Intersection Types and the Linear Lambda-Calculus

Mário Florido and Luís Damas
University of Porto
Dep. of Computer Science-Fac. of Science & LIACC
R. do Campo Alegre 823, 4150-180 Porto, Portugal
e-mail: {amf,luis}@ncc.up.pt

Abstract

In this paper we present a notion of expansion of a term in the lambda-calculus which transforms terms into linear terms. This transformation replaces each occurrence of a variable in the original term by a fresh variable taking into account non-trivial implications in the structure of the term caused by these simple replacements. We then prove that the class of terms which can be expanded is the same of terms typable in an Intersection Type System, i.e., the strongly normalizable terms. This shows that there is a clear relation between Intersection Types, where a different type is given to different occurrences of the same variable, and linearization, where each occurrence of a shared variable is transformed into a new variable.

1 Introduction

Linear programs are simple. Concerning implementation issues, for linear programs one may safely inline the term bound to any variable, or safely update structures in place. Type inference for linear programs is easy and has nice properties which do not hold in general. In this paper we show that linear programs (linear λ-terms) are enough to fully characterize terms typable by Intersection Types. By linear λ-terms we mean terms where every λ-abstraction binds at most one variable occurrence.

Intersection type systems without the universal type ω, as presented by Coppo and Dezani in [CD80], give a characterization of strongly normalizable terms, in the sense that a term is typed in an intersection type system without ω if and only if it is strongly normalizable. These systems type more terms than the Curry type system ([Cur34]) or the type system of pure ML ([DM82]). New attention was given to intersection type systems due to a result of Kfoury and Wells ([KW99]) which proved that these systems are decidable for restrictions of finite rank, which correspond to a large class of typable terms. Although Kfoury and Wells's work made the technology of intersection types more accessible avoiding the use of the complicated notion of type expansion, intersection type systems are rather complex systems whose definitions can be quite difficult to understand.

Intersection type systems are based on the idea of explicitly typing with a different type each different occurrence of a variable. In this way they are capable of typing terms where sharing of variables with non-unifiable types may cause non-typification in other type systems. For example, in Intersection Type Systems $\lambda x.xx$ has got type $(\alpha \cap \alpha \to \beta) \to \beta$. A more interesting example is the term $T \equiv (\lambda x.xx)I$, where I is the identity function $\lambda x.x$. This term has got type $\alpha \to \alpha$ which does not involve intersections, although it is not typable in the Curry Type System, because it has a non-typable subterm. However notice that there is a linear term, $(\lambda x_1 x_2.x_1 x_2)II$, which has the same type in the Curry Type System, and which is a linear version of $(\lambda x.xx)I$ in the sense that each occurrence of a shared variable in $(\lambda x.xx)I$ corresponds to a distinct variable in $(\lambda x_1 x_2.x_1 x_2)II$.

In this paper we address the following problem: to which extent can we approximate a typed term in the intersection type system by a linear term?

The first difficulty in answering this question is what is meant by *linear version* of a term. Consider the following example: Suppose one wants to define the linear version of the term $(\lambda xy.xy)(\lambda z.zz)(\lambda x.x)$. In this term the only variable which occurs more than once is z. Thus we can expand $(\lambda z.zz)$ to get the term $(\lambda z_1 z_2.z_1 z_2)$. But $(\lambda z.zz)$ will have y as an argument after one reduction step. Thus y in $(\lambda xy.xy)$ has to be copied and we get the term:

$$(\lambda xy.xyy)(\lambda z_1 z_2.z_2 z_2)(\lambda x.x)$$

Now, a variable which occurred once in the original term occurs twice in the expanded term, thus the expansion process has to go on, expanding $(\lambda xy.xyy)$ to $(\lambda xy_1y_2.xy_1y_2)$. Notice that y will be replaced by $\lambda x.x$ in the original version of the term, thus, as y was expanded to two new variables we have to duplicate $(\lambda x.x)$ in the expanded term to get the final term:

$$(\lambda xy_1y_2.xy_1y_2)(\lambda z_1 z_2.z_1 z_2)(\lambda x.x)(\lambda x.x)$$

The main contributions of this paper are the following:

- A notion of *term expansion*. Under that notion we show that one can define the *linear version* of a term if and only if the term is typable in an intersection type system which types exactly the strongly normalizable terms.
- To show that every term typable in an intersection type system has an expansion which is a linear term typable in the Curry type system (which is the simplest type system for the λ-calculus).
- If a term M is typable in an intersection type system with a simple type, then there is an expansion of M which is a linear term typable in the Curry type system with the same type.

These results have theoretical implications because we fully characterizes a large class of functions (the functions definable by the strongly normalizable terms) with a rather simple class: the linear λ-terms. They also have practical implications because, although our notion of *expansion* is not decidable (because otherwise knowing if a term is strongly normalizable would be decidable, which it is not) it characterizes the same set of terms as intersection type systems, for which, there are decidable classes with impact in programming language technology. This suggests that decidable restrictions of our framework may be successfully defined reducing the type inference problem to the linear case for some interesting classes of programs.

In this paper we assume that the reader is familiar with the λ-calculus and type systems for the λ-calculus. Good surveys of the area are [Bar92] and [Hin97].

We start in Section 2 with the presentation of related work. Section 3 defines the type systems that are going to be used later in the paper. Section 4 defines the notion of expansion. In section 5 we show the relation with intersection types. Finally we outline some future work in section 6.

2 Related Work

The linearization problem was the topic of a paper by Kfoury [Kfo00] where a new calculus was defined satisfying the linearity condition. This condition states that in an abstraction $\lambda x.M$ the free occurrences of x in M are in one-one correspondence with the arguments to which the function is applied. Kfoury enlarged the λ-calculus with a new kind of terms of the form $M.P_1 \wedge \ldots \wedge P_n$ and a new reduction $\beta^\wedge$, which, when the redex is of the form $(\lambda x.M)P_1 \wedge \ldots \wedge P_n$ and x occurs free is M n times, replaces the i^{th} occurrence of x by P_i $(1 \leq i \leq n)$. This reduction was made in parallel for every P_i. Finally an intersection type system for this new calculus was defined and used to give another proof of the well-known equivalence between strongly normalizing λ-terms and terms typable in an intersection type system. The main differences in our approach are:

1. We do not leave the λ-calculus. We force the linearity condition by transforming non-linear terms into linear ones and continue to use the β-reduction while Kfoury forces the linearity condition by changing the calculus and the reduction rule.
2. Kfoury uses an intersection type system to type the expanded versions of terms. Our approach is based on doing transformations on the term language that enables the use of the Curry type system instead of using a more complex type system for the original set of terms.

Transformation of terms typable in an intersection type system into terms typable in the Curry type systems was the topic of a paper by Bucciarelli, De Lorenzis, Piperno and Salvo ([BLPS99]). In this paper the authors presented an elegant translation from derivations in an intersection type system to derivations in the Curry type system. They used this result to prove that all functions uniformly defined using intersection types are also definable using Curry types. The main differences in our approach are the following:

1. In [BLPS99] it was proved that terms typable in an intersection type system can be characterized by terms typable in the Curry type system. Here we give an alternative characterization of strong normalization using a more restrictive class of terms: the linear terms.
2. The transformation presented in [BLPS99] was dependent of type derivations in an intersection type system. Our notion of expansion does not depend on the type derivation. It is a definition by induction on the terms which can be used as a basis to more specific term transformation processes.

The linear λ-calculus was also studied in a different context by the linear logic community [Abr93, LM92, Wad90, BBdH92]. In these works the linear λ-calculus is defined as the computational interpretation of linear logic and the problem of transforming non-linear terms into linear ones is not a central issue.

3 Types

Here we describe the type systems used in the rest of the paper.

3.1 Curry Types

The Curry type system ([Cur34]) (sometimes called the Simple type system) is the basis of every type system for the λ-calculus. Here we describe this system along the lines presented in [Hin97]. From now on terms of the λ-calculus are considered module α-equivalence. We also assume that in a term M no variable is bound more than once and no variable occurs both free and bound in M.

Definition 1 *An infinite sequence of type-variables is assumed to be given. Simple types are expressions defined thus:*

1. *each type-variable is a simple type;*
2. *if σ and τ are simple types then $(\tau \to \sigma)$ is a simple type.*

Definition 2 *A basis is a finite list of pairs of the form $x : \tau$, where x is a term variable and τ is a simple type, and which is consistent in the sense that the term variables are all distinct.*

Simple type derivations are defined thus

$$VAR \qquad \{x:\tau\} \vdash_C x : \tau$$

$$ABS-I \qquad \frac{A \cup \{x:\tau\} \vdash_C M : \sigma}{A \vdash_C \lambda x.M : \tau \to \sigma}$$

$$ABS-K \qquad \frac{A \vdash_C M : \sigma}{A \vdash_C \lambda x.M : \tau \to \sigma}, \text{ if } x \notin FV(M)$$

$$APP \qquad \frac{A \vdash_C M : \tau \to \sigma, B \vdash_C N : \tau}{A \cup B \vdash_C MN : \sigma}, \text{ if } A \cup B \text{ is consistent}$$

$M:\sigma$ is derivable from a basis A in the Curry type system, notation $A \vdash_C M:\sigma$, if and only if it is obtained using the previous rules.

Usually the Curry type system is defined using the same basis in the application rule and a variable rule of the form:

$$A \vdash x:\tau, \text{ if } x:\tau \in A$$

The definition presented here guarantees that every type in the basis is used in the type derivation. This feature is going to be important in subsequent results.

Notice that the two definitions are equivalent in the sense that $A \vdash_C M:\sigma$ if and only if there is a basis $B \supseteq A$ such that $B \vdash M:\sigma$ in the usual definition.

3.2 Intersection Types

Intersection types were introduced in [CD80] as powerful type systems with the ability of typing different occurrences of the same variable with different types. This makes it possible to type "dangerous" self-applications such as $\lambda x.xx$. Notice that $\lambda x.xx$ is not typable in the Curry type system and in ML. Here we define an intersection type system along the lines presented in [KW99]. In this type system every type declared in the environment is used in the type derivation, a property which is going to be crucial in subsequent results.

Definition 3 *An infinite sequence of type-variables is assumed to be given. Intersection types are expressions defined thus:*

1. *each type-variable is a type;*
2. *if σ and $\tau_1 \ldots \tau_n$ are types then $(\tau_1 \cap \ldots \cap \tau_n \to \sigma)$ is a type.*

Definition 4 *A type environment is a finite list of pairs of the form $x:\tau_1 \cap \ldots \cap \tau_n$, where x is a term variable, $\tau_1 \ldots \tau_n$ are types, and the term variables are all distinct.*

Definition 5 *Let A_1 and A_2 be two type environments. Then $A_1 \wedge A_2$ is the new environment given by $x:\sigma \in A_1 \wedge A_2$ if and only if σ is defined thus*

$$\sigma = \begin{cases} \sigma_1 \cap \sigma_2 & \textit{if } x:\sigma_1 \in A_1 \textit{ and } x:\sigma_2 \in A_2 \\ \sigma_1 & \textit{if } x:\sigma_1 \in A_1 \textit{ and } \neg\exists_\sigma.x:\sigma \in A_2 \\ \sigma_2 & \textit{if } x:\sigma_2 \in A_2 \textit{ and } \neg\exists_\sigma.x:\sigma \in A_1 \end{cases}$$

The intersection type system used in the rest of the paper is defined thus:

$$VAR \qquad \{x:\tau\} \vdash x:\tau$$

$$ABS-I \qquad \frac{A \cup \{x:\tau_1 \cap \ldots \cap \tau_n\} \vdash M:\sigma}{A \vdash \lambda x.M:\tau_1 \cap \ldots \cap \tau_n \to \sigma}$$

$$ABS-K \qquad \frac{A \vdash M:\sigma}{A \vdash \lambda x.M:\tau \to \sigma}, \text{ if } x \notin FV(M)$$

$$APP \qquad \frac{A_0 \vdash M:\tau_1 \cap \ldots \cap \tau_n \to \sigma, A_1 \vdash N:\tau_1 \ldots A_n \vdash N:\tau_n}{A_0 \wedge A_1 \wedge \ldots \wedge A_n \vdash MN:\sigma}$$

The two different ABS rules are necessary because in this system if there is a derivation of $A \vdash M:\sigma$ and $x \notin FV(M)$ then there is not a type declaration for x in A. The set of types for a given term M in this system is strictly included in the set of types for M in the original intersection type system of Coppo and Dezani ([CD80]). For example the type $(\alpha_1 \cap \alpha_2) \to \alpha_1$ types $\lambda x.x$ in the Coppo-Dezani type system but not in the system used in this paper. The reason for this is that types in intersections for free variables can only be introduced with the APP rule and thus each element of the intersection corresponds to a type that is actually used in the type derivation. However the set of terms typable in both system is the same and corresponds to the strongly normalizable terms.

Theorem 1 *A λ-term M is strongly normalizable (i.e. with no infinite reduction sequences starting from M) if and only if M is typable in the intersection type system presented.*

Proof (Sketch): The *if* part is proved by transforming a derivation in our type system in a derivation in the Coppo-Dezani type system (which types all strongly normalizable terms). This can be done by induction in the length of the derivation tree. The *only-if* part is similar to proofs of the same property of other intersection type systems (see, for example, [Pot80] or [vB93]). The technique used is to show that the intersection type system types terms in normal form. Then we show that if $A \vdash C[M[N/x]] : \tau$ and N is typable then $A \vdash C[(\lambda x.M)N] : \tau$ ($C[]$ is the usual notation for a context). Finally the result follows by induction on the inside-out reduction path of the term, noticing that if M is strongly normalizable then the normal form of M can be reached using the inside-out strategy and that this strategy has the property that a redex $(\lambda x.M)N$ can only be contracted if N is in normal form (and thus typable). $\square$

4 Expansions

In this section we define the notion of expansion of a λ-term. Expansion consists of replacing each occurrence of a variable in a term by a new variable. This operation may involve other transformations in the term. For example if x is expanded k times in $(\lambda x.M)N$ then N has to be copied k times. However if the expansion is inside N then M may be changed, because possible arguments of x may have to be copied.

To define the expansion we face one key problem: the expansion of MN is a term of the form $M_0N_1 \ldots N_k$ where M_0 is the expansion of M and $N_1 \ldots N_k$ are expansions of N. The problem here is to find the right k. The number of new arguments is easy to determinate when M is of the form $\lambda x.M'$ (just check how many fresh variables replace x), but if M is itself an application this information depends on expansions made inside M. The best way to propagate this information is by using types. As we need to explicitly count the number of types of each function argument we use intersection types. If M has type $\tau_1 \cap \ldots \cap \tau_k \to \sigma$ in the intersection type system we know that MN will be expanded to a term of the form $M_0N_1 \ldots N_k$. We now formalize these ideas by defining the expansion operation.

Let us first formalize the expansion of free variables:

Definition 6 *A variable expansion is an expression of the form*

$$x : S$$

where x is a variable and S is a sequence of pairs of the form $y : \tau$ where y is a term and τ an intersection type. ($x : S$ should be read informally as "x expands to the variables in S".)

Definition 7 *An expansion context A is any finite sequence of variable expansions*

$$A = \{x_1 : S_1, \ldots, x_n : S_n\}$$

where the variables $\{x_1 \ldots x_n\}$ are all different and the S_i are disjoint.

Let $++$ denote the concatenation of two sequences. We now define an operation which appends two expansion contexts.

Definition 8 *Let A_1 and A_2 be two expansion contexts. Then $A_1 \uplus A_2$ is a new context such that $x : S \in A_1 \uplus A_2$ if and only if S is defined thus*

$$S = \begin{cases} S_1{+}{+}S_2 & \text{if } x : S_1 \in A_1 \text{ and } x : S_2 \in A_2 \\ S_1 & \text{if } x : S_1 \in A_1 \text{ and } \neg\exists_S . x : S \in A_2 \\ S_2 & \text{if } x : S_2 \in A_2 \text{ and } \neg\exists_S . x : S \in A_1 \end{cases}$$

We are now able to formalize the notion of term expansion:

Definition 9 *Given a pair $M : \sigma$, where M is a term and σ an intersection type, a term N and an expansion context A we define a relation called expansion (written $\mathcal{E}(M : \sigma) \lhd (N, A)$). If A is empty we shall write just $\mathcal{E}(M : \sigma) \lhd N$. Expansions are defined thus*

$$\begin{array}{rcl}
\mathcal{E}(x : \tau) & \lhd & (y, \{x :< y : \tau >\}) \\
& & \quad \textit{if } x \textit{ is a variable and } y \textit{ is a fresh variable} \\
\\
\mathcal{E}(\lambda x.M : \tau_1 \cap \ldots \cap \tau_n \to \sigma) & \lhd & (\lambda x_1 \ldots x_n.M^*, A) \\
& & \quad \textit{if } x \textit{ occurs in } M \textit{ and} \\
& & \quad \mathcal{E}(M : \sigma) \lhd (M^*, A \cup \{x :< x_1 : \tau_1, \ldots, x_n : \tau_n >\}) \\
\\
\mathcal{E}(\lambda x.M : \tau \to \sigma) & \lhd & (\lambda y.M^*, A) \\
& & \quad \textit{if } x \textit{ does not occur in } M, \\
& & \quad y \textit{ is a fresh variable and} \\
& & \quad \mathcal{E}(M : \sigma) \lhd (M^*, A) \\
\\
\mathcal{E}(MN : \sigma) & \lhd & (M_0 N_1 \ldots N_k, A_0 \uplus A_1 \uplus \ldots \uplus A_n) \\
& & \quad \textit{if for some } k \textit{ and } \tau_1, \ldots \tau_k, \\
& & \quad \mathcal{E}(M : \tau_1 \cap \ldots \cap \tau_k \to \sigma) \lhd (M_0, A_0) \textit{ and} \\
& & \quad \mathcal{E}(N : \tau_i) \lhd (N_i, A_i), (1 \leq i \leq k)
\end{array}$$

Remark: It is possible that in order to define $\mathcal{E}(M : \sigma)$ we must take a representation of σ with some redundancy, e.g. instead of:

$$\mathcal{E}(\lambda x.x(xx) : ((\alpha \to \alpha) \cap \alpha) \to \alpha)$$

we may require:

$$\mathcal{E}(\lambda x.x(xx) : ((\alpha \to \alpha) \cap (\alpha \to \alpha) \cap \alpha) \to \alpha)$$

in order to get a number of types in the intersection which is the same as the free occurrences of the parameter in the function body.

As we shall see, expansion is an operation which linearizes the terms. By linear we mean exactly the following set of terms:

Definition 10 *A linear λ-term is a λ-term M such that:*

1. *for each subterm $\lambda x.N$ of M, x occurs free in N at most once,*
2. *each free variable of M has just one occurrence in M.*

This definition follows the definition of linear terms presented in [Kfo00]. Note that some people call this class of terms *affine*, and use the word *linear* for terms $\lambda x.M$ where x occurs free exactly once in M.

Lemma 1 *For all N, $M : \sigma$ and A, if $\mathcal{E}(M : \sigma) \lhd (N, A)$, then N is a linear term.*

Proof: By structural induction on M. The cases where M is a variable and a term of the form $\lambda x.N$ are trivial by structural induction. We just present the case where $M \equiv M_1 M_2$. In this case

$$\mathcal{E}(M_1 M_2 : \sigma) \lhd (M_0 N_1 \ldots N_k, A_0 \uplus A_1 \uplus \ldots \uplus A_n)$$

and

1. $\mathcal{E}(M_1 : \tau_1 \cap \ldots \cap \tau_n \to \sigma) \lhd (M_0, A_0)$
2. $\mathcal{E}(M_2 : \tau_i) \lhd (N_i, A_i), (1 \leq i \leq k)$

By the induction hypothesis M_0 and $N_1, \ldots, N_k$ are linear. Notice that the free variables of M_0, $N_1, \ldots, N_k$ are all different, because the expansion of an occurrence of a variable creates a fresh variable. Thus $M_0 N_1 \ldots N_k$ is linear. $\Box$

We will now present some examples:

Example 1 *Let $I \equiv \lambda x.x$ and $M \equiv \lambda x.xx$. Let us show step by step how to calculate an expansion of $MI : \alpha \to \alpha$:*

$$\mathcal{E}(x : (\alpha \to \alpha) \to (\alpha \to \alpha)) \lhd (x_1, \{x :< x_1 : (\alpha \to \alpha) \to (\alpha \to \alpha) >\})$$

and

$$\mathcal{E}(x : \alpha \to \alpha) \lhd (x_2, \{x :< x_2 : \alpha \to \alpha >\})$$

thus

$$\mathcal{E}(xx : \alpha \to \alpha) \lhd (x_1 x_2, \{x :< x_1 : (\alpha \to \alpha) \to (\alpha \to \alpha), x_2 : \alpha \to \alpha >\})$$

and

$$\mathcal{E}(\lambda x.xx : (((\alpha \to \alpha) \to (\alpha \to \alpha)) \cap (\alpha \to \alpha)) \to \alpha \to \alpha) \lhd \lambda x_1 x_2.x_1 x_2$$

It easy to show that

$$\mathcal{E}(I : \alpha \to \alpha) \lhd I$$

and

$$\mathcal{E}(I : (\alpha \to \alpha) \to (\alpha \to \alpha)) \lhd I$$

thus

$$\mathcal{E}(((\lambda x.xx)I) : \alpha \to \alpha) \lhd (\lambda x_1 x_2.x_1 x_2)II$$

Example 2 *Let $M \equiv (\lambda xy.xy)(\lambda x.xx)z$. Then*

$$\mathcal{E}(M : \beta) \lhd ((\lambda x_1 y_1 y_2.x_1 y_1 y_2)(\lambda x_2 x_3.x_2 x_3) z_1 z_2, \{z :< z_1 : \alpha \to \beta, z_2 : \alpha >\})$$

Example 3 *Let $M \equiv (I(\lambda y.yyy))I$ Then*

$$\mathcal{E}(M : \alpha \to \alpha) \lhd (I(\lambda y_1 y_2 y_3.y_1 y_2 y_3))III$$

Example 4 *Let $I \equiv \lambda x.x$ and $M \equiv (\lambda f.f(\lambda x.xx)(fI))I$ Then*

$$\mathcal{E}(M : \alpha \to \alpha) \lhd (\lambda f_1 f_2 f_3.f_1(\lambda x_1 x_2.x_1 x_2)(f_2 I)(f_3 I))III$$

In this example notice the use of type information to control the number of expansions. In$(\lambda f.f(\lambda x.xx)(fI))I$ the fact that f is going to be the identity function gives f three different types, one for the identity function applied to $\lambda x.xx$, and two more types, one for each type in the intersection in the argument type of $f(\lambda x.xx)$. These three types give rise to the three new expansion variables f_1, f_2 and f_3. The intersection of two types in the argument type of $f(\lambda x.xx)$ gives rise to the two new terms $(f_2 I)$ and $(f_3 I)$.

Example 5 *Let $M \equiv (\lambda vxyz.v(y(vxz)))I$. Then*

$$\mathcal{E}(M : (\alpha \to \beta) \to (\beta \to \sigma) \to \alpha \to \sigma) \lhd (\lambda v_1 v_2 xyz.v_1(y(v_2 xz)))II$$

This last example is used in Hindley's book [Hin97] to show that a term M may have a principal type in the Curry type system which is less general than the principal type of its normal form. In fact, in the Curry type system, M has principal type $(\alpha \to \beta) \to (\beta \to \alpha \to \beta) \to \alpha \to \alpha \to \beta$ and its normal form, $\lambda xyz.y(xz)$, has principal type $(\alpha \to \beta) \to (\beta \to \sigma) \to \alpha \to \sigma$. The reason for this is that in M the two occurrences of v must have the same type, but when v is replaced by I during the reduction process we drop that restriction. The interesting fact is that the expansion of M, $\lambda v_1 v_2 xyz.v_1(y(v_2 xz)))II$, has type $(\alpha \to \beta) \to (\beta \to \sigma) \to \alpha \to \sigma$ in the Curry type system, because when we expand v we do not have to impose the same type for v_1 and v_2.

4.1 Expansion and Types

In [BLPS99] a translation from intersection types to Curry types was given and used to show that terms typed in an intersection type system can be transformed into terms typed in the Curry type system. Here we show that our definition of expansion preserves this translation. In fact, let $\mathcal{T}$ be the translation from intersection types to Curry types defined in [BLPS99]. Then, if M is typable in the intersection type system with type σ, and $\mathcal{E}(M:\sigma) \lhd (N, A)$ then N is typable in the Curry type system with type $\mathcal{T}(\sigma)$.

Definition 11 *$\mathcal{T}$ is a translation from intersection types to Curry types defined thus:*

1. *$\mathcal{T}(\alpha) = \alpha$, if α is a type variable;*
2. *$\mathcal{T}((\tau_1 \cap \ldots \cap \tau_n) \to \sigma) = \mathcal{T}(\tau_1) \to \ldots \to \mathcal{T}(\tau_n) \to \mathcal{T}(\sigma)$.*

The previous definition can be extended to expansion contexts:

Definition 12 *Let $\mathcal{T}_e$ be a translation from expansion contexts to bases defined thus:*

1. *$\mathcal{T}_e(\emptyset) = \emptyset$;*
2. *$\mathcal{T}_e(A \cup \{x :< x_1 : \tau_1, \ldots, x_n : \tau_n >\}) = \mathcal{T}_e(A) \cup \{x_1 : \mathcal{T}(\tau_1), \ldots, x_n : \mathcal{T}(\tau_n)\}$.*

Theorem 2 *Let $\mathcal{E}(M:\sigma) \lhd (N, A)$. Then $\mathcal{T}_e(A) \vdash_C N : \mathcal{T}(\sigma)$, where $\vdash_C$ stands for type derivation in the Curry type system.*

Proof: By structural induction on M.

1. Base case. M is a term variable x. In this case $\mathcal{E}(x:\tau) \lhd (y, \{x :< y : \tau >\})$. $\mathcal{T}_e(\{x :< y : \tau >\}) = \{y : \mathcal{T}(\tau)\}$. The result follows by the VAR rule for the Curry type system.
2. Induction step:

   (a) M is of the form $\lambda x.N$. Suppose that x occurs free in M. Then

   $$\mathcal{E}(\lambda x.N : \tau_1 \cap \ldots \cap \tau_n \to \sigma) \lhd (\lambda x_1 \ldots x_n.N^*, A)$$

   and thus

   $$\mathcal{E}(N:\sigma) \lhd (N^*, A \cup \{x :< x_1 : \tau_1 \ldots x_n : \tau_n >\})$$

   By the induction hypothesis and the definition of $\mathcal{T}_e$:

   $$\mathcal{T}_e(A) \cup \{x_1 : \mathcal{T}(\tau_1), \ldots, x_n : \mathcal{T}(\tau_n)\} \vdash_C N^* : \mathcal{T}(\sigma)$$

   Thus, by successive applications of the ABS-I rule:

   $$\mathcal{T}_e(A) \vdash_C \lambda x_1 \ldots x_n.N^* : \mathcal{T}(\tau_1) \to \ldots \to \mathcal{T}(\tau_n) \to \mathcal{T}(\sigma)$$

   The result follows by the definition of $\mathcal{T}$. The case where x does not occur in N is similar, thus we will omit it.

   (b) M is of the form $(M_1 M_2) : \sigma$. In this case:

      i. $\mathcal{E}(M_1 : \tau_1 \cap \ldots \cap \tau_n \to \sigma) \lhd (M_0, A_0)$
      ii. $\mathcal{E}(M_2 : \tau_i) \lhd (N_i, A_i)(1 \leq i \leq n)$

   By the induction hypothesis

      i. $\mathcal{T}_e(A_0) \vdash_C M_0 : \mathcal{T}(\tau_1) \to \ldots \to \mathcal{T}(\tau_n) \to \mathcal{T}(\sigma)$
      ii. $\mathcal{T}_e(A_i) \vdash_C N_i : \mathcal{T}(\tau_i)$, for $(1 \leq i \leq n)$.

   Notice that the variables in $A_0, \ldots, A_n$ are all distinct, because expansion contexts are generated with expansions of occurrences of free variables, which always generate fresh variables. Thus the same happens for $\mathcal{T}_e(A_0), \ldots, \mathcal{T}_e(A_n)$. This guarantees that in $\mathcal{T}_e(A_0) \cup \ldots \cup \mathcal{T}_e(A_n)$ all variables are distinct. Thus

   $$\mathcal{T}_e(A_0) \cup \ldots \cup \mathcal{T}_e(A_n) \vdash_C M_0 N_1 \ldots N_n : \mathcal{T}(\sigma)$$

   □

5 Intersection Types and Expansion

In this section we will show some properties of expansion. The main result states that terms that we can expand are exactly the terms typable in an Intersection Type System. This result shows that there is a clear connection between intersection type systems, which give different types to different occurrences of the same variable and expansion where each occurrence of a variable is replaced by a new variable.

Let us first define two functions which transform expansion contexts in type environments and vice versa.

Definition 13 *Let A be a type environment and $\{x_1, \dots x_x\}$ be fresh term variables. $e(A)$ is the expansion context defined thus:*

$$e(A) = \{x :< x_1 : \tau_1, \dots, x_n : \tau_n > | x : \tau_1 \cap \dots \cap \tau_n \in A\}$$

Definition 14 *Let A be an expansion context. $l(A)$ is the type environment defined thus:*

$$l(A) = \{x : \tau_1 \cap \dots \cap \tau_n | x :< x_1 : \tau_1, \dots, x_n : \tau_n > \in A\}$$

Lemma 2 *Let A_1 and A_2 be type environments. Then*

$$e(A_1) \uplus e(A_2) = e(A_1 \wedge A_2)$$

Proof: By the definitions of $\wedge$ and $\uplus$. $\square$

Lemma 3 *Let A_1 and A_2 be two expansion contexts. Then*

$$l(A_1) \wedge l(A_2) = l(A_1 \uplus A_2)$$

Proof: By the definitions of $\wedge$ and $\uplus$. $\square$

Lemma 4 *Let M be a λ-term such that there is an environment A and an intersection type σ such that $A \vdash M : \sigma$. Then there is a linear term N such that $\mathcal{E}(M : \sigma) \lhd (N, e(A))$.*

Proof: By structural induction on M.

1. Base case: M is a term-variable x. $\{x : \tau\} \vdash x : \tau$. Then $\mathcal{E}(x : \tau) \lhd (y, \{x :< y : \tau >\})$ where y is a fresh variable. The result follows noticing that $\{x :< y : \tau >\} = e(\{x : \tau\})$.

2. Induction step:

   (a) M is of the form $\lambda x.N$ and x occurs in N. In this case $A \cup \{x : \tau_1 \cap \dots \cap \tau_n\} \vdash N : \sigma$. By the induction hypothesis,

   $$\mathcal{E}(N : \sigma) \lhd (N', e(A) \cup \{x :< x_1 : \tau_1, \dots, x_n : \tau_n >\})$$

   Thus by the definition of expansion

   $$\mathcal{E}(\lambda x.N : \tau_1 \cap \dots \cap \tau_n \to \sigma) \lhd (\lambda x_1 \dots x_n.N', e(A))$$

   (b) M is of the form $\lambda x.N$ and x does not occur in N. In this case $A \vdash N : \sigma$. By the induction hypothesis,

   $$\mathcal{E}(N : \sigma) \lhd (N', e(A))$$

   and by the definition of expansion

   $$\mathcal{E}(\lambda x.N : \tau \to \sigma) \lhd (\lambda y.N', e(A))$$

(c) M is of the form $M_1 M_2$. In this case we have

$$A_0 \wedge A_1 \wedge \ldots \wedge A_n \vdash M_1 M_2 : \sigma$$

Thus

i. $A_0 \vdash M_1 : \tau_1 \cap \ldots \cap \tau_n \to \sigma$
ii. $A_i \vdash M_2 : \tau_i (1 \leq i \leq n)$

By the induction hypothesis:

i. $\mathcal{E}(M_1 : \tau_1 \cap \ldots \cap \tau_n \to \sigma) \lhd (M_0, e(A_0))$
ii. $\mathcal{E}(M_2 : \tau_i) \lhd (N_i, e(A_i)), (1 \leq i \leq n)$

Thus, by the definition of expansion,

$$\mathcal{E}(M_1 M_2 : \sigma) \lhd (M_0 N_1 \ldots N_n, e(A_0) \uplus \ldots \uplus e(A_n))$$

and finally by lemma 2

$$\mathcal{E}(M_1 M_2 : \sigma) \lhd (M_0 N_1 \ldots N_n, e(A_0 \wedge \ldots \wedge A_n)) \square$$

Theorem 3 *Let M be a λ-term such that $A \vdash M : \sigma$ in the intersection type system. Then there is a basis B and a linear term N such that $B \vdash_C N : \mathcal{T}(\sigma)$, where $\vdash_C$ stands for type derivation in the Curry type system.*

Proof: By lemma 4, $A \vdash M : \sigma \Rightarrow \mathcal{E}(M : \sigma) \lhd (N, e(A))$. The result follows by theorem 2. $\square$

This theorem has, as a corollary, that if a term M is typable in the intersection type system with a Curry type, then there is a linear term with the same type derivable in the Curry type system. Just notice that $\mathcal{T}(\sigma) = \sigma$ when σ is a Curry type.

Lemma 5 *Let M be a λ-term such that there is a context A, an intersection type σ, and a linear term N such that $\mathcal{E}(M : \sigma) \lhd (N, A)$. Then $l(A) \vdash M : \sigma$.*

Proof: By structural induction on M.

1. Base case: M is a term variable x. In this case $\mathcal{E}(x : \tau) \lhd (y, \{x :< y : \tau >\})$. We have $l(\{x :< y : \tau >\}) = \{x : \tau\}$. Finally $\{x : \tau\} \vdash x : \tau$.

2. Induction step:

(a) M is of the form $\lambda x.N$, and x occurs in N.

$$\mathcal{E}(\lambda x.N : (\tau_1 \cap \ldots \cap \tau_n \to \sigma)) \lhd (\lambda x_1 \ldots x_n.N', A)$$

where

$$\mathcal{E}(N : \sigma) \lhd (N', A \cup \{x :< x_1 : \tau_1, \ldots, x_n : \tau_n >\})$$

By the induction hypothesis

$$l(A) \cup \{x : \tau_1 \cap \ldots \cap \tau_n\} \vdash N : \sigma$$

Thus

$$l(A) \vdash \lambda x.N : \tau_1 \cap \ldots \cap \tau_n \to \sigma$$

(b) M is of the form $\lambda x.N$ and x does not occur in N.

$$\mathcal{E}(\lambda x.N : \tau \to \sigma) \lhd (\lambda y.N', A)$$

where

$$\mathcal{E}(N : \sigma) \lhd (N', A)$$

By the induction hypothesis

$$l(A) \vdash N : \sigma$$

Thus

$$l(A) \vdash \lambda x.N : \tau \to \sigma$$

(c) M is of the form $M_1 M_2$. In this case we have:

$$\mathcal{E}((M_1 M_2) : \sigma) \lhd (M_0 N_1 \dots N_k, A_0 \uplus \dots \uplus A_k)$$

and

i. $\mathcal{E}(M_1 : \tau_1 \cap \dots \cap \tau_n \to \sigma) \lhd (M_0, A_0)$

ii. $\mathcal{E}(M_2 : \tau_i) \lhd (N_i, A_i)(1 \leq i \leq n)$

By the induction hypothesis

i. $l(A_0) \vdash M_1 : \tau_1 \cap \dots \cap \tau_n \to \sigma$

ii. $l(A_i) \vdash M_2 : \tau_i (1 \leq i \leq n)$

Thus

$$l(A_0) \wedge \dots \wedge l(A_n) \vdash M_1 M_2 : \sigma$$

Finally, by lemma 3,

$$l(A_0 \uplus \dots \uplus A_n) \vdash M_1 M_2 : \sigma \square$$

Theorem 4 *Let M be a λ-term. Then M is typable in the Intersection Type System presented if and only if there are a linear term N, an expansion context A and a type σ such that $\mathcal{E}(M : \sigma) \lhd (N, A)$.*

Proof: By lemmas 4 and 5. $\square$

6 Future Work

Here we present a transformation of terms typable in an Intersection Type System into linear terms. Intersection Type Systems are not decidable (otherwise, we could automatically determine whether a term was strongly normalizable, which is an undecidable property). Finding out classes of λ-terms with a decidable notion of expansion would be useful for program transformation and it is left as future work.

Acknowledgements The work presented in this paper has been partially supported by funds granted to *LIACC* through the *Programa de Financiamento Plurianual, Fundação para a Ciência e Tecnologia* and *Programa POSI.*

References

[Abr93] S. Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 1993.

[Bar92] H. Barendregt. Lambda calculi with types. In Abramsky, Gabbay, and Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Oxford Science Publications, 1992.

[BBdH92] N. Benton, G. Bierman, V. de Paiva, and M. Hyland. Term assignement for intuitionistic linear logic. Technical report, Computing Laboratory, University of Cambridge, 1992.

[BLPS99] A. Bucciarelli, S. De Lorenzis, A. Piperno, and I. Salvo. Some computational properties of intersection types. In *14th IEEE Symposium on Logic in Computer Science*, 1999.

[CD80] 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.

[Cur34] H.B. Curry. Functionality in combinatory logic. In *Proc. Nat. Acad. Sci. U.S.A.*, volume 20, pages 584–590, 1934.

[DM82] L. Damas and R. Milner. Principal type schemes for functional languages. In *9th ACM Symposium on Principles of Programming Languages*, 1982.

[Hin97] R. Hindley. *Basic Simple Type Theory.* Cambridge University Press, 1997.

[Kfo00] A. J. Kfoury. A linearization of the lambda-calculus. *Journal of Logic and Computation*, 2000.

[KW99] A.J. Kfoury and J.B. Wells. Principality and decidable type inference for finite-rank intersection types. In *Conf. Rec. POPL'99:26th ACM Symp. Principles of Prog. Languages*, 1999.

[LM92] P. Lincoln and J. Mitchell. Operational aspects of linear lambda calculus. In *7th Symposium on Logic in Computer Science*, 1992.

[Pot80] G. Pottinger. A type assignement for the strongly normalizable terms. In J.R. Hindley and J.P. Seldin, editors, *To H.B. Curry, Essays in combinatory logic, lambda-calculus and formalism*, pages 561–577. Academic Press, 1980.

[vB93] S. van Bakel. *Intersection Type Disciplines in Lambda Calculus and Applicative Term Rewriting Systems.* PhD thesis, Department of Computer Science, University of Nijmegen, 1993.

[Wad90] P. Wadler. Linear types can change the world! In M. Broy and C. Jones, editors, *Programming Concepts and Methods.* North Holland, 1990.

A Strong Normalization for Intersection Types

Here we present a proof of theorem 1 and several lemmas needed to the main proof.

Lemma 6 *If $A \vdash C[M[N/x]] : \tau$ and there is an environment B and a type σ such that $B \vdash N : \sigma$, then $A \vdash C[(\lambda x.M)N] : \tau$. $C[...]$ is the notation for a context.*

Proof: By induction in the structure of contexts. We will omit the case where the context is an application, since it is trivial using the induction hypothesis.

1. $C[M[N/x]] \equiv M[N/x]$. We can assume that x does not occur in A.

   (a) N occurs n times in $M[N/x]$, each time typed by τ_1. Then we replace each $B_i \vdash N : \tau_i (1 \leq i \leq n)$ in the derivation tree of $A \vdash C[M[N/x]] : \tau$ by $\{x : \tau_i\} \vdash x : \tau_i$ to get a derivation $A' \cup \{x : \tau_1 \cap \ldots \cap \tau_n\} \vdash M : \tau$. Then $A' \wedge B_1 \ldots \wedge B_n \vdash (\lambda x.M)N : \tau$. A' is build from A by taking every declaration and type information introduced by the environments $B_1, \ldots, B_n$, i.e. $A = A' \wedge B_1 \wedge \ldots \wedge B_n$. Thus

   $$A \vdash (\lambda x.M)N : \tau$$

   (b) N does not occur in $M[N/x]$. By hypothesis $A \vdash M : \tau$ and $B \vdash N : \sigma$. Thus

   $$A \vdash \lambda x.M : \sigma \to \tau$$

   and thus

   $$A \vdash (\lambda x.M)N : \tau$$

2. $A \vdash \lambda y.C[M[N/x]] : \tau_1 \cap \ldots \cap \tau_n \to \tau$ and $B \vdash N : \sigma$. In this case

   $$A \cup \{y : \tau_1 \cap \ldots \cap \tau_n\} \vdash C[M[N/x]] : \tau$$

   By the induction hypothesis:

   $$A \cup \{y : \tau_1 \cap \ldots \cap \tau_k\} \vdash C[(\lambda x.M)N] : \tau$$

   Thus

   $$A \vdash \lambda y.C[(\lambda x.M)N] : \tau_1 \cap \ldots \cap \tau_n \to \tau \Box$$

Lemma 7 *If M is in normal form, then there are A and σ such that $A \vdash M : \sigma$.*

Proof: By structural induction on terms in normal form.

1. M is a term variable. Trivial by the definition of $\vdash$.

2. M is of the form $\lambda x.N$. Suppose that x occurs in N. Then by the induction hypothesis there is an environment $A \cup \{x : \tau_1 \cap \ldots \cap \tau_n\} \vdash N : \sigma$. Thus $A \vdash \lambda x.N : \tau_1 \cap \ldots \cap \tau_n \to \sigma$.

3. M is of the form $xM_1 \ldots M_n$ with $M_1, \ldots, M_n$ in normal form. By the induction hypothesis there are B_i, τ_i such that $B_i \vdash M_i : \tau_i$ for $(1 \leq i \leq n)$. Let

   $$A \equiv \{x : \tau_1 \cap \ldots \cap \tau_n \to \sigma\} \wedge B_1 \wedge \ldots \wedge B_n$$

   Then $A \vdash xM_1 \ldots M_n : \sigma$. $\Box$

Theorem 5 *If M is strongly normalizable, then there are A and σ such that $A \vdash M : \sigma$.*

Proof: If M is strongly normalizable then its normal form can be reached using an inside-out reduction strategy. Using this strategy, when a redex $(\lambda x.M)N$ is contracted, N is in normal form. The proof follows by induction on the inside-out reduction path, using lemma 6 and lemma 7. $\Box$

Theorem 6 *If M is such that there is an environment A and a type σ such that $A \vdash M : \sigma$, then M is typable in the Coppo-Dezani type system ([CD80])*

Proof: Easy by induction in the derivation tree for $A \vdash M : \sigma$. $\square$

Theorem 7 *A λ-term M is strongly normalizable if and only if M is typable in the intersection type system presented.*

Proof: By theorems 5 and 6. $\square$