended by:
   (a) $\lambda x.\bot \rightarrow_{\beta\bot} \bot$.
   (b) $\bot M \rightarrow_{\beta\bot} \bot$.
3. The set of normal forms for elements of $\Lambda\bot$ with respect to $\rightarrow_{\beta\bot}$ is the set $\mathcal{N}$ of $\lambda\bot$-normal forms or *approximate normal forms*, and is inductively defined by:
   (a) All term-variables are elements of $\mathcal{N}$; $\bot \in \mathcal{N}$.
   (b) If $A \in \mathcal{N}$, $A \neq \bot$, then $\lambda x.A \in \mathcal{N}$.
   (c) If $A_1, \ldots, A_n \in \mathcal{N}$, then $xA_1 \ldots A_n \in \mathcal{N}$.
4. $A \in \mathcal{N}$ is a *direct approximant* of $M \in \Lambda$ if A matches M except for occurrences of $\bot$.
5. $A \in \mathcal{N}$ is an *approximant* of $M \in \Lambda$ (written: $A \sqsubseteq M$) if there is an $M' =_\beta M$ such that A is a direct approximant of M'.
6. $\mathcal{A}(M) = \{A \in \mathcal{N} \mid A \sqsubseteq M\}$.
7. The type assignment rules of Definition 2.2 (1) are generalized to elements of $\mathcal{N}$ by allowing the terms to be elements of $\Lambda\bot$. Also the notion of used basis as in Definition 2.7 is generalized to elements of $\mathcal{N}$.

Notice that, because the notion of strict type assignment as defined in Definition 2.2 is syntax directed, if $\bot$ occurs in a term M and $B \vdash_{\mathrm{S}} M{:}\sigma$, then by Definition 3.1 (7) either $\sigma = \omega$, or in the derivation for $M{:}\sigma$ the term $\bot$ appears in the right-hand subterm of an application on which the derivation rule ($\rightarrow$E) is used with $n = 0$.

In the construction of principal pairs for lambda terms, we will first choose, for any $A \in \mathcal{N}$, a particular pair $\langle P, \pi\rangle$ of basis P and type π, which we call respectively the *principal basis scheme* and *principal type scheme* of A. This pair will be called the *principal pair* of A.

DEFINITION 3.2 (cf. [8, 22])

1. Let $A \in \mathcal{N}$. $pp(A)$, the *principal pair* of A, is defined by:
   (a) $pp(\bot) = \langle \emptyset, \omega\rangle$.
   (b) $pp(x) = \langle \{x{:}\varphi\}, \varphi\rangle$.
   (c) If $A' \neq \bot$, and $pp(A') = \langle P', \pi'\rangle$, then:

i. If x occurs free in A', and $x{:}\sigma \in P'$, then $pp(\lambda x.A') = \langle P' \backslash x,\ \sigma \to \pi' \rangle$.
ii. otherwise $pp(\lambda x.A') = \langle P',\ \omega \to \pi' \rangle$.

(d) If $pp(A_i) = \langle P_i, \pi_i \rangle$ for $1 \leq i \leq n$ (we choose trivial variants that are disjoint in pairs), then $pp(xA_1 \ldots A_n) = \langle \Pi\{P_1, \ldots, P_n, \{x{:}\pi_1 \to \cdots \to \pi_n \to \varphi\}\},\ \varphi \rangle$, where φ is a type-variable that does not occur in $pp(A_i)$ for $1 \leq i \leq n$.

2. $\mathcal{P} = \{\langle P, \pi \rangle \mid \exists\, A \in \mathcal{N}\ [\ pp(A) = \langle P, \pi \rangle\ ]\}$.

Principal pairs are not completely well defined, since the type-variables mentioned are not unique. However types that only differ in the names of type-variables can be considered to be the same. Notice that if $\langle P, \pi \rangle \in \mathcal{P}$, then $\pi \in \mathcal{T}_s$, and that if $pp(A) = \langle P, \pi \rangle$, then P is used for $A{:}\pi$.

The principal pairs for lambda terms in the systems as presented in [8] and [22] are exactly the same. Since the strict type assignment system is a sub-system of the BCD-system (in the sense that if $B \vdash M{:}\sigma$ in the strict system, then also $B \vdash M{:}\sigma$ in the BCD-system, but not vice-versa), and it is a super-system for the CDV-system, it not surprising that the principal pair for a lambda term M in the strict system turns out to be exactly the same as the principal pair for that term in both the other systems.

4 Operations on pairs

In this section we present three different operations on pairs of $\langle$*basis*, *type*$\rangle$, namely substitution, expansion, and lifting. The operation of substitution – defined in Subsection 4.1 – is a slight modification of the one normally used; this modification is needed to make sure that substitution is closed on strict types. The operation of expansion – defined in Subsection 4.2 – coincides with the one given in [8, 22]. The operations of lifting – defined in Subsection 4.3 – is new, and forms the strict counterpart of the operation of rise. It deals with the introduction of extra (types to) statements in the basis of a derivation and introduces extra types for term-variables that are bound.

For the operations of substitution and expansion we prove soundness: for every $A \in \mathcal{N}$, basis B and type σ: if $B \vdash_{\mathrm{S}} A{:}\sigma$, then for every B', σ' such that $\langle B', \sigma' \rangle$ is a pair that can be obtained from $\langle B, \sigma \rangle$ through an operation of substitution or expansion, then $B' \vdash_{\mathrm{S}} A{:}\sigma'$. For the operation of lifting we prove a more restricted result: for every $A \in \mathcal{N}$, basis B and type σ: if $\langle B, \sigma \rangle$ is a primitive pair – defined in Subsection 4.3 – for A (then also $B \vdash_{\mathrm{S}} A{:}\sigma$), then every pair $\langle B', \sigma' \rangle$ that can be obtained from $\langle B, \sigma \rangle$ by lifting is also a primitive pair for A. We also show that there is no operation of lifting that is sound on all pairs.

We define a hierarchy of pairs, consisting of (in order): principal pairs, ground pairs and primitive pairs. We show that the set of ground pairs is closed under the operation of expansion, and that the set of primitive pairs is closed under the operation of lifting. These results are needed in the completeness proofs of Section 5.

4.1 Substitution

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 $\sigma \to \varphi$ (or $\sigma \cap \varphi$) into $\sigma \to \omega$ ($\sigma \cap \omega$), 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 4.1

1. The *substitution* $(\varphi := \alpha) : \mathcal{T}_{\mathrm{S}} \to \mathcal{T}_{\mathrm{S}}$, where φ is a type-variable and $\alpha \in \mathcal{T}_s \cup \{\omega\}$, is defined by:
   (a) $(\varphi := \alpha)(\varphi) = \alpha$.
   (b) $(\varphi := \alpha)(\varphi_0) = \varphi_0$, if $\varphi \neq \varphi_0$.
   (c) $(\varphi := \alpha)(\sigma \to \tau) = \omega$, if $(\varphi := \alpha)(\tau) = \omega$.

(d) $(\varphi := \alpha)(\sigma{\to}\tau) = (\varphi := \alpha)(\sigma) \to (\varphi := \alpha)(\tau)$, if $(\varphi := \alpha)(\tau) \neq \omega$.
(e) $(\varphi := \alpha)(\sigma_1 \cap \cdots \cap \sigma_n) = (\varphi := \alpha)(\sigma_1') \cap \cdots \cap (\varphi := \alpha)(\sigma_m')$,
where $\{\sigma_1', \ldots, \sigma_m'\} = \{\sigma_i \in \{\sigma_1, \ldots, \sigma_n\} \mid (\varphi := \alpha)(\sigma_i) \neq \omega\}$.

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. $S(B) = \{x{:}S(\alpha) \mid x{:}\alpha \in B \ \&\ S(\alpha) \neq \omega\}$.
4. $S(\langle B, \sigma\rangle) = \langle S(B), S(\sigma)\rangle$.

Notice that in (i.e), if $n = 0$, or for $1 \leq i \leq n$, $(\varphi := \alpha)(\sigma_i) = \omega$, then $(\varphi := \alpha)(\sigma_1 \cap \cdots \cap \sigma_n) = \omega$.

For substitutions, the following properties hold:

LEMMA 4.2
1. If $\sigma \leq_{\mathrm{S}} \tau$, then $S(\sigma) \leq_{\mathrm{S}} S(\tau)$.
2. If $\sigma \in \mathcal{T}_{\mathrm{s}}$, and $S(\sigma) \neq \omega$, then $S(\sigma) \in \mathcal{T}_{\mathrm{s}}$.

PROOF. Easy. ■

LEMMA 4.3
Let $\tau \in \mathcal{T}_{\mathrm{s}}$ and S be a substitution such that $S(\tau) = \tau'$, where $\tau' \neq \omega$. Then:

1. If $S(B \cup \{x{:}\sigma\}) = B' \cup \{x{:}\sigma'\}$, then $S(\langle B, \sigma{\to}\tau\rangle) = \langle B', \sigma'{\to}\tau'\rangle$.
2. If for every $1 \leq i \leq n$, $S(\langle B_i, \sigma_i\rangle) = \langle B_i', \sigma_i'\rangle$, then

$$S(\langle \Pi\{B_1, \ldots, B_n, \{x{:}\sigma_1 {\to} \cdots {\to} \sigma_n {\to} \tau\}\}, \tau\rangle) = \langle \Pi\{B_1', \ldots, B_n', \{x{:}\sigma_1' {\to} \cdots {\to} \sigma_n' {\to} \tau'\}\}, \tau'\rangle.$$

PROOF. Immediately by Definition 4.1. ■

We now prove that the operation of substitution is sound on all pairs of basis and type.

THEOREM 4.4
If $B \vdash_{\mathrm{S}} A{:}\sigma$, then for every substitution S: $S(B) \vdash_{\mathrm{S}} A{:}S(\sigma')$.

PROOF. By induction on the definition of $\vdash_{\mathrm{S}}$, using Definition 4.1, and Lemmas 4.2 and 4.3. ■

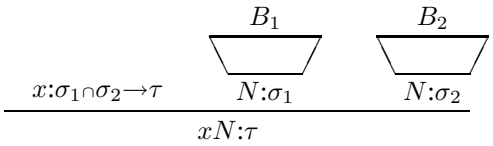
4.2 Expansion

The operation of expansion, as defined in this subsection, corresponds to the one given in [8] and is a simplified version of the one defined in [22]. 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. Expansion on types corresponds to the duplication of (sub)derivations: a subderivation in either the right-hand side of an ($\to$E)-step or the final step in a derivation in $\vdash_{\mathrm{S}}$ is expanded by copying. In this process the types that occur in the subderivation are also copied: the types in the conclusion and in the basis of the subderivation will be instantiated into a number of copies.

Suppose the following is a correct derivation:

$$\dfrac{\begin{array}{cc} & B \\ x{:}\sigma{\to}\tau & \qquad N{:}\sigma \end{array}}{xN{:}\tau}$$

then in general the expansion that replaces σ by $\sigma_1 \cap \sigma_2$ creates the following derivation:



(see also Example 4.15).

An expansion indicates not only the type to be expanded, but also the number of copies that has to be generated. It is a complex operation, possibly affecting more types than just those in which the one to be expanded occurs. To clarify this, consider the following: suppose that μ is a subtype of σ that is to be expanded into n copies. If $\tau \to \mu$ is also a subtype of σ, then just replacing μ by an intersection of copies of μ, would generate $\tau \to (\mu_1 \cap \cdots \cap \mu_n)$. This is a not a legal type. Defining an operation of expansion by saying that it should expand the subtype $\tau \to \mu$ into the type $(\tau \to \mu_1) \cap \cdots \cap (\tau \to \mu_n)$ – which is by definition of the relation $\leq$ a type equivalent to $\tau \to (\mu_1 \cap \cdots \cap \mu_n)$ – would give an expansion that is sound, but not sufficient: it would not be closed for ground pairs, a property we need in the proof of Theorem 5.2.

The subtype $\tau \to \mu$ will, therefore, be expanded into $(\tau_1 \to \mu_1) \cap \cdots \cap (\tau_n \to \mu_n)$, where the $\tau_1, \ldots, \tau_n$ are copies of τ. This means that also all other occurrences of τ should be expanded into $\tau_1 \cap \cdots \cap \tau_n$, with possibly the same effect on other types. Moreover, if φ is a type-variable that occurs in μ (or τ), then all occurrences of φ outside μ (τ) will be expanded into $\varphi_1 \cap \cdots \cap \varphi_n$, and all types of the shape $\rho \to \varphi$ will be expanded into $(\rho_1 \to \varphi_1) \cap \cdots \cap (\rho_n \to \varphi_n)$, etc.

Apparently, the expansion of μ can have a more than local effect on σ. Therefore, the expansion of a type is defined in a such a way that, before the replacement of types by intersections, all type-variables are collected that are affected by the expansion of μ. Then types are traversed top down, and subtypes are replaced if they end with one of the type-variables found. Types that are generated in this way need not be traversed, because all type-variables that are affected by the expansion are replaced, so no subtype can after expansion end with one of the type-variables found.

The definition of the operation of expansion can be found in Definition 4.10; it is based on the definition of a type-expansion, as given in Definition 4.6. The definition of expansion as presented here is different from the ones given in [8] and [22]. In those definitions subtypes are collected, whereas this definition collects type-variables.

DEFINITION 4.5
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 $\sigma_1 \cap \cdots \cap \sigma_n \to \tau$ ($n \geq 0$) is the last type-variable of τ.

DEFINITION 4.6 (cf. [22])
For every $\mu \in \mathcal{T}_S$, $n \geq 2$, basis B and $\sigma \in \mathcal{T}_S$, the quadruple $\langle \mu, n, B, \sigma \rangle$ determines a *type-expansion* $T_{\langle \mu,n,B,\sigma \rangle} : \mathcal{T}_S \to \mathcal{T}_S$, that is constructed as follows.

1. The set of type-variables $\mathcal{V}_\mu(\langle B, \sigma \rangle)$ is constructed by:
   (a) If φ occurs in μ, then $\varphi \in \mathcal{V}_\mu(\langle B, \sigma \rangle)$.
   (b) Let $\tau \in \mathcal{T}_{\langle B,\sigma \rangle}$ with last type-variable φ'. If $\varphi' \in \mathcal{V}_\mu(\langle B, \sigma \rangle)$, then for all type-variables φ that occur in τ: $\varphi \in \mathcal{V}_\mu(\langle B, \sigma \rangle)$.
2. Suppose $\mathcal{V}_\mu(\langle B, \sigma \rangle) = \{\varphi_1, \ldots, \varphi_m\}$. 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 $\mathcal{T}_{\langle B,\sigma \rangle}$, for $1 \leq i \leq n$ and $1 \leq j \leq m$. Let S_i be the substitution that replaces every φ_j by φ_j^i.
3. (a) $T_{\langle \mu,n,B,\sigma \rangle}(\alpha) = S_1(\alpha) \cap \cdots \cap S_n(\alpha)$, if the last type-variable of α is in $\mathcal{V}_\mu(\langle B, \sigma \rangle)$,

(b) $T_{\langle\mu,n,B,\sigma\rangle}(\alpha{\to}\beta) = T_{\langle\mu,n,B,\sigma\rangle}(\alpha) \to T_{\langle\mu,n,B,\sigma\rangle}(\beta)$, if the last type-variable of β is not in $\mathcal{V}_\mu(\langle B,\sigma\rangle)$.
(c) $T_{\langle\mu,n,B,\sigma\rangle}(\varphi) = \varphi$, if $\varphi \notin \mathcal{V}_\mu(\langle B,\sigma\rangle)$.
(d) $T_{\langle\mu,n,B,\sigma\rangle}(\alpha_1{\cap}\cdots{\cap}\alpha_n) = T_{\langle\mu,n,B,\sigma\rangle}(\alpha_1) \cap \cdots \cap T_{\langle\mu,n,B,\sigma\rangle}(\alpha_n)$.

4. $T_{\langle\mu,n,B,\sigma\rangle}(B') = \{x{:}T_{\langle\mu,n,B,\sigma\rangle}(\rho) \mid x{:}\rho \in B'\}$.
5. $T_{\langle\mu,n,B,\sigma\rangle}(\langle B',\sigma'\rangle) = \langle T_{\langle\mu,n,B,\sigma\rangle}(B'), T_{\langle\mu,n,B,\sigma\rangle}(\sigma')\rangle$.

Instead of $T_{\langle\mu,n,B,\sigma\rangle}$ we will simply write $\langle\mu,n,B,\sigma\rangle$.

For an operation of type-expansion the following properties hold:

LEMMA 4.7
Let $T = \langle\mu,n,B,\sigma\rangle$ be a type-expansion.

1. If $\tau \in \mathcal{T}_{\langle B,\sigma\rangle}$, then either
   (a) $T(\tau) = \tau_1{\cap}\cdots{\cap}\tau_n$ with for every $1 \le i \le n$, τ_i is a trivial variant of τ,
   or
   (b) $T(\tau) \in \mathcal{T}_s$.
2. $T(\Pi\{B_1,\ldots,B_n\}) = \Pi\{T(B_1),\ldots,T(B_n)\}$.
3. If $\tau \le_S \rho$, then $T(\tau) \le_S T(\rho)$.
4. If $B' \le_S B''$, then $T(B') \le_S T(B'')$.

PROOF. Immediately by Definition 4.6. ■

The following Lemmas are needed in the proofs of the following theorems. The first states that if the last type-variable of the type in a pair (of used basis and type) is affected by an expansion, then all type-variables in that pair are affected.

LEMMA 4.8
Let B' be used for $A{:}\tau$, where $\tau \in \mathcal{T}_s$ with last type-variable φ, and $\langle\mu,n,B,\sigma\rangle$ be a type-expansion such that $\mathcal{T}_{\langle B',\tau\rangle} \subseteq \mathcal{T}_{\langle B,\sigma\rangle}$. If $\varphi \in \mathcal{V}_\mu(\langle B,\sigma\rangle)$, then for every type-variable φ' that occurs in $\langle B',\tau\rangle$: $\varphi' \in \mathcal{V}_\mu(\langle B,\sigma\rangle)$.

PROOF. By induction on elements of $\mathcal{N}$.

1. $A \equiv x$, then $B' = \{x{:}\tau\}$. Since the last type-variable of τ is in $\mathcal{V}_\mu(\langle B,\sigma\rangle)$, and
$$\tau \in \mathcal{T}_{\langle B',\tau\rangle} \subseteq \mathcal{T}_{\langle B,\sigma\rangle},$$
all type-variables that occur in τ are in $\mathcal{V}_\mu(\langle B,\sigma\rangle)$.
2. $A \equiv \lambda x.A'$, then $\tau = \alpha{\to}\beta$. Since B' is used for $\lambda x.A'{:}\alpha{\to}\beta$, there is an α' such that $\alpha \le_S \alpha'$, and $B' \cup \{x{:}\alpha'\}$ is used for $A'{:}\beta$. Because the last type-variable of $\alpha{\to}\beta$ is in $\mathcal{V}_\mu(\langle B,\sigma\rangle)$, and
$$\alpha{\to}\beta \in \mathcal{T}_{\langle B',\alpha\to\beta\rangle} \subseteq \mathcal{T}_{\langle B,\sigma\rangle},$$
all type-variables of $\alpha{\to}\beta$ are in $\mathcal{V}_\mu(\langle B,\sigma\rangle)$. Since the last type-variable of $\alpha{\to}\beta$ is the last type-variable of β, and
$$\mathcal{T}_{\langle B'\cup\{x:\alpha'\},\beta\rangle} \subseteq \mathcal{T}_{\langle B'\cup\{x:\alpha\},\beta\rangle} \subseteq \mathcal{T}_{\langle B',\alpha\to\beta\rangle} \subseteq \mathcal{T}_{\langle B,\sigma\rangle},$$
by induction all type-variables in $\langle B'\cup\{x{:}\alpha'\},\beta\rangle$ are in $\mathcal{V}_\mu(\langle B,\sigma\rangle)$. So all type-variables in $\langle B',\alpha{\to}\beta\rangle$ are in $\mathcal{V}_\mu(\langle B,\sigma\rangle)$.

3. $A \equiv xA_1 \ldots A_m$. Then there are $\tau_1, \ldots, \tau_m$, and $B_1, \ldots, B_m$, such that for every $1 \leq j \leq m$, B_j is used for $A_j{:}\tau_j$ and $B' = \Pi\{B_1, \ldots, B_m, \{x{:}\tau_1 \rightarrow \cdots \rightarrow \tau_m \rightarrow \tau\}\}$. Since the last type-variable of τ is in $\mathcal{V}_\mu(\langle B, \sigma\rangle)$, and

$$\tau_1 \rightarrow \cdots \rightarrow \tau_m \rightarrow \tau \in \mathcal{T}_{\langle B',\ \tau\rangle} \subseteq \mathcal{T}_{\langle B,\sigma\rangle},$$

every type-variable in $\tau_1 \rightarrow \cdots \rightarrow \tau_m \rightarrow \tau$ is in $\mathcal{V}_\mu(\langle B, \sigma\rangle)$. Assume, without loss of generality, that each τ_j is strict. Then, for every $1 \leq j \leq m$, the last type-variable of τ_j is in $\mathcal{V}_\mu(\langle B, \sigma\rangle)$, and

$$\mathcal{T}_{\langle B_j,\ \tau_j\rangle} \subseteq \mathcal{T}_{\langle B',\ \tau\rangle} \subseteq \mathcal{T}_{\langle B,\sigma\rangle},$$

so by induction all type-variables in $\langle B_j, \tau_j\rangle$ are in $\mathcal{V}_\mu(\langle B, \sigma\rangle)$. So all type-variables in $\langle \Pi\{B_1, \ldots, B_m, \{x{:}\tau_1 \rightarrow \cdots \rightarrow \tau_m \rightarrow \tau\}\}, \tau\rangle$ are in $\mathcal{V}_\mu(\langle B, \sigma\rangle)$. ■

LEMMA 4.9
Let B' be used for $A{:}\tau$, where $\tau \in \mathcal{T}_s$, and $\langle \mu, n, B, \sigma\rangle$ be a type-expansion such that $\mathcal{T}_{\langle B',\ \tau\rangle} \subseteq \mathcal{T}_{\langle B,\sigma\rangle}$. Then either

1. There are $B_1, \ldots, B_n, \tau_1, \ldots, \tau_n$, such that

$$\langle \mu, n, B, \sigma\rangle\,(\langle B', \tau\rangle) = \langle \Pi\{B_1, \ldots, B_n\}, \tau_1 \cap \cdots \cap \tau_n\rangle$$

and for every $1 \leq i \leq n$, $\langle B_i, \tau_i\rangle$ is a trivial variant of $\langle B', \tau\rangle$,

or

2. $\langle \mu, n, B, \sigma\rangle\,(\langle B', \tau\rangle) = \langle B'', \tau'\rangle$, with $\tau' \in \mathcal{T}_s$.

PROOF. By Lemmas 4.7, and 4.8. ■
Notice that in particular this Lemma holds for the case that $\langle B', \tau\rangle = \langle B, \sigma\rangle$.

We now define the operation of expansion, that maps pairs to pairs.

DEFINITION 4.10
Let $\mathcal{P}airs$ be the set of all pairs of basis and type. For every $\mu \in \mathcal{T}_s$ and $n \geq 2$, the pair $\langle \mu, n\rangle$ determines an *expansion* $E_{\langle \mu,n\rangle} : \mathcal{P}airs \rightarrow \mathcal{P}airs$, defined by:

$$E_{\langle \mu,n\rangle}\,(\langle B, \sigma\rangle) = \langle \mu, n, B, \sigma\rangle\,(\langle B, \sigma\rangle).$$

Instead of $E_{\langle \mu,n\rangle}$ we write $\langle \mu, n\rangle$.

The following Lemma is needed in the proofs for the fact that expansion is closed on ground pairs (Theorem 4.14) and for completeness of chains of operations (Theorem 5.2).

LEMMA 4.11
Let $E = \langle \mu, n\rangle$ be an expansion, and $\tau \in \mathcal{T}_s$.

1. $E\,(\langle B \cup \{x{:}\sigma\}, \tau\rangle) = \langle B' \cup \{x{:}\sigma'\}, \tau'\rangle$, where $\tau' \in \mathcal{T}_s$, if and only if

$$E\,(\langle B, \sigma \rightarrow \tau\rangle) = \langle B', \sigma' \rightarrow \tau'\rangle.$$

2. Let $E\,(\langle B_i, \sigma_i\rangle) = \langle B_i', \sigma_i'\rangle$, for every $1 \leq i \leq n$. If for $1 \leq i \leq n$, $\varphi \notin \mathcal{V}_\mu(\langle B_i, \sigma_i\rangle)$, then

$$E\,(\langle \Pi\{B_1, \ldots, B_n, \{x{:}\sigma_1 \rightarrow \cdots \rightarrow \sigma_n \rightarrow \varphi\}\}, \varphi\rangle) = \langle \Pi\{B_1', \ldots, B_n', \{x{:}\sigma_1' \rightarrow \cdots \rightarrow \sigma_n' \rightarrow \varphi\}\}, \varphi\rangle.$$

PROOF. Easy, using Definitions 4.10 and 4.6, and Lemma 4.7. ■

We now prove that the operation of expansion is closed on the set of ground pairs, as defined in [8]. This property is needed in the proof that the operations defined in this paper are complete (as given in Subsection 5). Ground pairs are those pairs that can be obtained from principal pairs (and ground pairs) by expansion.

DEFINITION 4.12 ([8])
The pair $\langle B, \sigma\rangle$ is a *ground pair* for $A \in \mathcal{N}$ if and only if:

1. If $\sigma \in \mathcal{T}_s$, then:
   (a) If $A \equiv x$, then $B = \{x{:}\varphi\}$, and $\sigma = \varphi$.
   (b) If $A \equiv \lambda x.A'$, then:
      i. If $x \in \mathrm{FV}(A')$, then $\sigma = \alpha{\to}\beta$, and $\langle B \cup \{x{:}\alpha\}, \beta\rangle$ is a ground pair for A'.
      ii. If $x \notin \mathrm{FV}(A')$, then $\sigma = \omega{\to}\beta$, and $\langle B, \beta\rangle$ is a ground pair for A'.
   (c) If $A \equiv xA_1 \ldots A_m$, then $\sigma = \varphi$, and there are $B_1, \ldots, B_m, \tau_1, \ldots, \tau_m$ such that $B = \Pi\{B_1, \ldots, B_m, \{x{:}\tau_1{\to}\cdots{\to}\tau_m{\to}\varphi\}\}$, the $\langle B_j, \tau_j\rangle$ are disjoint in pairs, and for every $1 \le j \le m$, φ does not occur in $\langle B_j, \tau_j\rangle$, and $\langle B_j, \tau_j\rangle$ is a ground pair for A_j.
2. If $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$, then there are $B_1, \ldots, B_n$ such that $B = \Pi\{B_1, \ldots, B_n\}$, the $\langle B_i, \sigma_i\rangle$ are disjoint in pairs and for $1 \le i \le n$, $\langle B_i, \sigma_i\rangle$ is a ground pair for A.
3. If $\sigma = \omega$, then $B = \emptyset$ and $A \equiv \bot$.

LEMMA 4.13
1. If $\langle B, \sigma\rangle$ is a ground pair for A, then B is used for $A{:}\sigma$.
2. For every A, $pp(A)$ is a ground pair for A.

PROOF. Easy. ■

The following theorem states that expansion is closed on the set of ground pairs.

THEOREM 4.14
If $\langle B, \sigma\rangle$ is a ground pair for A, and $E = \langle \mu, n\rangle$ is an expansion such that $E(\langle B, \sigma\rangle) = \langle B', \sigma'\rangle$, then $\langle B', \sigma'\rangle$ is a ground pair for A.

PROOF. By induction on the definition of ground pairs. Again we only show the part $\sigma \in \mathcal{T}_s$. Notice that because of Lemma 4.13 (1) we already know that B is used for $A{:}\sigma$, and, therefore, by Lemma 4.9 either

1. $\sigma' = \sigma_1 \cap \cdots \cap \sigma_n$, $B' = \Pi\{B_1, \ldots, B_n\}$ where each $\langle B_i, \sigma_i\rangle$ is a trivial variant of $\langle B, \sigma\rangle$ and, therefore, a ground pair for A. Because of Definition 4.10, the $\langle B_i, \sigma_i\rangle$ are disjoint in pairs. So $\langle B', \sigma'\rangle$ is a ground pair for A.

or

2. $\sigma' \in \mathcal{T}_s$. This part is proved by induction on elements of $\mathcal{N}$. Notice that we need not consider the case that $A \equiv \bot$.
   (a) $A \equiv x$, $B = \{x{:}\varphi\}$, and $\sigma = \varphi$. Since $\sigma' \in \mathcal{T}_s$, $\varphi \notin \mathcal{V}_\mu(\langle B, \sigma\rangle)$ and $\langle B', \sigma'\rangle = \langle\{x{:}\varphi\}, \varphi\rangle$.
   (b) $A \equiv \lambda x.A'$, $\sigma = \alpha{\to}\beta$, and $\langle B \cup \{x{:}\alpha\}, \beta\rangle$ is a ground pair for A'. Let $\sigma' = \alpha'{\to}\beta'$. (Notice that if $x \notin \mathrm{FV}(A')$, then $\alpha = \alpha' = \omega$). By Lemma 4.11 (1)
   $$E(\langle B \cup \{x{:}\alpha\}, \beta\rangle) = \langle B' \cup \{x{:}\alpha'\}, \beta'\rangle,$$
   which is by induction a ground pair for A'. Because $\beta' \in \mathcal{T}_s$, also $\langle B', \alpha'{\to}\beta'\rangle$ is a ground pair for $\lambda x.A'$.
   (c) $A \equiv xA_1 \ldots A_m$, $\sigma = \varphi$, and $B = \Pi\{B_1, \ldots, B_m, \{x{:}\tau_1{\to}\cdots{\to}\tau_m{\to}\varphi\}\}$, the (disjoint in pairs) $\langle B_j, \tau_j\rangle$ are ground pairs for A_j, in which φ does not occur. Since $\sigma' \in \mathcal{T}_s$, $\sigma' = \varphi$. Let

$$E(\langle B_j, \tau_j\rangle) = \langle B_j', \tau_j'\rangle,$$

which is by induction a ground pair for A_j, for every $1 \leq j \leq m$. Then by Lemma 4.11 (2)

$$E(\langle \Pi\{B_1, \ldots, B_m, \{x{:}\tau_1 \to \cdots \to \tau_m \to \varphi\}\}, \varphi\rangle) = \langle \Pi\{B_1', \ldots, B_m', \{x{:}\tau_1' \to \cdots \to \tau_m' \to \varphi\}\}, \varphi\rangle.$$

Since the $\langle B_j, \tau_j\rangle$ are disjoint in pairs, because of Definition 4.10 also the $\langle B_j', \tau_j'\rangle$ are disjoint in pairs, and φ does not occur in any of these. So

$$\langle \Pi\{B_1', \ldots, B_m', \{x{:}\tau_1' \to \cdots \to \tau_m' \to \varphi\}\}, \varphi\rangle$$

is a ground pair for $xA_1 \ldots A_m$. ■

EXAMPLE 4.15 (cf. [8])
Take the pair $\langle \emptyset, (\omega \to (\varphi_0 \to \varphi_0) \to \varphi_1) \to \varphi_1\rangle$, which is the principal pair for $\lambda x.x\bot(\lambda y.y)$.

$$\dfrac{\dfrac{\dfrac{[x{:}\omega \to (\varphi_0 \to \varphi_0) \to \varphi_1]}{x\bot{:}(\varphi_0 \to \varphi_0) \to \varphi_1} \qquad \dfrac{[y{:}\varphi_0]}{\lambda y.y{:}\varphi_0 \to \varphi_0}}{x\bot(\lambda y.y){:}\varphi_1}}{\lambda x.x\bot(\lambda y.y){:}(\omega \to (\varphi_0 \to \varphi_0) \to \varphi_1) \to \varphi_1}$$

Take the expansion $E = \langle \varphi_0 \to \varphi_0, 2\rangle$. Then

$$E(\langle \emptyset, (\omega \to (\varphi_0 \to \varphi_0) \to \varphi_1) \to \varphi_1\rangle) = \langle \emptyset, (\omega \to (\varphi_2 \to \varphi_2) \cap (\varphi_3 \to \varphi_3) \to \varphi_1) \to \varphi_1\rangle,$$

which is by the previous Lemma a ground pair for the term $\lambda x.x\bot(\lambda y.y)$.

$$\dfrac{\dfrac{\dfrac{[x{:}\omega \to (\varphi_2 \to \varphi_2) \cap (\varphi_3 \to \varphi_3) \to \varphi_1]}{x\bot{:}(\varphi_2 \to \varphi_2) \cap (\varphi_3 \to \varphi_3) \to \varphi_1} \qquad \dfrac{[y{:}\varphi_2]}{\lambda y.y{:}\varphi_2 \to \varphi_2} \qquad \dfrac{[y{:}\varphi_3]}{\lambda y.y{:}\varphi_3 \to \varphi_3}}{x\bot(\lambda y.y){:}\varphi_1}}{\lambda x.x\bot(\lambda y.y){:}(\omega \to (\varphi_2 \to \varphi_2) \cap (\varphi_3 \to \varphi_3) \to \varphi_1) \to \varphi_1}$$

We can also prove that the operation of expansion is sound on all pairs. For that we need to prove first that the operation of type-expansion is sound on the pairs $\langle B, \sigma\rangle$ for the terms $A \in \mathcal{N}$ such that B is used for $A{:}\sigma$. Notice that the possible type-expansions are limited to those that at least compute the effect of the expansion on all types in $\mathcal{T}_{\langle B,\sigma\rangle}$.

THEOREM 4.16
If B_1 is used for $A{:}\sigma_1$, $\langle \mu, n, B, \sigma\rangle$ is a type-expansion such that $\langle \mu, n, B, \sigma\rangle(\langle B_1, \sigma_1\rangle) = \langle B_2, \sigma_2\rangle$, and $\mathcal{T}_{\langle B_1, \sigma_1\rangle} \subseteq \mathcal{T}_{\langle B,\sigma\rangle}$, then $B_2 \vdash_{\mathrm{S}} A{:}\sigma_2$.

PROOF. Again we only show the part $\sigma_1 \in \mathcal{T}_{\mathrm{s}}$. Because of Lemma 4.9, we know that either

1. $\sigma_2 = \sigma_1' \cap \cdots \cap \sigma_n'$, $B_2 = \Pi\{B_1', \ldots, B_n'\}$, and for every $1 \leq i \leq n$, $\langle B_i', \sigma_i'\rangle$ is a trivial variant of $\langle B_1, \sigma_1\rangle$. So $B_2 \vdash_{\mathrm{S}} A{:}\sigma_2$.

or

2. $\sigma_2 \in \mathcal{T}_{\mathrm{s}}$. This part is proved by induction on elements of $\mathcal{N}$. Again we need not consider the case that $A \equiv \bot$.
   (a) $A \equiv x$. Then $B_1 = \{x{:}\sigma_1\}$, and $B_2 = \{x{:}\sigma_2\}$. Then $B_2 \vdash_{\mathrm{s}} x{:}\sigma_2$.
   (b) $A \equiv \lambda x.A'$. Then $\sigma_1 = \alpha \to \beta$, and $B_1 \cup \{x{:}\alpha\} \vdash_{\mathrm{s}} A'{:}\beta$. Since $\sigma_2 \in \mathcal{T}_{\mathrm{s}}$, $\sigma_2 = \alpha' \to \beta'$ and by Definition 4.6, $\alpha' = \langle \mu, n, B, \sigma\rangle(\alpha)$, and $\beta' = \langle \mu, n, B, \sigma\rangle(\beta)$, so

$$\langle \mu, n, B, \sigma \rangle \, (\langle B_1 \cup \{x{:}\alpha\}, \, \beta \rangle) = \langle B_2 \cup \{x{:}\alpha'\}, \, \beta' \rangle.$$

B_1 is used for $\lambda x.A'{:}\alpha{\to}\beta$, so by Lemma 2.8 (3) there is a ρ such that $\alpha \leq_{\mathrm{S}} \rho$, and $B_1 \cup \{x{:}\rho\}$ is used for $A'{:}\beta$. Let $\rho' = \langle \mu, n, B, \sigma \rangle \, (\rho)$, then

$$\langle \mu, n, B, \sigma \rangle \, (\langle B_1 \cup \{x{:}\rho\}, \, \beta \rangle) = \langle B_2 \cup \{x{:}\rho'\}, \, \beta' \rangle.$$

Since

$$\mathcal{T}_{\langle B_1 \cup \{x:\rho\}, \, \beta \rangle} \subseteq \mathcal{T}_{\langle B_1 \cup \{x:\alpha\}, \, \beta \rangle} \subseteq \mathcal{T}_{\langle B_1, \, \alpha \to \beta \rangle} \subseteq \mathcal{T}_{\langle B, \sigma \rangle},$$

and $\beta' \in \mathcal{T}_{\mathrm{s}}$, by induction $B_2 \cup \{x{:}\rho'\} \vdash_{\mathrm{s}} A'{:}\beta'$.
By Lemmas 4.7 (4) and 2.6 also $B_2 \cup \{x{:}\alpha'\} \vdash_{\mathrm{s}} A'{:}\beta'$. Then $B_2 \vdash_{\mathrm{s}} \lambda x.A'{:}\sigma_2$.

(c) $A \equiv xA_1 \ldots A_m$. Then $B_1 = \Pi\{B_1', \ldots, B_m', \{x{:}\tau_1{\to}\cdots{\to}\tau_m{\to}\sigma_1\}\}$, where B_j' is used for $A_j{:}\tau_j$ (notice that τ_j need not be strict). Take

$$\langle B_j'', \tau_j'' \rangle = \langle \mu, n, B, \sigma \rangle \, (\langle B_j', \tau_j \rangle).$$

Since

$$\mathcal{T}_{\langle B_j', \, \tau_j \rangle} \subseteq \mathcal{T}_{\langle B_1, \, \sigma_1 \rangle} \subseteq \mathcal{T}_{\langle B, \sigma \rangle},$$

by induction: $B_j'' \vdash_{\mathrm{S}} A_j{:}\tau_i''$, for every $1 \leq j \leq m$. By Definition 4.6 $\tau_1''{\to}\cdots{\to}\tau_m''{\to}\sigma_2 = \langle \mu, n, B, \sigma \rangle \, (\tau_1{\to}\cdots{\to}\tau_m{\to}\sigma_1)$ so

$$\langle B_2, \sigma_2 \rangle = \langle \Pi\{B_1'', \ldots, B_m'', \{x{:}\tau_1''{\to}\cdots{\to}\tau_m''{\to}\sigma_2\}\}, \sigma_2 \rangle.$$

So $B_2 \vdash_{\mathrm{S}} xA_1 \ldots A_m{:}\sigma_2$. ■

The next theorem is also proved in [22] for the BCD-system, and states that expansion is a sound operation on pairs.

THEOREM 4.17
If $B \vdash_{\mathrm{S}} A{:}\sigma$, and $\langle \mu, n \rangle \, (\langle B, \, \sigma \rangle) = \langle B', \, \sigma' \rangle$, then $B' \vdash_{\mathrm{S}} A{:}\sigma'$.

PROOF. By Definition 4.10 $\langle \mu, n \rangle \, (\langle B, \, \sigma \rangle) = \langle \mu, n, B, \sigma \rangle \, (\langle B, \, \sigma \rangle)$. If $B \vdash_{\mathrm{S}} A{:}\sigma$, then by Lemma 2.8 (2) there is a B_1 such that $B \leq_{\mathrm{S}} B_1$ (so in particular $\mathcal{T}_{\langle B_1, \, \sigma \rangle} \subseteq \mathcal{T}_{\langle B, \sigma \rangle}$), and B_1 is used for $A{:}\sigma$. By Theorem 4.16, if $\langle B_2, \, \sigma' \rangle = \langle \mu, n, B, \sigma \rangle \, (\langle B_1, \, \sigma \rangle)$, then $B_2 \vdash_{\mathrm{S}} A{:}\sigma'$. Since $B \leq_{\mathrm{S}} B_1$, by Lemma 4.7 (4) $B' \leq_{\mathrm{S}} B_2$, so by Lemma 2.6 $B' \vdash_{\mathrm{S}} A{:}\sigma'$. ■

4.3 Lifting

The operation on pairs defined in this subsection forms the strict counterpart of the operation of rise, that consists of applying an extra derivation rule ($\leq$) to a derivation. As mentioned in the beginning of this paper, in the BCD-system the derivation rule ($\leq$) can be applied anywhere in the derivation, and the derivation rule ($\cap$E) can be omitted, since it is derivable from the rule ($\leq$).

The main difference between the BCD-system and the strict one is that in the latter the derivation rule ($\leq$) is omitted. In the strict type assignment system the only part of the ($\leq$)-rule that is kept, is the derivation rule ($\cap$E). Moreover, this rule can only be applied to term-variables. Therefore, the strict counterpart of the operation of rise will in fact be introducing extra ($\cap$E)-rules (or extra types in the upper half of that rule) for premisses.

In general, the derivation

$$\dfrac{\begin{array}{c}[x{:}\sigma]\\ \vdots\\ M{:}\tau\end{array}}{\lambda x.M{:}\sigma{\to}\tau}\ (\to\text{I}) \qquad \text{will be transformed into:} \qquad \dfrac{\begin{array}{c}\dfrac{[x{:}\sigma\cap\rho]}{x{:}\sigma}\ (\cap\text{E})\\ \vdots\\ M{:}\tau\end{array}}{\lambda x.M{:}\sigma\cap\rho{\to}\tau}\ (\to\text{I})$$

The notion of lifting as defined in this section, is not sound on all pairs $\langle B, \sigma\rangle$. Take for example the pair $\langle\emptyset, (\alpha{\to}\beta){\to}\alpha{\to}\beta\rangle$. This is a pair for both the lambda terms $\lambda xy.xy$ and $\lambda z.z$.

$$\dfrac{\dfrac{\dfrac{[x{:}\alpha{\to}\beta]\qquad[y{:}\alpha]}{xy{:}\beta}}{\lambda y.xy{:}\alpha{\to}\beta}}{\lambda xy.xy{:}(\alpha{\to}\beta){\to}\alpha{\to}\beta} \qquad\qquad \dfrac{[z{:}\alpha{\to}\beta]}{\lambda z.z{:}(\alpha{\to}\beta){\to}\alpha{\to}\beta}$$

Suppose there is an operation O that is able to express that in the derivation for

$$\lambda xy.xy{:}(\alpha{\to}\beta){\to}\alpha{\to}\beta,$$

the predicate for the term-variable y is lifted to $\alpha\cap\gamma$, which corresponds to the following derivation:

$$\dfrac{\dfrac{\dfrac{[x{:}\alpha{\to}\beta]\qquad\dfrac{[y{:}\alpha\cap\gamma]}{y{:}\alpha}}{xy{:}\beta}}{\lambda y.xy{:}\alpha\cap\gamma{\to}\beta}}{\lambda xy.xy{:}(\alpha{\to}\beta){\to}\alpha\cap\gamma{\to}\beta}$$

Then O should be such that $O(\langle\emptyset, (\alpha{\to}\beta){\to}\alpha{\to}\beta\rangle) = \langle\emptyset, (\alpha{\to}\beta){\to}\alpha\cap\gamma{\to}\beta\rangle$. However, the pair $\langle\emptyset, (\alpha{\to}\beta){\to}\alpha\cap\gamma{\to}\beta\rangle$ is not a suitable pair for $\lambda z.z$: there exists no derivation in $\vdash_{\text{S}}$ for

$$\lambda z.z{:}(\alpha{\to}\beta){\to}\alpha\cap\gamma{\to}\beta.$$

So it is not true that for every M, B and σ:

$$\text{If } B \vdash_{\text{S}} M{:}\sigma, \text{ and } O(\langle B, \sigma\rangle) = \langle B', \sigma'\rangle, \text{ then also } B' \vdash_{\text{S}} M{:}\sigma'.$$

Therefore, we are not able to prove that this O produces correct results for all lambda terms. We will however show that the operation defined here is closed for a certain class of pairs, being the primitive pairs. The problem mentioned above is then solved by allowing liftings only on primitive pairs for terms: the pair $\langle\emptyset, (\alpha{\to}\beta){\to}\alpha{\to}\beta\rangle$ is a primitive pair for the term $\lambda xy.xy$, not for the term $\lambda z.z$.

Before we define the operation of lifting, we need to define a relation $\leq_{\text{L}}$ on types. It is a natural extension of the relation $\leq_{\text{S}}$, that was only defined for intersection types. Notice that, in the definition of $\leq_{\text{L}}$, the arrow type constructor is contravariant in its left-hand argument.

DEFINITION 4.18

1. We define the partial order relation $\leq_{\text{L}}$ on $\mathcal{T}_{\text{S}}$ like $\leq_{\text{S}}$, but by adding the last alternative.
   (a) $\sigma \leq_{\text{L}} \sigma$.
   (b) $\sigma \leq_{\text{L}} \omega$.
   (c) $\sigma\cap\tau \leq_{\text{L}} \sigma \ \&\ \sigma\cap\tau \leq_{\text{L}} \tau$.
   (d) $\sigma \leq_{\text{L}} \tau \leq_{\text{L}} \rho \Rightarrow \sigma \leq_{\text{L}} \rho$.
   (e) $\sigma \leq_{\text{L}} \rho \ \&\ \sigma \leq_{\text{L}} \tau \Rightarrow \sigma \leq_{\text{L}} \rho\cap\tau$.
   (f) $\rho \leq_{\text{L}} \sigma \ \&\ \tau \leq_{\text{L}} \mu \Rightarrow \sigma{\to}\tau \leq_{\text{L}} \rho{\to}\mu$.
2. On $\mathcal{T}_{\text{S}}$, the relation $\sim_{\text{L}}$ is defined by: $\sigma \sim_{\text{L}} \tau \Leftrightarrow \sigma \leq_{\text{L}} \tau \leq_{\text{L}} \sigma$.

The relations are indexed by 'L' to indicate that they are related to lifting.

Notice that $\sim_{\mathrm{S}}$ is a true subrelation of $\sim_{\mathrm{L}}$, since for example $(\sigma{\to}\tau){\cap}(\sigma{\cap}\rho{\to}\tau) \sim_{\mathrm{L}} \sigma{\to}\tau$, but this does not hold in $\sim_{\mathrm{S}}$.

We can observe that strict type assignment is not closed for this relation $\leq_{\mathrm{L}}$, so the following does not hold:

$$\text{If } B \vdash_{\mathrm{S}} M{:}\sigma, \text{ and } \sigma \leq_{\mathrm{L}} \tau, \text{ then } B \vdash_{\mathrm{S}} M{:}\sigma.$$

As a counter example, take $\{x{:}\sigma{\to}\sigma\} \vdash_{\mathrm{s}} x{:}\sigma{\to}\sigma$. Notice that $\sigma{\to}\sigma \leq_{\mathrm{L}} \sigma{\cap}\tau{\to}\sigma$, but we can not derive $\{x{:}\sigma{\to}\sigma\} \vdash_{\mathrm{S}} x{:}\sigma{\cap}\tau{\to}\sigma$.

(A notion of type assignment that is a minor variant of the strict system is presented in [3]. (A short version of this paper appeared as [4].) This variant consists of replacing the type assignment rule ($\cap$E) of the strict system by one for ($\leq_{\mathrm{L}}$), also allowed only for term-variables. In this way a notion of type assignment is obtained that is truly the essential intersection type assignment system. The main properties of the strict- and the BCD-system also hold for the essential one, but, moreover, type assignment in this system is also closed for η-reduction. Also this notion of type assignment has the principal type property; the needed operations are exactly the same as defined in this paper for the strict system, only for the proof of that property one could use the exact same technique as used in [22], which is different from the one used here.)

As in Definition 2.5 (1) with the relation $\leq_{\mathrm{S}}$ we extend the relation $\leq_{\mathrm{L}}$ to bases. For the relation $\leq_{\mathrm{L}}$ the following properties hold:

LEMMA 4.19
1. $\sigma \leq_{\mathrm{S}} \tau \Rightarrow \sigma \leq_{\mathrm{L}} \tau$.
2. $\varphi \leq_{\mathrm{L}} \sigma \Rightarrow \sigma = \varphi$. So $\{\sigma \mid \sigma \sim_{\mathrm{L}} \varphi\} = \{\varphi\}$.
3. $\omega \leq_{\mathrm{L}} \sigma \Rightarrow \sigma = \omega$. So $\{\sigma \mid \sigma \sim_{\mathrm{L}} \omega\} = \{\omega\}$.
4. $\sigma{\to}\tau \leq_{\mathrm{L}} \rho \in \mathcal{T}_{\mathrm{s}} \Leftrightarrow \exists\, \alpha \in \mathcal{T}_{\mathrm{S}}, \beta \in \mathcal{T}_{\mathrm{s}} [\, \rho = \alpha{\to}\beta \;\&\; \alpha \leq_{\mathrm{L}} \sigma \;\&\; \tau \leq_{\mathrm{L}} \beta \,]$.
5. $\sigma_1{\cap}\cdots{\cap}\sigma_n \leq_{\mathrm{L}} \tau \in \mathcal{T}_{\mathrm{s}} \Rightarrow \exists\, 1 \leq i \leq n [\, \sigma_i \leq_{\mathrm{L}} \tau \,]$.

PROOF. Easy. ■

Using this Lemma, we can prove the following:

LEMMA 4.20
1. $\sigma_1{\cap}\cdots{\cap}\sigma_n \leq_{\mathrm{L}} \tau \in \mathcal{T}_{\mathrm{S}} \Rightarrow$
   $\exists\, \tau_1, \ldots, \tau_m \in \mathcal{T}_{\mathrm{s}} [\, \tau = \tau_1{\cap}\cdots{\cap}\tau_m \;\&\; \forall\, 1 \leq j \leq m\; \exists\, 1 \leq i \leq n [\, \sigma_i \leq_{\mathrm{L}} \tau_j \,] \,]$.
2. $\sigma \leq_{\mathrm{L}} \tau \;\&\; \sigma \in \mathcal{T}_{\mathrm{s}} \Rightarrow$
   $\exists\, \tau_1, \ldots, \tau_m\; (m \geq 0) [\, \tau = \tau_1{\cap}\cdots{\cap}\tau_m \;\&\; \forall\, 1 \leq j \leq m [\, \sigma \leq_{\mathrm{L}} \tau_j \,] \,]$.
3. $B' \leq_{\mathrm{L}} B \leq_{\mathrm{S}} \{x{:}\sigma\} \;\&\; \sigma \in \mathcal{T}_{\mathrm{s}} \Rightarrow \exists\, \sigma' \in \mathcal{T}_{\mathrm{s}} [\, B' \leq_{\mathrm{S}} \{x{:}\sigma'\} \;\&\; \sigma' \leq_{\mathrm{L}} \sigma \,]$.

PROOF. Easy. ■

We now come to the definition of lifting. It is based on the relation $\leq_{\mathrm{L}}$, in the same way as the operation of rise is based on $\leq$.

DEFINITION 4.21
A *lifting L* is an operation denoted by a pair of pairs $\langle\langle B_0, \tau_0\rangle, \langle B_1, \tau_1\rangle\rangle$ such that $\tau_0 \leq_{\mathrm{L}} \tau_1$ and $B_1 \leq_{\mathrm{L}} B_0$, and is defined by:

1. (a) $L(\sigma) = \tau_1$, if $\sigma = \tau_0$.
   (b) $L(\sigma) = \sigma$, otherwise.
2. (a) $L(B) = B_1$, if $B = B_0$.
   (b) $L(B) = B$, otherwise.
3. $L(\langle B, \sigma\rangle) = \langle L(B), L(\sigma)\rangle$.

For operations of lifting, the following properties hold:

LEMMA 4.22

1. $\langle\langle B \cup \{x{:}\sigma\}, \tau\rangle, \langle B' \cup \{x{:}\sigma'\}, \tau'\rangle\rangle$ is a lifting (where $\tau, \tau' \in \mathcal{T}_s$), if and only if $\langle\langle B, \sigma{\to}\tau\rangle, \langle B', \sigma'{\to}\tau'\rangle\rangle$ is a lifting.
2. $\langle\langle B_i, \sigma_i\rangle, \langle B_i', \sigma_i'\rangle\rangle$ is a lifting for every $1 \leq i \leq n$, if and only if

$$\langle\langle \Pi\{B_1, \ldots, B_n, \{x{:}\sigma_1{\to}\cdots{\to}\sigma_n{\to}\varphi\}\}, \varphi\rangle, \langle \Pi\{B_1', \ldots, B_n', \{x{:}\sigma_1'{\to}\cdots{\to}\sigma_n'{\to}\varphi\}\}, \varphi\rangle\rangle$$

   is a lifting.

PROOF. By Definitions 4.18 and 4.23. ■

As remarked before, we are not able to prove that this operation of lifting is sound on all pairs for a fixed term A. We can, however, show that there is a set of primitive pairs, a subset of the set of pairs, for which the operation is sound. The definition for primitive pairs we give here is based on the definition of ground pairs as given in Definition 4.12. The main difference between ground pairs and primitive pairs is that in a primitive pair a predicate for a term-variable x (bound or free) is not the smallest type needed, but can contain some additional, irrelevant types. Central in the definition is that a pair is called a primitive pair for an application if the type in the pair is a type-variable, and that term-variables that are not head-variables of a term $xA_1 \ldots A_m$ are also typed with type-variables.

DEFINITION 4.23
The pair $\langle B, \sigma\rangle$ is a *primitive pair* for A if and only if:

1. If $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$ ($n \geq 0$), then for every $1 \leq i \leq n$, $\langle B, \sigma_i\rangle$ is a primitive pair for A.
2. If $\sigma \in \mathcal{T}_s$, then:
   (a) If $A \equiv x$, then $\sigma = \varphi$, and $B \leq_S \{x{:}\varphi\}$.
   (b) If $A \equiv \lambda x.A'$, then there are $\alpha \in \mathcal{T}_S$ and $\beta \in \mathcal{T}_s$, such that: $\sigma = \alpha{\to}\beta$, and $\langle B \cup \{x{:}\alpha\}, \beta\rangle$ is a primitive pair for A'.
   (c) If $A \equiv xA_1 \ldots A_m$, then there are $\tau_1, \ldots, \tau_m \in \mathcal{T}_S$, and a type-variable φ such that $\sigma = \varphi$, $B \leq_S \{x{:}\tau_1{\to}\cdots{\to}\tau_m{\to}\varphi\}$, and for every $1 \leq j \leq m$, $\langle B, \tau_j\rangle$ is a primitive pair for A_j.

Notice that in this definition the relation $\leq_S$ is used, not $\leq_L$.

For primitive pairs, the following properties hold.

LEMMA 4.24

1. If $\langle B, \sigma\rangle$ is a primitive pair for A, then $B \vdash_S A{:}\sigma$.
2. For every A, every ground pair for A is a primitive pair for A.

PROOF. By easy induction. ■

The next theorem states that a lifting performed on a primitive pair for A, produces a primitive pair for A.

THEOREM 4.25
For all $A \in \mathcal{N}$, liftings L: if $\langle B, \sigma\rangle$ is a primitive pair for A, and $L(\langle B, \sigma\rangle) = \langle B', \sigma'\rangle$, then $\langle B', \sigma'\rangle$ is a primitive pair for A.

PROOF. By induction on the definition of primitive pairs.

1. $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$ ($n \geq 0$), and for every $1 \leq i \leq n$, $\langle B, \sigma_i\rangle$ is a primitive pair for A.
   Since $\sigma_1 \cap \cdots \cap \sigma_n \leq_{\mathrm{L}} \sigma'$, by Lemma 4.20 (1) there are ${\sigma_1}', \ldots, {\sigma_m}'$ such that $\sigma' = {\sigma_1}' \cap \cdots \cap {\sigma_m}'$, and for every $1 \leq j \leq m$, there is an $1 \leq i_j \leq n$, such that $\sigma_{i_j} \leq_{\mathrm{L}} {\sigma_j}'$. Let for every $1 \leq j \leq m$,
   $$L_j = \langle\langle B, \sigma_{i_j}\rangle, \langle B', {\sigma_j}'\rangle\rangle,$$
   then by induction for every $1 \leq j \leq m$, $\langle B', {\sigma_j}'\rangle$ is a primitive pair for A.
   So $\langle B', {\sigma_1}' \cap \cdots \cap {\sigma_m}'\rangle$ is a primitive pair for A.
2. $\sigma \in \mathcal{T}_{\mathrm{s}}$. This part is proved by induction on elements of $\mathcal{N}$. Notice that we need not consider the case that $A \equiv \bot$.
   (a) $A \equiv x$, and $\sigma = \varphi$. By Lemmas 4.19 (2) and 4.20 (2) $\sigma' = \varphi$, so by Lemma 4.20 (3) $B' \leq_{\mathrm{S}} \{x{:}\varphi\}$. So $\langle B', \varphi\rangle$ is a primitive pair for x.
   (b) $A \equiv \lambda x.A'$. Then there are α and β, such that $\sigma = \alpha \to \beta$ and $\langle B \cup \{x{:}\alpha\}, \beta\rangle$ is a primitive pair for A'. Then by Lemma 4.19 (4) and 4.20 (2) there are $\rho_1, \ldots, \rho_n, \mu_1, \ldots, \mu_n$ such that $\sigma' = (\rho_1 \to \mu_1) \cap \cdots \cap (\rho_n \to \mu_n)$, and for $1 \leq i \leq n$, $\rho_i \leq_{\mathrm{L}} \alpha$ and $\beta \leq_{\mathrm{L}} \mu_i$. Take for every $1 \leq i \leq n$,
   $$L_i = \langle\langle B \cup \{x{:}\alpha\}, \beta\rangle, \langle B' \cup \{x{:}\rho_i\}, \mu_i\rangle\rangle,$$
   then by induction $\langle B' \cup \{x{:}\rho_i\}, \mu_i\rangle$ is a primitive pair for A' for every $1 \leq i \leq n$. So for every $1 \leq i \leq n$, $\langle B', \rho_i \to \mu_i\rangle$ is a primitive pair for A, so $\langle B', \sigma'\rangle$ is a primitive pair for A.
   (c) $A \equiv xA_1 \ldots A_m$, there are $\tau_1, \ldots, \tau_m$ and φ such that $\sigma = \varphi$, $B \leq_{\mathrm{S}} \{x{:}\tau_1 \to \cdots \to \tau_m \to \varphi\}$, and for every $1 \leq j \leq m$, $\langle B_j, \tau_j\rangle$ is a primitive pair for A_j. Then by Lemmas 4.19 (2) and 4.20 (2) $\sigma' = \varphi$, and by Lemmas 4.19 (4) and 4.20 (3) there are ${\tau_1}', \ldots, {\tau_m}' \in \mathcal{T}_{\mathrm{S}}$ such that
   $$B' \leq_{\mathrm{S}} \{x{:}{\tau_1}' \to \cdots \to {\tau_m}' \to \varphi\}, \text{ and for } 1 \leq j \leq m,\ \tau_j \leq_{\mathrm{L}} {\tau_j}'.$$
   Take for $1 \leq j \leq m$, $L_j = \langle\langle B, \tau_j\rangle, \langle B', {\tau_j}'\rangle\rangle$, then by induction $\langle B', {\tau_j}'\rangle$ is a primitive pair for A_j.
   So $\langle B', \varphi\rangle$ is a primitive pair for A. ■

EXAMPLE 4.26
1. Take the pair $\langle\{y{:}(\varphi_0 \to \varphi_0) \cap (\varphi_1 \to \varphi_1) \to \varphi_2\}, \varphi_2\rangle$, which is a primitive pair for the lambda term $y(\lambda x.x)$.
   $$\dfrac{y{:}(\varphi_0 \to \varphi_0) \cap (\varphi_1 \to \varphi_1) \to \varphi_2 \qquad \dfrac{[x{:}\varphi_0]}{\lambda x.x{:}\varphi_0 \to \varphi_0} \qquad \dfrac{[x{:}\varphi_1]}{\lambda x.x{:}\varphi_1 \to \varphi_1}}{y(\lambda x.x){:}\varphi_2}$$
   Since $(\varphi_0 \cap \varphi_3 \to \varphi_0) \cap (\varphi_1 \to \varphi_1) \to \varphi_2 \leq_{\mathrm{L}} (\varphi_0 \to \varphi_0) \cap (\varphi_1 \to \varphi_1) \to \varphi_2$, the pair
   $$\langle\{y{:}(\varphi_0 \cap \varphi_3 \to \varphi_0) \cap (\varphi_1 \to \varphi_1) \to \varphi_2\}, \varphi_2\rangle$$
   is a primitive pair for $y(\lambda x.x)$.
   $$\dfrac{y{:}(\varphi_0 \cap \varphi_3 \to \varphi_0) \cap (\varphi_1 \to \varphi_1) \to \varphi_2 \qquad \dfrac{\dfrac{[x{:}\varphi_0 \cap \varphi_3]}{x{:}\varphi_0}}{\lambda x.x{:}\varphi_0 \cap \varphi_3 \to \varphi_0} \qquad \dfrac{[x{:}\varphi_1]}{\lambda x.x{:}\varphi_1 \to \varphi_1}}{y(\lambda x.x){:}\varphi_2}$$
2. Take the pair $\langle\emptyset, ((\varphi_0 \to \omega \to \varphi_0) \to \varphi_1) \to \varphi_1\rangle$, which is a primitive pair for the lambda term $\lambda z.z(\lambda xy.x)$.

$$\dfrac{\dfrac{[z{:}(\varphi_0{\to}\omega{\to}\varphi_0){\to}\varphi_1]\quad \dfrac{\dfrac{[x{:}\varphi_0]}{\lambda y.x{:}\omega{\to}\varphi_0}}{\lambda xy.x{:}\varphi_0{\to}\omega{\to}\varphi_0}}{z(\lambda xy.x){:}\varphi_1}}{\lambda z.z(\lambda xy.x){:}((\varphi_0{\to}\omega{\to}\varphi_0){\to}\varphi_1){\to}\varphi_1}$$

Since $((\varphi_0{\to}\omega{\to}\varphi_0){\to}\varphi_1){\to}\varphi_1 \leq_{\mathrm{L}} ((\varphi_0{\to}\varphi_2{\to}\varphi_0){\to}\varphi_1){\to}\varphi_1$, the pair

$$\langle \emptyset, ((\varphi_0{\to}\varphi_2{\to}\varphi_0){\to}\varphi_1){\to}\varphi_1\rangle$$

is a primitive pair for $\lambda z.z(\lambda xy.x)$.

$$\dfrac{\dfrac{[z{:}(\varphi_0{\to}\varphi_2{\to}\varphi_0){\to}\varphi_1]\quad \dfrac{\dfrac{[x{:}\varphi_0]}{\lambda y.x{:}\varphi_2{\to}\varphi_0}}{\lambda xy.x{:}\varphi_0{\to}\varphi_2{\to}\varphi_0}}{z(\lambda xy.x){:}\varphi_1}}{\lambda z.z(\lambda xy.x){:}((\varphi_0{\to}\varphi_2{\to}\varphi_0){\to}\varphi_1){\to}\varphi_1}$$

5 Completeness for lambda terms

Although lifting is not sound on all pairs, we are able to prove that the three operations defined in the previous section are sufficient (complete): for every pair $\langle B, \sigma\rangle$ and $A \in \mathcal{N}$, if $B \vdash_{\mathrm{S}} A{:}\sigma$, then there exists a number of operations of expansion, one operation of lifting, and a number of operations of substitution, such that $\langle B, \sigma\rangle$ can be obtained from $PP_{\mathrm{S}}(A)$ by performing these operations in sequence. We will finish the proofs of this paper by generalizing this result to arbitrary lambda terms.

5.1 Soundness and completeness for terms in $\lambda\perp$-normal form

DEFINITION 5.1

1. A *chain* is an object $\langle O_1, \ldots, O_n\rangle$, where each O_i is an operation of expansion, lifting, or substitution, and
$$\langle O_1, \ldots, O_n\rangle(\langle B, \sigma\rangle) = O_n(\cdots(O_1(\langle B, \sigma\rangle))\cdots).$$
2. On chains we denote the operation of concatenation by $*$, and
$$\langle O_1, \ldots, O_i\rangle * \langle O_{i+1}, \ldots, O_n\rangle = \langle O_1, \ldots, O_n\rangle.$$
3. A *strict chain* is a chain consisting of a number of expansions, one lifting, and a number of substitutions, in that order. So a strict chain is a chain $\langle E_1, \ldots, E_m, L, S_1, \ldots, S_n\rangle$, where $m \geq 0$, and $n \geq 0$. We also write $\vec{E} * \langle L\rangle * \vec{S}$.

We will now prove that for every suitable pair for a term A, there exists a strict chain such that the result of the application of this chain on the principal pair of A produces the desired pair. Part (1) of the Lemmas 4.3, 4.11, and 4.22 are needed for the inductive step in case of an abstraction term, part (3.b) of the proof, part (2) of those Lemmas are needed for the inductive step in case of an application term, part (3.c). Notice that, by construction, all operations mentioned in that part satisfy the conditions required by these Lemmas.

THEOREM 5.2
If $B \vdash_{\mathrm{S}} A{:}\sigma$, and $PP_{\mathrm{S}}(A) = \langle P, \pi\rangle$, then there exists a strict chain C such that $C(\langle P, \pi\rangle) = \langle B, \sigma\rangle$.

PROOF. By induction on the definition of $\vdash_{\mathrm{S}}$.

1. $B \vdash_{\mathrm{S}} A{:}\omega$. Assume, without loss of generality, that P and B are disjoint. Take
$$L = \langle\langle P, \pi\rangle, \langle\Pi\{P, B\}, \pi\rangle\rangle,$$
and S the substitution that replaces all type-variables occurring in $\langle P, \pi\rangle$ by ω.
Take $C = \langle L, S\rangle$.
2. $B \vdash_{\mathrm{S}} A{:}\sigma_1 \cap \cdots \cap \sigma_n$. Then for $1 \leq i \leq n$, $B \vdash_{\mathrm{S}} A{:}\sigma_i$. For $1 \leq i \leq n$, take $\langle B_i, \tau_i\rangle$ (disjoint in pairs), trivial variants of $\langle B, \sigma_i\rangle$. Take S such that
$$S(\langle\Pi\{B_1, \ldots, B_n\}, \tau_1 \cap \cdots \cap \tau_n\rangle) = \langle B, \sigma_1 \cap \cdots \cap \sigma_n\rangle,$$
and let $E = \langle\pi, n\rangle$, then
$$E(\langle P, \pi\rangle) = \langle\Pi\{P_1, \ldots, P_n\}, \pi_1 \cap \cdots \cap \pi_n\rangle,$$
with $PP_{\mathrm{S}}(A) = \langle P_i, \pi_i\rangle$. By induction there exist strict chains $C_1, \ldots, C_n$ such that for $1 \leq i \leq n$, $C_i(\langle P_i, \pi_i\rangle) = \langle B_i, \tau_i\rangle$. Let for $1 \leq i \leq n$, $C_i = \vec{E}_i * \langle L_i\rangle * \vec{S_i}$. Let $\vec{E} = \vec{E}_1 * \cdots * \vec{E}_n$, and $\vec{S} = \vec{S_1} * \cdots * \vec{S_n}$. Let for $1 \leq i \leq n$, $\vec{E}_i(\langle P_i, \pi_i\rangle) = \langle B_i', \tau_i'\rangle$; by construction there are B_i'', τ_i'', such that $L_i = \langle\langle B_i', \tau_i'\rangle, \langle B_i'', \tau_i''\rangle\rangle$. Take
$$L = \langle\langle\Pi\{B_1', \ldots, B_n'\}, \tau_1' \cap \cdots \cap \tau_n'\rangle, \langle\Pi\{B_1'', \ldots, B_n''\}, \tau_1'' \cap \cdots \cap \tau_n''\rangle\rangle.$$
Take $C = \langle E\rangle * \vec{E} * \langle L\rangle * \vec{S} * \langle S\rangle$.
3. $B \vdash_{\mathrm{s}} A{:}\sigma$, so $\sigma \in \mathcal{T}_{\mathrm{s}}$. This part is proved by induction on the structure of elements of $\mathcal{N}$. Notice that we need not consider the case that $A \equiv \bot$.
   (a) $A \equiv x$. Then $B \leq_{\mathrm{S}} \{x{:}\sigma\}$, $P = \{x{:}\varphi\}$, and $\pi = \varphi$. Assume, without loss of generality, that P and B are disjoint. Take $L = \langle\langle P, \varphi\rangle, \langle\Pi\{P, B\}, \varphi\rangle\rangle$, and $S = (\varphi := \sigma)$. Take $C = \langle L, S\rangle$.
   (b) $A \equiv \lambda x.A'$. Then there are α, β such that $\sigma = \alpha{\to}\beta$, and $B \cup \{x{:}\alpha\} \vdash_{\mathrm{S}} A'{:}\beta$.
   We distinguish two cases:
      i. $x \in \mathrm{FV}(A')$. Then $PP_{\mathrm{S}}(\lambda x.A') = \langle P, \mu{\to}\pi\rangle$, where $PP_{\mathrm{S}}(A') = \langle P \cup \{x{:}\mu\}, \pi\rangle$. By induction there exists a strict chain $C' = \vec{E} * \langle L'\rangle * \vec{S}$ such that
$$C'(\langle P \cup \{x{:}\mu\}, \pi\rangle) = \langle B \cup \{x{:}\alpha\}, \beta\rangle.$$
      Let
$$\vec{E}(\langle P \cup \{x{:}\mu\}, \pi\rangle) = \langle B_1 \cup \{x{:}\alpha_1\}, \beta_1\rangle, \text{ and}$$
$$L' = \langle\langle B_1 \cup \{x{:}\alpha_1\}, \beta_1\rangle, \langle B_2 \cup \{x{:}\alpha_2\}, \beta_2\rangle\rangle.$$
      Since $\beta \in \mathcal{T}_{\mathrm{s}}$, by construction also $\beta_2 \in \mathcal{T}_{\mathrm{s}}$ and by Lemma 4.11 (1) $\vec{E}(\langle P, \mu{\to}\pi\rangle) = \langle B_1, \alpha_1{\to}\beta_1\rangle$.
      Take
$$L = \langle\langle B_1, \alpha_1{\to}\beta_1\rangle, \langle B_2, \alpha_2{\to}\beta_2\rangle\rangle,$$
      which, by Lemma 4.22 (1), is a lifting.
      ii. $x \notin \mathrm{FV}(A')$. Then $PP_{\mathrm{S}}(\lambda x.A') = \langle P, \omega{\to}\pi\rangle$, where $PP_{\mathrm{S}}(A') = \langle P, \pi\rangle$. By induction there exists a strict chain $C' = \vec{E} * \langle L'\rangle * \vec{S}$ such that
$$C'(\langle P, \pi\rangle) = \langle B \cup \{x{:}\alpha\}, \beta\rangle.$$
      Let $\vec{E}(\langle P, \pi\rangle) = \langle B_1, \beta_1\rangle$, and $L' = \langle\langle B_1, \beta_1\rangle, \langle B_2 \cup \{x{:}\alpha_2\}, \beta_2\rangle\rangle$.
      Since $\beta \in \mathcal{T}_{\mathrm{s}}$, by construction also $\beta_2 \in \mathcal{T}_{\mathrm{s}}$ and by Lemma 4.11 (1) $\vec{E}(\langle P, \omega{\to}\pi\rangle) = \langle B_1, \omega{\to}\beta_1\rangle$.
      Take
$$L = \langle\langle B_1, \omega{\to}\beta_1\rangle, \langle B_2, \alpha_2{\to}\beta_2\rangle\rangle,$$

which, by Lemma 4.22 (1), is a lifting.
Notice that in both cases by Theorems 4.14 and 4.25, $\langle B_2, \alpha_2{\to}\beta_2\rangle$ is a primitive pair for $\lambda x.A'$. Then by Lemma 4.3 (1) $\vec{S}(\langle B_2, \alpha_2{\to}\beta_2\rangle) = \langle B, \alpha{\to}\beta\rangle$. Take $C = \vec{E} * \langle L\rangle * \vec{S}$.

(c) $A \equiv xA_1 \ldots A_m$. Then there are $\sigma_1, \ldots, \sigma_m$ such that $B \leq_{\mathrm{S}} \{x{:}\sigma_1{\to}\cdots{\to}\sigma_m{\to}\sigma\}$, and for every $1 \leq j \leq m$, $B \vdash_{\mathrm{S}} A_j{:}\sigma_j$, and

$$P = \Pi\{P_1, \ldots, P_m, \{x{:}\pi_1{\to}\cdots{\to}\pi_m{\to}\varphi\}\},\ \pi = \varphi,$$

with for every $1 \leq j \leq m$, $PP_{\mathrm{S}}(A_j) = \langle P_j, \pi_j\rangle$, in which φ does not occur.
Take $\langle B_j, \tau_j\rangle$ (disjoint in pairs), trivial variants of $\langle B, \sigma_j\rangle$. Let S be such that

$$S(\langle\Pi\{B_1, \ldots, B_m\}, \tau_1{\cap}\cdots{\cap}\tau_m\rangle) = \langle B, \sigma_1{\cap}\cdots{\cap}\sigma_m\rangle.$$

By induction there are $C_1, \ldots, C_m$ such that for $1 \leq j \leq m$, $C_j(\langle P_j, \pi_j\rangle) = \langle B_j, \tau_j\rangle$. Notice that the strict chains $C_1, \ldots, C_m$ do not interfere, and that φ does not occur in any of the operations or pairs. Let for $1 \leq j \leq m$, $C_j = \vec{E}_j * \langle L_j\rangle * \vec{S}_j$, and $\vec{E} = \vec{E}_1 * \cdots * \vec{E}_m$, and $\vec{S} = \vec{S}_1 * \cdots * \vec{S}_m$.
Let for $1 \leq j \leq m$, $\vec{E}_i(\langle P_j, \pi_j\rangle) = \langle B_j{'}, \tau_j{'}\rangle$, and $L_j = \langle\langle B_j{'}, \tau_j{'}\rangle, \langle B_j{''}, \tau_j{''}\rangle\rangle$.
Then by Lemma 4.11 (2):

$$\vec{E}(\langle\Pi\{P_1, \ldots, P_m, \{x{:}\pi_1{\to}\cdots{\to}\pi_m{\to}\varphi\}\}, \varphi\rangle) = \langle\Pi\{B_1{'}, \ldots, B_m{'}, \{x{:}\tau_1{'}{\to}\cdots{\to}\tau_m{'}{\to}\varphi\}\}, \varphi\rangle.$$

Take

$$L = \langle\langle\Pi\{B_1{'}, \ldots, B_m{'}, \{x{:}\tau_1{'}{\to}\cdots{\to}\tau_m{'}{\to}\varphi\}\}, \varphi\rangle\rangle, \langle\Pi\{B_1{''}, \ldots, B_m{''}, \{x{:}\tau_1{''}{\to}\cdots{\to}\tau_m{''}{\to}\varphi\}\}, \varphi\rangle,$$

which, by Lemma 4.22 (2), is a lifting. By Lemma 4.3 (2)

$$\vec{S}(\langle\Pi\{B_1{''}, \ldots, B_m{''}, \{x{:}\tau_1{''}{\to}\cdots{\to}\tau_m{''}{\to}\varphi\}\}, \varphi\rangle) = \langle\Pi\{B_1, \ldots, B_n, \{x{:}\sigma_1{\to}\cdots{\to}\sigma_m{\to}\varphi\}\}, \varphi\rangle.$$

Take $C = \vec{E} * \langle L\rangle * \vec{S} * \langle S, (\varphi := \sigma)\rangle$. ■

THEOREM 5.3

1. *Soundness of strict chains.* If $PP_{\mathrm{S}}(A) = \langle P, \pi\rangle$, and there exists a strict chain C such that $C(\langle P, \pi\rangle) = \langle B, \sigma\rangle$, then $B \vdash_{\mathrm{S}} A{:}\sigma$.
2. *Completeness of strict chains.* If $B \vdash_{\mathrm{S}} A{:}\sigma$, and $PP_{\mathrm{S}}(A) = \langle P, \pi\rangle$, then there exists a strict chain C such that $C(\langle P, \pi\rangle) = \langle B, \sigma\rangle$.

PROOF. 1. By Theorems 4.14, 4.25, and 4.4.
2. By Theorem 5.2. ■

5.2 Principal pairs for lambda terms

We conclude this paper by, like in [22], generalizing the concept of principal pairs to arbitrary lambda terms, using the fact that $\langle B, \sigma\rangle$ is a suitable pair for M if and only if there is an $A \in \mathcal{A}(M)$ such that $\langle B, \sigma\rangle$ is a suitable pair for A.

THEOREM 5.4 ([1])
$B \vdash_{\mathrm{S}} M{:}\sigma \iff \exists\, A \in \mathcal{A}(M)\ [\ B \vdash_{\mathrm{S}} A{:}\sigma\ ]$. ■

DEFINITION 5.5 (cf. [22])
1. For all terms M we define the set $\Pi(M)$ as follows:

$$\Pi(M) = \{\langle P, \pi\rangle \mid \exists\, A \in \mathcal{A}(M)\ [\ pp(A) = \langle P, \pi\rangle\ ]\}.$$

2. On $\mathcal{P}$ it is possible to define the preorder relation $\sqsubseteq_\omega$ by:

$$\langle P, \pi\rangle \sqsubseteq_\omega \langle P', \pi'\rangle \iff \exists\, \varphi_1, \ldots, \varphi_n\ [\langle P, \pi\rangle = (\varphi_1 := \omega)\circ\cdots\circ(\varphi_n := \omega)\ (\langle P', \pi'\rangle)\ ].$$

Since our definition of principal pairs of an approximate normal form coincides with the corresponding definition in [8] and [22], we can use the following result proved there:

THEOREM 5.6 ([8, 22])
1. $\mathcal{P}, \sqsubseteq_\omega$ is a meet semilattice, isomorphic to $\mathcal{N}, \leq$.
2. $\Pi(M)$ is an ideal in $\mathcal{P}$ and therefore:
   (a) If $\Pi(M)$ is finite, there exists a pair $\langle P, \pi\rangle = \bigsqcup\Pi(M)$, where $\langle P, \pi\rangle \in \mathcal{P}$. This pair is then called the principal pair of M.
   (b) If $\Pi(M)$ is infinite, $\bigsqcup\Pi(M)$ does not exist in $\mathcal{P}$. The principal pair of M is then the infinite set of pairs $\Pi(M)$. ■

By Theorems 5.4 and 5.3, we have the following:

THEOREM 5.7 (cf. [22])
Let $M \in \Lambda$, and B and σ such that $B \vdash_S M{:}\sigma$.

1. $\mathcal{A}(M)$ is finite. Let $\langle P, \pi\rangle$ be the principal pair of M. Then there exists a strict chain C such that $C(\langle P, \pi\rangle) = \langle B, \sigma\rangle$.
2. $\mathcal{A}(M)$ is infinite. Then there exist a pair $\langle P, \pi\rangle \in \Pi(M)$ and a strict chain C such that $C(\langle P, \pi\rangle) = \langle B, \sigma\rangle$. ■

Final remarks

In [21] a unification semi-algorithm for intersection types is presented, together with a semi-algorithm PP that finds the principal type for every strongly normalizable lambda term. The unification algorithm is built on the operations of substitution and expansion and is only proved correct when used *inside* the procedure PP. Because in this paper we prove that the strict type assignment system has the principal type property, it is possible, as in [21], to implement an efficient semi-algorithm for finding types for untyped lambda terms, using the here defined operations.

But since principal pairs for the BCD-system and the strict system coincide, this procedure PP can, as it stands, also be used to find the principal pair in the strict system for an untyped lambda term.

When the results of this paper are used to build a type-*checker*, then a unification procedure is needed that is correct for all strict types, and is built on substitution, expansion, *and* lifting, in order to be able to approve of for example the type $\alpha{\to}\beta{\to}\alpha$ for the lambda term $\lambda xy.x$.

It seems straightforward to use the results of this paper to prove the principal type property for a strict intersection type assignment system without ω, which is a strict subsystem of the intersection type assignment system without ω that was studied in [1] and [21]. There is, however, one great difficulty: the technique used in this paper cannot be used for such a proof. This is caused by the fact that the notion of type assignment without ω is not closed for β-equality, as is remarked in [1]. Take for example the two lambda terms $\lambda yz.(\lambda b.z)(yz)$ and $\lambda yz.z$. Notice that the first term reduces to the second. Let $\vdash_{-\omega}$ denote the notion of derivability in the system without ω. We know that

$$\{z{:}\sigma, y{:}\tau\} \vdash_{-\omega} z{:}\sigma,$$

but it is impossible to give a derivation for $(\lambda b.z)(yz)$:σ from the same basis without using ω. This is caused by the fact that we can only type $(\lambda b.z)(yz)$ in the system without ω from a basis in which the predicate for y is an arrow type scheme. We can for example derive

$$\{z{:}\sigma,\ y{:}\sigma{\to}\tau\} \vdash_{-\omega} (\lambda b.z)(yz){:}\sigma.$$

We can, therefore, only state that we can derive

$$\vdash_{-\omega} \lambda yz.(\lambda b.z)(yz){:}(\sigma{\to}\tau){\to}\sigma{\to}\sigma \text{ and } \vdash_{-\omega} \lambda yz.z{:}\tau{\to}\sigma{\to}\sigma$$

but we are not able to give a derivation without ω for $\lambda yz.(\lambda b.z)(yz){:}\tau{\to}\sigma{\to}\sigma$.

This means that the approach of this paper cannot be used, since there is no counterpart for Theorem 5.4 in this system.

This problem can be solved in a different way. In [21], section 6 it is overcome by defining a unification procedure for ω-free intersection types, and a proof for the statement that the procedure PP' (as specified in that section) which calls this unification procedure, returns in fact the principal pair for a lambda term within the system without ω. Notice that in the same way as the BCD-system was restricted to obtain the strict system, the system without ω of [21] can be restricted to obtain a strict system without ω. As mentioned before, the algorithm for the computation of principal pairs in both the BCD-system and the strict system is exactly the same; in both cases the procedure PP suffices. By similar reasoning, we can conclude that the algorithm PP', that computes the principal pair for a term in Ronchi's system without ω, would do so also for a strict system without ω.

References

[1] S. van Bakel. Complete restrictions of the Intersection Type Discipline. *Theoretical Computer Science*, 102:135–163, 1992.

[2] S. van Bakel. Partial Intersection Type Assignment of Rank 2 in Applicative Term Rewriting Systems. Technical Report 92-03, Department of Computer Science, University of Nijmegen, 1992.

[3] S. van Bakel. Essential Intersection Type Assignment. *Proceedings of FST&TCS '93. 13^{th} Conference on Foundations of Software Technology and Theoretical Computer Science*, 1993. To appear.

[4] S. van Bakel. Partial Intersection Type Assignment in Applicative Term Rewriting Systems. In M. Bezem, J.F. Groote, editors, *Proceedings of TLCA '93. International Conference on Typed Lambda Calculi and Applications, Utrecht, The Netherlands*, volume 664 of *Lecture Notes in Computer Science*, pages 29–44. Springer-Verlag, 1993.

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

[6] H. Barendregt, 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.

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

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

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

[10] M. Coppo and P. Giannini. A complete type inference algorithm for simple intersection types. In J.-C. Raoult, editor, *Proceedings of CAAP '92. 17th Colloquim on Trees in Algebra and Programming, Rennes, France*, volume 581 of *Lecture Notes in Computer Science*, pages 102–123. Springer-Verlag, 1992.

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

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

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

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

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

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

[17] B. Jacobs, I. Margaria, and M. Zacchi. Filter Models with Polymorphic Types. *Theoretical Computer Science*, 95:143–158, 1992.

[18] I. Margaria and M. Zacchi. Principal Typing in a $\forall\cap$-Discipline. *Journal of Logic and Computation*, to appear.

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

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

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

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

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

Received 3 August 1992