A typed λ-calculus with two arrows

Patrick THÉVENON *

May 31, 2007

Abstract

We present a simple typed lambda calculus similar to the one in [CePfe1], with two kind of arrows, namely linear and intuitionistic, and two kind of variables and thus of abstractions, but only one application. As the unique application carries a kind of non determinism, most of the typable terms do not have a unique principal type. We show that in some fragments of the calculus (the linear terms and the η-long terms) we can define a notion of principal type, which is unique, where negative intuitionistic arrows can be replaced by linear ones. Moreover, for any typed η-long term one keeps a valid type by applying a type substitution and replacing negative intuitionistic arrows by linear ones.

Keywords: linear and intuitionistic lambda calculus, principal type

Introduction

We would like to explain in this introduction what gave rise to the λ-calculus we will introduce, and how the problem studied and the given solution are justified.
Within the abstract categorial grammars (ACG) introduced by P. de Groote (cf. [DeG]), it is possible to make translations between two signatures, for example from a syntactic structure to a semantic structure. Each signature is based on the same structure, which allows a homogeneous framework. Basically developed on the linear lambda calculus, useful in linguistics, the system is limited in its expressivity in particular for semantic, where one wishes to be able to use several times the same variable.
The idea was then to introduce some intuitionism in the linear lambda calculus and thus to make a lambda-calculus with two kinds of variables and two kinds of arrows (intuitionistic and linear).
Finding the principal type of terms is needed in the ACGs. Indeed, the use of the ACGs is the following: two signatures, the abstract signature and the object signature, are defined (cf. [DeG] for more details and formal

*LAMA, Université de Savoie, FRANCE - e-mail : patrick.thevenon@univ-savoie.fr

definitions). This means that one gives the constants of each signature, with their types.
Then the lexicon $\mathcal{L}$ is defined. In order to give all informations, one has to give the image of the constants of the abstract signature into the terms in the object signature, and the image of the abstract atomic types into the object types.
It must then be checked that the given lexicon satisfies the commutation property, i.e. that for all term t of type T, $\mathcal{L}(t)$ has type $\mathcal{L}(T)$. In order to do that, one has to find a principal type of $\mathcal{L}(t)$, and check that this type is more general than $\mathcal{L}(T)$.
Because of the undetermined application, we can't have a unique principal type for most of the terms in this calculus. Then we must introduce some unspecified arrows ($-?$), used by the typing algorithm, in order to get a unique principal type. The unspecified arrows can then be indifferently replaced either by intuitionistic arrows ($\rightarrow$) or by linear arrows ($\multimap$).
In order to keep a light lambda calculus, one would like to avoid the unspecified arrows, and to have a notion of unique principal type without these arrows. In most cases it is impossible, but in the linear case and the η-long case, we can define a notion of principal type without unspecified arrows.
If the linear case is just an anecdotic example, the η-long terms exist in the ACGs, when one does syntax analysis. The problem is the following: having a given objet term u, what are the abstract terms t of type S (where S is a special type in the abstract signature) such that $\mathcal{L}(t) = u$?
In a more general case, given Γ an abstract context, v an η-long and β-normal object term, and an abstract type α, find an abstract term t such that $\Gamma \vdash t : \alpha$ and $\mathcal{L}(t) =_{\beta\eta} u$.
To solve this problem is to solve a matching problem (that is unification of equation where the right term has no variable). This problem is NP-complete, and a semi-algorithm for λ-calculus given by G. Huet (cf. **[Hu]**) uses η-long forms of the terms. That is why the object term considered is η-long.

Overview

The calculus will be defined in section 1. It can be seen as mixing the linear lambda calculus and the intuitionistic (standard) one. Each judgement contains a context in two parts, the linear variables and the intuitionistic variables. The usual notion of a principal type will also be given.
In general the typable terms do not have a (unique) principal type because of the non determinism of the application. Thus in section 2 we will give a different notion of principal type. In this notion, negative intuitionistic arrows can be replaced by linear ones. We will express the fact that this principal type is unique for linear and η-long terms, which is the main result

of the paper. This will be proved in the following sections, as its proof needs the introduction of more objects.
The typing algorithm will be described in section 3. It is mainly the Damas-Milner algorithm, adapted to the existence of two arrows and of the unique application that implies the use of unspecified arrows. The initial typing system must then be extended for unspecified arrows. The notion of principal type in the system obtained is the usual one, except that the unspecified arrows belong to the domain of the substitutions.
In section 4 we introduce the SNIP property. For typed terms having this property, it is possible to replace all the unspecified arrows, which are negative, by intuitionistic ones. In fact the SNIP property induces the notion of principal type given in the section 2.
In section 5 we will prove the result for linear terms, where all variables, even intuitionistic, appear exactly once in the terms. The SNIP property needs to be generalized into another one which is valid for β-reduced terms and is stable by expansion.
In section 6, we will define a more general property of the SNIP property for the η-long terms, which is stable by a unification process during the typing. The main tool for that section is a function that justifies each atom and arrow in a principal type, and is well defined for η-long terms.

1 The type system

Definition 1.1 (Types) *Let $\mathcal{A}$ be a set of atomic (constant) types. The types are defined by the following grammar:*

$$\beta ::= a \mid \alpha \mid \beta \multimap \beta \mid \beta \to \beta$$

where a belongs to $\mathcal{A}$ and α is a type variable. Both a and α are called atoms. The $\multimap$ arrow is the linear arrow and $\to$ is the intuitionisticic arrow.

Note 1.2 We will use the notation $\rightsquigarrow$ to represent any arrow.

Definition 1.3 (Terms) *The terms are defined by the following grammar:*

$$t ::= c \mid x \mid \parr x.t \mid \lambda x.t \mid (t\,t)$$

where c belongs to C, a set of constants and x belongs either to V_i, a set of intuitionistic variables, or to V_l, a set of linear variables. The symbol $\parr$ is the abstraction for linear variables and λ is the abstraction for intuitionistic variables.

The sets V_i and V_l can be considered disjoint, but in practice we will use the same notations x, y, z, ... for all variables. The kind of the variables used will always be clear from the context.

We impose restrictions on linear rules: one can build $\lambda^{\circ} x.t$ only when x is a linear free variable of t, that is appears exactly once in t. The term $(t_1\, t_2)$ is built only when the free linear variables of t_1 and t_2 are disjoint.

Note 1.4 We write $(t_1\, t_2\, t_3\, t_4)$ for the term $(((t_1\, t_2)\, t_3)\, t_4)$ and $\lambda xy.t$ for the term $\lambda x.\lambda y.t$. We write $\lambda^{\star}$ to represent any abstraction, λ or λ°.

Definition 1.5 (Typing rules) *Let τ be an application that associates to each constant its type.*

1. *A typing judgement has the shape $[\Gamma\ ;\ \Delta] \vdash t : \beta$, where t is a term and β the type of t, Γ and Δ are two disjoint sets of variables declarations $x : \tau$. The set Γ is the set of intuitionistic variables Δ is the set of linear variables.*

2. *The typing rules are the following:*

$$\frac{}{[\Gamma\ ;\ ] \vdash c : \tau(c)}$$

$$\frac{}{[\Gamma\ ;\ x : \gamma] \vdash x : \gamma} \qquad \frac{}{[\Gamma, x : \gamma\ ;\ ] \vdash x : \gamma}$$

$$\frac{[\Gamma\ ;\ \Delta, x : \alpha] \vdash t : \beta}{[\Gamma\ ;\ \Delta] \vdash \lambda^{\circ} x.t : \alpha \multimap \beta} \qquad \frac{[\Gamma, x : \alpha\ ;\ \Delta] \vdash t : \beta}{[\Gamma\ ;\ \Delta] \vdash \lambda x.t : \alpha \to \beta}$$

$$\frac{[\Gamma\ ;\ \Delta_1] \vdash t : \alpha \multimap \beta \quad [\Gamma\ ;\ \Delta_2] \vdash u : \alpha}{[\Gamma\ ;\ \Delta_1, \Delta_2] \vdash (t\, u) : \beta}\ (*)$$

$$\frac{[\Gamma\ ;\ \Delta] \vdash t : \alpha \to \beta \quad [\Gamma\ ;\ ] \vdash u : \alpha}{[\Gamma\ ;\ \Delta] \vdash (t\, u) : \beta}$$

   $(*)$ $Dom(\Delta_1) \cap Dom(\Delta_2) = \emptyset$

3. *We say that a judgement is valid if it can be derived from the typing rules. We say that a term is typable if there is a valid judgement of this term, that is of the shape $[\Gamma\ ;\ \Delta] \vdash t : \beta$.*

Sometimes the judgements will be written $x : \tau \vdash t : \theta$. This means that only one variable is considered, even if there are many others, and the fact that it is linear or intuitionistic has no interest.

Let us give some explanations on these rules. The rules on the left column are the linear rules, the ones on the right column are the intuitionistic rules. For variable introductions, the instuitionistic context is allways any set, but in order to keep the fact that linear variables must appear only once, the

linear context is empty when introducing an intuitionistic variable, and only the introduced variable belongs to the linear context when introducing a linear variable.
For arrow introductions, there is no particular condition.
For arrow elimination the intuitionistic context is indifferent, but the same in each premiss (which explains that there can be any intuitionistic context for variable introductions). For the linear rule, the conditions on linear context allow to keep a unique occurrence of each linear variable in the terms. For the intuitionistic rule, note that the linear context for the right premiss is empty. The reason is that if t is $\lambda x.t'$ the term $(t\,u)$ is then a redex. Reducing it to $t[x := u]$, one must be sure that no free linear variable of u is duplicated. As x can appear any times, u must not contain free linear variables.

Remark 1.6 A similar calculus has been introduced by I. Cervesato and F. Pfenning (cf. [CePfe1]). The main difference is that in [CePfe1] there are two applications and here there is no distinction between linear and intuitionistic application. The reason is that for a user of such a calculus, it could be tedious in some cases to choose between both applications. This difference leads to the problem of principal typing, that needs the introduction of unspecified arrows, as we will see next.

There exists a notion of β-reduction similar to the one in standard lambda-calculus. Results of strong normalisation come by a straightforward mapping of the calculus into standard lambda calculus.

The following lemma proves that linearity is preserved by the typing rules:

Lemma 1.7 *The terms built from the typing rules verify the restrictions given on linear terms. More precisely, let* $[\Gamma\ ;\ \Delta] \vdash t : \beta$ *be a judgement. Then*

- *if* $t = \lambda\!\!\!^{\circ} x.t'$ *then* x *appears freely exactly once in* t'
- *if* Δ *contains a variable* x *then* x *appears freely exactly once in* t
- *if* $t = (t_1\,t_2)$ *then the free linear variables of* t_1 *and* t_2 *are distincts*

Proof: By induction on the size of the tree. □

Definition 1.8 (Substitution) *We call substitution an application* S *from type variables into types. We define then* $S(T)$ *(also denoted* ST*) The application of* S *to a type* T*. The definition is done by induction on the type complexity:*

- $S(a) = a$;
- $S(\alpha) = S(\alpha)$ *defined by* S;

- $S(T_1 \rightsquigarrow T_2) = S(T_1) \rightsquigarrow S(T_2)$.

Note that T_1 and T_2 are equal (up to renaming) if and only if there are two substitutions S_1 and S_2 such that $S_1T_1 = T_2$ and $S_2T_2 = T_1$.

Definition 1.9 *Applying a substitution S to a judgement is applying S to each of the types of the judgement. Applying a substitution S to a tree is applying S to each of the judgements of the tree.*

Definition 1.10 (Usual principal type) *Let t be a term.*

- *A principal typing tree for t is a tree $\mathcal{T}$ such that for every typing tree $\mathcal{T}'$ for t there is a substitution S such that $S\mathcal{T}=\mathcal{T}'$.*
- *The judgement (or indifferently the type) given to t at the root of a principal typing tree for t is called a principal type of t.*

It is obvious that the principal type of a term t is unique (we will sometimes write it $\mathrm{PT}(t)$).

In general, typable terms restricted to this type system don't have a (unique) usual principal type, because the application does not make distinction between linear and intuitionistic application. We could at best define a set of principal types.
For example, let us consider the term $t := \lambda xy.(x\,y)$. This term has for types $(\alpha \to \beta) \to \alpha \to \beta$ and $(\alpha \multimap \beta) \to \alpha \to \beta$. None is an instance of the other, and any type of t is an instance of one of these two types.
There are at least two fragments in which it is possible to define a different notion of principal type, which in this case becomes unique.

2 Fragments with a unique principal type

We first need to introduce a notion of polarity in types. It is a common notion, but our purpose is just to fix notations.

Definition 2.1 (Head of a type - Polarity) *Let T be a type.*

1. *The head of T is defined by induction:*
   - *if $T = a$ (resp. α) then the head of T is a (resp. α);*
   - *if $T = A \rightsquigarrow B$ then the head of T is $\rightsquigarrow$.*
2. *Let T' be a subtype of T. The polarity of T' in T is, by induction on T:*
   - *if T=T' then T' is positive in T;*

- *if $T = T_1 \rightsquigarrow T_2$ and T' is a subtype of T_1 (resp. T_2) then T' is negative in T if T' is positive in T_1 (resp. negative in T_2) otherwise T' is positive in T.*

3. *Each atom or arrow of T appears as the head of a (unique) subtype T' of T. We define the polarity of this atom or arrow in T as the polarity of T' in T.*

4. *Consider a judgement $x : \tau \vdash t : \theta$. A positive (resp. negative) subtype in the judgement is a negative (resp. positive) subtype in τ or a positive (resp. negative) subtype in θ.*

   Similarly for atoms and arrows in the judgement.

Definition 2.2 (Linear terms) *A term t is linear if all variables, free or bounded, linear or intuitionistic, are used exactly once. Moreover, the type of the constants must satisfy the fact that the intuitionistic arrows are all positive.*

The last condition on the type of the constants will be explaned later, but can also be clear from the result on principal type we give in this section.

Definition 2.3 (η-long terms) *Let t be a term.*

1. *Let $\mathcal{T}$ be a typing tree for t. $\mathcal{T}$ is η-long if its root is in one of the following cases:*

   - *$[\Gamma\ ;\ \Delta] \vdash (x\, t_1 \ldots t_n) : \alpha$ where $n \geq 0$ and α is an atom (i.e. a type variable or an atomic type), and the typing tree of each t_i is η-long;*
   - *$[\Gamma\ ;\ \Delta] \vdash (c\, t_1 \ldots t_n) : a$ where $n \geq 0$ and the typing tree of each t_i is η-long;*
   - *$[\Gamma\ ;\ \Delta] \vdash \lambda x.t' : \alpha \to \beta$ where the typing tree of t' is η-long;*
   - *$[\Gamma\ ;\ \Delta] \vdash \lambda^{\circ} x.t' : \alpha \multimap \beta$ where the typing tree of t' is η-long;*
   - *$[\Gamma\ ;\ \Delta] \vdash (\lambda^{*} x_1 \ldots x_m.t'\, t_1 \ldots t_n) : \alpha$ where α is an atom, t' has not the shape $\lambda^{*} x.u$ and the typing trees of $\lambda^{*} x_1 \ldots x_m.t'$ and t_k are η-long.*

2. *We say that t is η-long if it has an η-long typing tree.*

We can now set out the main theorem of the paper:

Theorem 2.4 *Let t be a typable term. Assume t is either linear or η-long. Then there exists a unique judgement J of t such that any judgement of t is obtained by using the two following rules:*

- *type variables of J can be substituted by types;*
- *negative intuitionistic arrows in J can be replaced by linear arrows.*

That is, if J' is a judgement for t, there is a judgement J_o which is J with some negative intuitionistic arrows replaced by linear arrows and a substitution S such that $J' = SJ_o$.

Proof: it will be given as a corollary of theorem 4.2. □

This judgement is called the principal type of t. In the example given at the end of the previous section, the principal type of t is $(\alpha \to \beta) \to \alpha \to \beta$.

3 The typing algorithm

We will mainly use the typing algorithm of Damas and Milner. But there are changes to do due to the presence of two arrows. While typing an application $(t_1\, t_2)$, if t_2 has no free linear variable it is possible for t_1 to have either the type $\multimap$ or $\to$. Remember that the application is the same in our system for the linear and the intuitionistic case. Consequently in the typing algorithm it will be necessary to introduce unspecified arrows. Some of them will be identified during the process, and other will remain unspecified.

3.1 Unspecified arrows

An unspecified arrow is an arrow variable written $-?_n$ (or $-?$ when only one is considered) that is added to the grammar for the types. It can be substituted by other arrows, specified or not.
All the definitions and properties involving types in the previous sections can be easily adapted for the unspecified arrows. In particular, we now consider that substitutions have for domain both type variables and unspecified arrows.
We add to the initial type system the following typing scheme:

$$\frac{[\Gamma\ ;\ \Delta] \vdash t : \alpha -?_n \beta \quad [\Gamma\ ;\ ] \vdash u : \alpha}{[\Gamma\ ;\ \Delta] \vdash (t\, u) : \beta}$$

and obtain an extended type system. We will prove in the next section that the notion of usual principal type is valid in the extended type system. Obviously, if a term is typable in the initial system then it is typable in the extended one.

Lemma 3.1 *Let T be a type and S be a substitution. Let T' be a subtype of T. Then the polarity of ST' in ST is the same as the polarity of T' in T.*

Proof: By induction on T. □

Lemma 3.2 *Let t be a term typed in the extended system, in a typing tree $\mathcal{T}$.*

- *Let S be a substitution. Then $S\mathcal{T}$ is a typing tree for t.*
- *Let S be a substitution of domain the unspecified arrows of $\mathcal{T}$ and of image $\{\rightarrow, \multimap\}$. Then $S\mathcal{T}$ is a typing tree in the initial system.*

Proof: By induction on the size of $\mathcal{T}$. □

3.2 Unification

Given two types T_1 and T_2, we wonder if they are unifiable, that is if there is a substitution S such that $ST_1 = ST_2$. We can see this problem as first order unification. Indeed we can see a type as a term written with only one fonction *arrow* taking three arguments and constants. For example, the type

$$(a \rightarrow \alpha)-?_1(b \multimap \beta)-?_2 c$$

can be seen as

$$arrow(x_1, arrow(int, a, \alpha), arrow(x_2, arrow(lin, b, \beta), c))$$

Thus the notion of most general unifier (mgu) exists as in every problem of first order unification. We sometime write $U(T_1, T_2)$ the mgu of T_1 and T_2.

3.3 Algorithm 1

Here is a typing algorithm for terms which takes a term for which we know all the free variables, in particular their kind (linear or intuitionistic), and returns a typing tree if the term is typable.

Input: a term t for which linear and intuitionistic free variables are given.
Output: a typing tree $\mathcal{T}$ for t if it terminates without error.

Algorithm: By induction on the complexity of t.

- If t=x, let α be a type variable then $\mathcal{T}$ is

$$\frac{}{[\,;\, x : \alpha] \vdash x : \alpha} \text{ if } x \text{ is linear,}$$

$$\frac{}{[x : \alpha\,;\,] \vdash x : \alpha} \text{ if } x \text{ is intuitionistic.}$$

- If t=c then t has no free variable and $\mathcal{T}$ is

$$\frac{}{[\,;\,] \vdash c : \tau(c)}$$

- If $t = \lambda x.t'$ let $\mathcal{T}'$ be the typing tree of t' if it exists. Let $\mathcal{T}''$ be $\mathcal{T}'$ if x is free in t' and be $\mathcal{T}'$ where $x : \alpha$ is added to the intuitionistic side of each judgement of $\mathcal{T}$ (α being a fresh type variable) otherwise. If the root of $\mathcal{T}''$ is $[\Gamma, x : \alpha \,;\, \Delta] \vdash t' : \beta$ then $\mathcal{T}$ is

$$\frac{\mathcal{T}''}{[\Gamma \,;\, \Delta] \vdash \lambda x.t' : \alpha \to \beta}$$

- If $t = \lambda\!\!\!/ x.t'$ let $\mathcal{T}'$ be the typing tree of t' if it exists. As the variable x is free in t' (it is linear), the root of $\mathcal{T}'$ as the shape $[\Gamma \,;\, \Delta, x : \alpha] \vdash t' : \beta$ and $\mathcal{T}$ is

$$\frac{\mathcal{T}'}{[\Gamma \,;\, \Delta] \vdash \lambda\!\!\!/ x.t' : \alpha \multimap \beta}$$

- If $t = (t_1\, t_2)$ let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the typing trees of t_1 and t_2. We assume that the type variables are distinct in $\mathcal{T}_1$ and $\mathcal{T}_2$. Let us write $[\Gamma_1, \Gamma_1' \,;\, \Sigma_1] \vdash t_1 : \tau$ the judgement for t_1 and $[\Gamma_2, \Gamma_2' \,;\, \Sigma_2] \vdash t_2 : \alpha$ the judgement for t_2 where Γ_1 and Γ_2 are the set of variables in common of t_1 and t_2 and Γ_1' and Γ_2' are the set of variables not appearing in the other term.

  Let S be the most general unifier of the types of the common free variables of t_1 and t_2. Then $S\Gamma_1 = S\Gamma_2$. Let $\mathcal{T}_1'$ be the tree $\mathcal{T}_1$ in which the missing variables Γ_2' of t_2 have been added and let $\mathcal{T}_2'$ be the tree $\mathcal{T}_2$ in which the missing variables Γ_1' of t_1 have been added. Let β be a fresh type variable. We unify then $S\tau$ and $S\alpha \text{–}?_1 \beta$ where $\text{–}?_1$ is $\multimap$ if t_2 contains free linear variables (i.e. if Δ_2 is not empty) and is a fresh unspecified arrow otherwise. We so get (if it exists) S' the most general unifier and $\mathcal{T}$ is

$$\frac{S'S\mathcal{T}_1' \qquad\qquad S'S\mathcal{T}_2'}{S'S([\Gamma_1, \Gamma_1', \Gamma_2' \,;\, \Sigma_1, \Sigma_2] \vdash (t_1\, t_2) : \beta)}$$

3.4 Algorithm 2

There is also another algorithm which is equivalent to the first one, for which the terms are taken as (we consider always the greatest number of applications):

- $(x\, t_1 \ldots t_n)$ $n \geq 0$;
- $(c\, t_1 \ldots t_n)$ $n \geq 0$;
- $\lambda x.u$;
- $\lambda\!\!\!/ x.u$;

- $(u\, t_1 \ldots t_n)$ with u an abstraction, $n \geq 1$.

This algorithm is the same as the previous one for the abstractions. For the application of a term t (which can be a variable, a constant or an abstraction) to n arguments $t_1, \ldots, t_n$, the algorithm is a bit more complex. However there are stil the similar successive steps:

- By induction the term and its arguments are typed (for a variable a new type variable is given);
- The types of the common free variables are unified;
- The type of t is obtained by unifying its type with the type constructed from the ones of the t_i. That is, let β be a new type variable, let T_i be the types of the t_i, then the type constructed is $T_1 \rightsquigarrow_1 \ldots T_n \rightsquigarrow_n \beta$ where the $\rightsquigarrow_i$ are $\multimap$ if t_i has linear free variables, and a fresh unspecified arrow $-?_n$ otherwise.

Remark 3.3 We have to stress the fact that these algorithms are not the most efficient ones and are only designed to be useful tools for the proofs we will make later.

3.5 Properties

The difficult part of this paper being the properties of the principal type and not the typing algorithm, we will not prove the following propositions. More details can be read in [The].

Proposition 3.4

- *The algorithm always terminates. When successful it returns a typing tree, otherwise it stops during a unification.*
- *Let t be a typable term. Then the algorithm applied to t gives a typing tree (in the extended system) for t and the tree is principal.*

4 The SNIP property

Our purpose in the following sections is to give some fragments in which it is possible to avoid unspecified arrows for principal types. In these fragments, the following property will be satisfied by the principal type given by the algorithm:

Definition 4.1 (SNIP property) *Let T be a type.*

- *We say that T has the SNIP property if T is such that the unspecified arrows are all distinct and negative and the intuitionistic arrows are positive.*

- *We say that T has the SPIN property if T is such that the unspecified arrows are all distinct and positive and the intuitionistic arrows are negative.*
- *Let $x : \tau \vdash t : \theta$ be a typing judgement. The judgement has the SNIP property if the types of the variables have the SPIN property and if θ has the SNIP property.*
- *We say that a term t has the SNIP property if its principal type has the SNIP property.*

SNIP stands for unSpecified Negative, Intuitionistic Positive. For SPIN it is then obvious.
We have the following theorem:

Theorem 4.2 *Let t be a typable term. Assume t is either linear or η-long. Then t has the SNIP property.*

Proof: For the linear case it is proposition 5.10 and for the η-long case it is proposition 6.41. □

Proof of theorem 2.4: We know from 4.2 that t has the SNIP property. Then the arrows $-?$ are all distinct and negative and the arrows $\to$ are positive. As the unspecified arrows can be replaced by any arrow, we can replace all of them by $\to$ and thus obtain a type PT_i. In PT_i we know that any negative intuitionistic arrow (which before was unspecified) can be replaced by a linear arrow, keeping a valid type for the term (the unspecified arrows are all distinct).
Let T be a valid type for t in the initial system. It is also a valid type in the extended system, thus there is a substitution S such that $T = S\,PT$. The substitution S can be split into two substitutions, one S_t with domain type variables, and another S_a with domain unspecified arrow. It is easy to see that the type $S_a\,PT$ has no unspecified arrows, and is in fact PT_i where some negative intuitionistic arrows have been replaced by linear ones. Finally, $T = S_t(S_a\,PT)$, where S_t is a substitution for the initial system. □

Thus, we only need now to establish the proof of theorem 4.2.

Here follows an example showing why one must consider some fragments in order to obtain a notion of unique principal type.

Example 4.3 Let $t = \lambda g \lambda f \wp x \lambda u.(g\,(f\,x)\,(f\,\lambda t.(t\,u)))$.
Its principal type is

$$(b \multimap b -?_1 n) \to (((a -?_2^- e) \to^+ e) \multimap b) \to ((a -?_2^+ e) \to^- e) \multimap a \to n$$

The + and − symbols are just notations for the following. This type has not the SNIP property because none of the conditions is fufilled:

1. There is a $\rightarrow$ arrow which has a negative occurrence (the one with exponent $-$). The reason is that f has for argument $\lambda t.(t\,u)$ which has type $((a{-?_2}e) \rightarrow e)$. As x appears also as argument of f, x must have the same type. Thus $((a{-?_2}e) \rightarrow e)$ appears twice, with two different polarities.

2. The $-?_2$ arrow has two occurrences, with a positive occurrence (the one with exponent $+$) for the same reason.

Let us now argue on a possible replacement of the unspecified arrows:

- Let us assume that we replace $-?_2$ by an intuitionistic arrow. The fact that there are intuitionistic arrows in both polarities makes impossible the existence of a criterion (or a simple one) on the type allowing to decide to change some intuitionistic arrows into linear ones, keeping a valid type for t. Indeed if in the principal type an arrow is intuitionistic, it can't be replaced by a linear one.

- The same reasoning is valid if we want to replace $-?_2$ by a linear arrow, because there are linear arrow in both polarities, which is the case for most of the terms containing linear variables.

- Another important fact is that $-?_2$ has two occurrences in the type. This implies that in order to keep a valid type, one has to change both in the same way, which makes an hypothetical criterion even more complex than the one we propose.

Note that t is not η-long because x, whose type is an arrow, has no argument and t is not linear because of f.

5 Linear terms

The first fragment we study is the linear fragment. We already gave the definition (2.2) of linear terms. Recall in particular that the constants type must satisfy that the intuitionistic arrows are positive. Otherwise any constant with a negative intuitionistic arrow in its type would contradict the SNIP property.
The typing algorithm in the case of linear terms is easier because it becomes useless to unify the type variables, as no variable can be common to two terms.
We want to prove that a linear term has the SNIP property. In fact we will give a more general result, which can be easily proved for the standard linear λ-calculus. We will not give all the proofs, as most of them are easy.

We need a specific lemma for terms with constants.

Lemma 5.1 *Let C be a type with no type variable. Let T be a type unifiable with C, containing a type variable α which has exactly one occurrence in T. Let S be the unifier of C and T.*

- *if α is positive (resp. negative) in T and C has the SNIP property then $S\alpha$ has no type variable and has the SNIP property (resp. the SPIN property).*
- *If α is negative (resp. positive) in T and C has the SPIN property then $S\alpha$ has no type variable and has the SNIP property (resp. the SPIN property).*

Proof: By induction on the complexity of C, making case analysis on of shape of T. □

We first study the case of normal terms (β-reduced).

Lemma 5.2 *Let t be a normal term, linear and typable. The following properties hold for t:*

- *Each type variable appearing in the principal type appears twice, with a positive occurrence and a negative occurrence;*
- *The term t has the SNIP property.*

Proof: By induction on the complexity of t. The proof is easy and is just a case analysis on the shape of t. We use the algorithm 2 in order to type the terms. We use the previous lemma for the case $t = (c\, t_1 \ldots t_n)$. □

Definition 5.3 *Let S be a substitution. We say that S is SL if S changes unspecified arrows only into $\multimap$, and does not change any type variable.*

Lemma 5.4 *Let A and B be two unifiable types. Let S be SL. Assume that SA and SB are unifiable too. Then there is a SL substitution S' such that $U(SA,SB) \circ S = S' \circ U(A,B)$.*

Proof: By induction on the sum of the number of unification steps of A and B and of SA and SB, making case analysis on A and B. □

Lemma 5.5 *Let u_1, u_2 and v be three typable terms. Assume that no free variable of u_1 and u_2 is in v. Assume there is a SL substitution S such that $PT(u_1) =_S PT(u_2)$. We have the following properties:*

- *If $(u_1\, v)$ and $(u_2\, v)$ are typable then there is a SL substitution S' such that $PT((u_1\, v)) =_{S'} PT((u_2\, v))$.*

- *If* $(v\,u_1)$ *and* $(v\,u_2)$ *are typable then there is a SL substitution* S' *such that* $PT((v\,u_1))=S'\ PT((v\,u_2))$.

Proof: Straightforward, using lemma 5.4 with, in the first case, $A :=\mathrm{PT}(u_2)$ and $B :=\mathrm{PT}(v)\multimap ?_1\alpha$. We use the algorithm 1 in order to type the terms. □

The following lemma gives us a way to prove the equality of the principal types of two terms.

Lemma 5.6 *Let* t *and* t' *be two terms having a principal type. We assume that if* t *has type* T *then* t' *has type* T *and vice versa. Then* t *and* t' *have the same principal type.*

Proof: Straightforward. □

Lemma 5.7 *Let* w_1, w_2 *and* v *be terms.*

1. *Assume that* $t = (\lambda x.(w_1\,w_2)\,v)$ *is typable and that* x *appears exactly once in* $(w_1\,w_2)$.
   (a) *If* x *is in* w_1 *then* $PT(t)=PT((\lambda x.w_1\,v\,w_2))$
   (b) *If* x *is in* w_2 *then* $PT(t)=PT((w_1\,(\lambda x.w_2\,v)))$
2. *Assume that* $t = (\lambda\!\!\!^{\circ} x.(w_1\,w_2)\,v)$ *is typable and that* x *appears exacly once in* $(w_1\,w_2)$ *then*
   (a) *if* x *is in* w_1 *then* $PT(t)=PT((\lambda\!\!\!^{\circ} x.w_1\,v\,w_2))$
   (b) *if* x *is in* w_2 *then* $PT(t)=S\ PT((w_1\,(\lambda\!\!\!^{\circ} x.w_2\,v)))$ *where* S *is SL.*

Proof: For the first part and the first point of the second part, we use lemma 5.6. Let us prove the last point. Consider the typing tree of t in this case:

$$\dfrac{\dfrac{\dfrac{[\,;\,]\vdash w_1 : \beta\multimap\gamma \quad [\,;x:\alpha]\vdash w_2:\beta}{[\,;x:\alpha]\vdash (w_1\,w_2):\gamma}}{[\,;\,]\vdash \lambda\!\!\!^{\circ}x.(w_1\,w_2):\alpha\multimap\gamma} \qquad [\,;\,]\vdash v:\alpha}{[\,;\,]\vdash(\lambda\!\!\!^{\circ}x.(w_1\,w_2)\,v):\gamma}$$

Consider the typing tree of $(w_1\,(\lambda\!\!\!^{\circ}x.w_2\,v))$:

$$\dfrac{[\,;\,]\vdash w_1:\beta\rightsquigarrow_1\gamma \qquad \dfrac{\dfrac{[\,;x:\alpha]\vdash w_2:\beta}{[\,;\,]\vdash\lambda\!\!\!^{\circ}x.w_2:\alpha\multimap\beta} \qquad [\,;\,]\vdash v:\alpha}{[\,;\,]\vdash(\lambda\!\!\!^{\circ}x.w_2\,v):\beta}}{[\,;\,]\vdash(w_1\,(\lambda\!\!\!^{\circ}x.w_2\,v)):\gamma}$$

Notice that the type of w_1 can be different in the two cases, because x can be the only free linear variable in w_2, thus $\rightsquigarrow_1$ can be different from $\multimap$.

- If the first tree is a principal typing tree for $(\lambda x.(w_1\, w_2)\, v)$,

  then $(w_1\, (\lambda x.w_2\, v)$ can be typed with the same type because its typing tree is still valid if we change $\rightsquigarrow_1$ into $\multimap$. Thus there is S_1 such that

  $S_1\mathrm{TP}((w_1\, (\lambda x.w_2\, v)))=\mathrm{TP}((\lambda x.(w_1\, w_2)\, v))$

  with S_1 not necessarily SL.

- If the second tree is a principal typing tree for $(w_1\, (\lambda x.w_2\, v))$, let us consider the cases for $\rightsquigarrow_1$:

  - $\rightsquigarrow_1=\multimap$: the term $(\lambda x.(w_1\, w_2)\, v)$ can be typed of the same type and there is S_2 such that
    $\mathrm{TP}((w_1\, (\lambda x.w_2\, v)))=S_2\mathrm{TP}((\lambda x.(w_1\, w_2)\, v))$. Thus both types are equal and we can take the identity for S.
  - $\rightsquigarrow_1=\rightarrow$: it is impossible. Indeed we know that the term is typable with an arrow $\multimap$ for w_1, thus there is a substitution that changes the arrow of the type of w_2 into $\multimap$ by definition of the principal type. But one can not change an intuitionistic arrow into a linear arrow.
  - $\rightsquigarrow_1= -?_1$ an unspecified arrow. Let S be the substitution that changes $-?_1$ into $\multimap$.

    If $-?_1$ does not belong the the principal type, as we can apply S to the tree and as S keeps the type invariant, there is S_2 such that $\mathrm{TP}((w_1\, (\lambda x.w_2\, v)))=S_2\mathrm{TP}((\lambda x.(w_1\, w_2)\, v))$. The two terms have thus the same principal type and we can take the identity for S.

    Otherwise there is S_2 such that
    $S\mathrm{TP}((w_1\, (\lambda x.w_2\, v)))= S_2\mathrm{TP}((\lambda x.(w_1\, w_2)\, v))$.
    Necessarily the substitution S_1 seen above contains S in this case because the typing tree of t uses $\multimap$. Thus if we define S_1' as the substitution equal to S_1 that keeps $-?_1$ invariant,
    $S_1'\ S\mathrm{TP}((w_1\, (\lambda x.w_2\, v)))= \mathrm{TP}((\lambda x.(w_1\, w_2)\, v))$.
    Thus $S\mathrm{TP}((w_1\, (\lambda x.w_2\, v)))=\mathrm{TP}((\lambda x.(w_1\, w_2)\, v))$.

Thus in all cases: $S\mathrm{TP}((w_1\, (\lambda x.w_2\, v)))=\mathrm{TP}((\lambda x.(w_1\, w_2)\, v))$ with S a SL substitution. □

The following lemma shows that the property holds by β-expansion.

Lemma 5.8 *Let t be a linear term. Let t' be such that $t \rightarrow_\beta t'$. Then there is a SL substitution S such that PT(t)=S PT(t').*

Proof: By induction on the complexity of t, looking at the β-reduction made on t, using lemma 5.5 and lemma 5.7, for the most difficult cases of an application. □

Remark 5.9

1. Notice we really used in this lemma the fact that the terms are linear, because $\mathrm{PT}((\lambda x.(w_1\, w_2)\, v))$ is not equal to $\mathrm{PT}(((\lambda x.w_1\, v)\, (\lambda x.w_2\, v)))$, what we should prove in order to get the result in a non linear case. One can check for example that for $w_1 = \lambda z.(z\, (x\, \xi\, \xi'))$, $w_2 = \lambda z.(x\, \zeta\, z)$ and $v = \lambda w.w$ both types are distinct. Moreover if $(\lambda x.(w_1\, w_2)\, v)$ is typable then $((\lambda x.w_1\, v)\, (\lambda x.w_2\, v))$ is typable but the converse is false. See for example $w_1 = \lambda z.(z\, (x\, \xi))$, $w_2 = \lambda z.(x\, \zeta\, z)$ and $v = \lambda w.w$.

2. One can also wonder if the lemma is true for affine terms (whose variables appear at most once). But the answer is negative: if one considers $t = \lambda z.(\lambda x.y\, (z\, u))$, with u any (closed) normal term, one can see that after a β-reduction the bounded variable z does not appear free anymore and its type becomes an atom in the principal type.

We can now give the promissed proposition:

Proposition 5.10 *Let t be a typable linear term. Then its principal type has the following properties:*

- *Each type variable appearing appears twice with a positive and a negative occurrence.*
- *The type has the SNIP property.*

Proof: We use the fact that these properties hold for normal terms, and that they also hold by β-expansion thanks to previous lemmas. □

Remark 5.11 These results are true only for the principal type of the linear terms. The fact that all type variables appear twice, in different polarities, implies that if we change a type variable into a type containing a $\rightarrow$ arrow, this arrow will then have a positive and a negative occurrence.

6 η-long terms

Before going into details and giving all the definitions we need, let us explain the intuition behind the fact that η-long terms have the SNIP property. This comes from the fact that the unspecified arrows appear in the typing algorithm during application rules only. So they correspond to elimination rules, and so have negative polarities.
Nevertheless, in order that none of the unspecified arrow appear positively, it is necessary that each positive arrow is explicitly introduced by an abstraction, which is the case for η-long terms.
If the intuitive argument is quite easy, the proof of the property is not so easy, and even looses this intuition. The difficult part is in fact to prove

that during the unifications used by the typing algorithm, all the properties are kept. And the most restrictive one is the uniqueness of the unspecified arrows, and of some type variables.
The greatest obstacle came from the subterms of the shape $\lambda x.t$ where x is not free in t. In this case, x can have any type and t still be η-long. So the type of x must not be changed into any type, it must satisfy the properties. Thus we had to introduce a way to follow the unification process which allow to check that the properties are kept.

6.1 Syntactic properties of η-long terms

We already gave a definition (2.3) of the η-long terms.

Lemma 6.1 *Let t be an η-long term. If $t = (\not\lambda x_1 \dots x_m.t'\, t_1 \dots t_n)$ and t' has not the shape $\not\lambda x.u$ then $m = n$.*

Proof: Straightforward. □

Remark 6.2 This lemma above implies that an η-long term has necessarily one of the following shape:

- $t = (x\, t_1 \dots t_n)$;
- $t = (c\, t_1 \dots t_n)$;
- $t = \lambda x.u$;
- $t = \not\lambda x.u$;
- $t = (\not\lambda x_1 \dots x_n.t'\, t_1 \dots t_n)$ where t' has not the shape $\not\lambda x.u$. We will sometime write $t = (\not\lambda x_1 \dots x_n.(\dots)\, t_1 \dots t_n)$ such a term.

The following lemma proves that our notion is the same as the usual notion of η-long terms.

Lemma 6.3 *Let t be an η-long term. Let t' be a subterm of t having an arrow type. Then t' is applied or has the shape $\not\lambda x.u$.*

Proof: By induction on t. □

Lemma 6.4 *Let $\mathcal{T}$ be a typing tree of a term t, let S be a substitution. If $S\mathcal{T}$ is η-long then $\mathcal{T}$ is η-long.*

Proof: By induction on t. □

Algorithm 3

The typing algorithm 2 is slightly modified in order to take into account the fact that the terms are η-long:

- The shape of the terms is always supposed to be in one of the cases described above.
- The algorithm always checks if the type obtained is η-long. So after each recursive call, all the subterms have an η-long type.

This new algorithm is called algorithm 3.

We now have to introduce a function whose aim is to justify each element, arrow or atom, of the principal type of a term. Essentially this function takes as arguments an address in a type and a term, and outputs a set of subterms of the term whose type is the one pointed by the address.
It is important to notice that if a type is principal, each of its elements has a reason to be. In the precise case of η-long terms, it is possible to associate some subterms to each atom and arrow in the principal type.
This is possible because each subterm that has an arrow type is either an abstraction or is applied to arguments. Thus in the second case it is always possible to take the arguments.
In the case of arbitrary terms, such a task would be difficult, because of the lack of arguments for some subterms having an arrow type. This subterms are indeed of the same type as other subterms that can be difficult to detect.
The function that we will define is mainly used with η-long terms and addresses in their principal type, even if the definition is more general.
We first have to precise our notion of address.

6.2 Addresses

Definition 6.5

1. *An address is a finite list, possibly empty, of elements of* $\{0, 1\}$. *We will write* $[]$ *the empty address,* $\epsilon :: c$ *the list* c *with* ϵ *added as first element and* $c :: d$ *the concatenation of two addresses. In order to simplify, we will write* 1^k *for a list containing* k *times* 1.

2. *We define the application* f *which to an address* c *and a type* T *associates the subtype of* T *met at the address* c *by induction:*

   - $f([], T) = T$
   - $f(0 :: c, T_1 \rightsquigarrow T_2) = f(c, T_1)$
   - $f(1 :: c, T_1 \rightsquigarrow T_2) = f(c, T_2)$
   - *In the other cases,* f *is not defined*

3. *We say that an address c is an address of a type T if $f(c,T)$ is defined. In this case we define $h(c,T)$ which denotes the head of the type $f(c,T)$ (which is either an arrow or an atom).*

4. *We say that c is positive if c has an even number of 0 and that c is negative otherwise.*

Lemma 6.6 *Let T be a type, T' a subtype of T and c an address in T.*

- *There is an address c' such that $f(c',T) = T'$*
- *c has the same polarity as $f(c,T)$ in T.*
- *Let S be a substitution. Then c is an address in ST.*

Proof: By induction on the complexity of the type T. □

Lemma 6.7 *Let c be an address in the type T and in the type T'. Let S be a substitution. If $h(c,T) = h(c,T')$ then $h(c,ST) = h(c,ST')$.*

Proof: By induction on the size of c. □

6.3 Justifing terms

We assume that bounded variables are also subterms, even if they do not appear freely in a sub-term. This imply that they must have a fixed name. As we do not perform reductions from now, this does not lead to problems of α-conversion.
In practice, when we will take a set E of terms all the terms of E will be some subterms of a unique term t, whose variables (bounded or not) are all distinct.
Thus we assume for example that if two terms in E have (bounded) variables with the same name then these variables are the same.

Definition 6.8 *The notation $\not\lambda\overline{x}.E$, for a set E of terms, a set $\overline{x}$ of variables and $\not\lambda$ a given (fixed) abstraction, will stand for a set of terms $\not\lambda x.t$ for t in E and x in $\overline{x}$.*

Note 6.9 The notation given in definition 6.8 is ambiguous, as it does not precise which are the terms of $\not\lambda\overline{x}.E$. In fact it will be mainly used to match the shape of a set of terms. There is a case where the notation is not ambiguous: when the set $\overline{x}$ contains only one variable x, $\not\lambda\overline{x}.E$ stands for the set of $\not\lambda x.t$ for t in E. We will write it $\not\lambda x.E$. This kind of set is the only one that we will build from a set E of terms.

Definition 6.10 *Let us define φ the application defined on the pairs (address,set of terms) by induction on the size of the addresses:*

- $\varphi([], E) = (E, [], E)$
- $\varphi(1 :: c, \lambda\!\!\!^{*}\overline{x}.E) = \varphi(c, E)$
- $\varphi([0], \lambda\!\!\!^{*}\overline{x}.E) = (\overline{x}, [0], \lambda\!\!\!^{*}\overline{x}.E)$
- $\varphi(0 :: 1^k :: 0 :: c, \lambda\!\!\!^{*}\overline{x}.E) = \varphi(c, F)$ $(k \geq 0)$ *(⋆)*
- $\varphi(0 :: 1^k, \lambda\!\!\!^{*}\overline{x}.E) = (G, 0 :: 1^k, \lambda\!\!\!^{*}\overline{x}.E)$ $(k \geq 1)$ *(⋆⋆)*
- φ *is not defined in the other cases.*

(⋆) where $F = \{t_{k+1}\,;\, (x\, t_1 \ldots t_{k+1})$ *is a subterm of a term* $\lambda\!\!\!^{*}x.t$ *of* $\lambda\!\!\!^{*}\overline{x}.E\ \}$. *If* F *is empty then* φ *is not defined.*
(⋆⋆) where $G = \{(x\, t_1 \ldots t_k)\,;\, (x\, t_1 \ldots t_k)$ *is a subterm of a term* $\lambda\!\!\!^{*}x.t$ *of* $\lambda\!\!\!^{*}\overline{x}.E\ \}$. *If* G *is empty then* φ *is not defined.*

Remark 6.11

1. In the definition 6.10 the cases $c = [0]$ and $c = 0 :: 1^k$ for $k \geq 1$ are separated. The reason is that in the second case we are interested in the occurrences of x but not in the first one. Indeed, if $c = [0]$, what interests us is, for terms of the shape $\lambda\!\!\!^{*}x.t$, what justifies the type on the left of the arrow. It is obvious that it is x that justifies it, even if x has no occurrence. In the other case, we are interested in the type of one of the arguments of x, and thus we need the occurrences of x.
2. Notice that if one only wants to know if φ is defined for a precise input, the name of variables have no real importance. The most important fact is that the input must have the good shape, and that the sets of subterms taken by φ must not be empty.

Definition 6.12

1. *φ gives back a triple.*
   - *The first object is a set called the set of justifying subterms.*
   - *The second object is called the address of the last call.*
   - *The elements of the last object are called the terms of the last call.*
   - *The pair formed by the address of the last call and the terms of the last call is called the last call to φ.*
2. *There are three cases in the definition of φ that are terminal, and the address of the last call distinguish them:*
   - *If it is $[]$, we say that φ gives back η-long terms.*
   - *If it is $[0]$, we say that it gives back variables.*

- *If it has the shape* $0 :: 1^k$ *with* $k \geq 1$*, we say that it gives back terms of the shape* $(x\, t_1 \ldots t_k)$.

Remark 6.13

1. Remember that we said that in practice, when we will take a set E of terms all the terms of E will be some subterms of a unique term t, whose variables (bounded or not) are all distinct. Thus the set of the justifing terms is well determined, in the sense that there is no ambiguity thanks to the uniqueness of all variables. Subterms shared by terms of E are the same sub-term in t.

2. The words "last call" come from the fact that φ is a recursive function, and so the last call corresponds to the natural notion of last recursive call. It is nethertheless important to notice that the recursion is not on the a simple shape of the address: the cases are more complex than $[]$, $0 :: c$ and $1 :: c$. This implies that sometimes φ is not defined. But it is obvious that after each recursive call the size of the address decreases.

Definition 6.14 *Let* $\tilde{\varphi}$ *be the function defined on triples (address, variable, set of terms) by* $\tilde{\varphi}(c, x, E) = \varphi(0 :: c, \not\lambda x.E)$ *where* $\not\lambda$ *depends on the variable* x *(that must be of the same kind for all terms of* E*).*

Example 6.15 *Let* t *be* $\lambda x.(f\,(x\, t_1\, \lambda y.t_3)\,(x\, t_2\, \lambda z.t_4))$ *with the* t_i *be any closed terms and* c *be* $[0, 1, 0, 0]$.
Then we have $\varphi(c, \{t\}) = \varphi([0], \{\lambda y.t_3, \lambda z.t_4\}) = (\{y, z\}, [0], \{\lambda y.t_3, \lambda z.t_4\})$.
Notice that the type of t *has the shape* $T = (\alpha \to (\beta \to \gamma) \to \delta) \to \epsilon$.
We have $f(c, T) = \beta$ *and* y *and* z *do have the type* β.

We will need the results of the following lemma. Some similar results for $\tilde{\varphi}$ can be proven in the same way. They justify the vocabulary used in the definition. Most of the time we will use these results implicitly.

Lemma 6.16 *Let* E *and* F *be set of terms. Let* c *be an address.*

- *If* $\varphi(c, E)$ *is defined let us write* (c^*, E^*) *its last call. Then* $\varphi(c, E) = \varphi(c^*, E^*)$.

- *If* $\varphi(c, E)$ *is defined then the terms of the last call are* η*-long. In particular if the address of the last call is* $[]$ *then the justifying subterms are* η*-long.*

- *If* $\varphi(c, E)$ *and* $\varphi(c, F)$ *are defined then the address of the last call for* E *and* F *are the same.*

- *If $\varphi(c, E)$ and $\varphi(c, F)$ are defined then $\varphi(c, E \cup F)$ is defined. Moreover, the set of the justifying subterms is the union of the two justifying sets of $\varphi(c, E)$ and $\varphi(c, F)$, and it is the same thing for the set of the subterms of the last call.*

Proof: By induction on the size of the address. □

Remark 6.17 The last part of lemma 6.16 above justifies that if we have a term t of the shape $(x\, t_1 \ldots t_n)$, and if c is an address in the type of a variable y, then the set of the justifying subterms of $\tilde{\varphi}(c, y, \{t\})$ contains the set of the justifying subterms of each of the $\tilde{\varphi}(c, y, \{t_i\})$. In order to see that, one just have to look at the definition of $\tilde{\varphi}$, and of φ, in order to check that we are in the case of the last part of lemma 6.16.

The following two lemmas justify the name 'justifying subterms'. When we speak about the type of a justifying subterm, we speak about the type given to it in the typing tree of the term it comes from. Here we suppose that we are in the case where all the terms of a set E of terms are subterms of a unique term t, whose variables are all distinct, in order to have well determined subterms. But no particular condition is needed on the types of the terms of E appart from the one in the lemmas.

Lemma 6.18 *Let E be a set of typed terms. Let c be an address such that $\varphi(c, E)$ is defined. If there is an atom (resp. an arrow) such that, for all term t of type T of E, $h(c, T)$ is this atom (resp. arrow), then the justifying subterms have a type with this atom (resp. arrow) as head.*

Proof: By induction on the size of the address. □

Lemma 6.19 *Let E be a set of terms having the same type T. Let c be an address such that $\varphi(c, E)$ is defined. Then the justifying subterms have all the same type, that is $f(c, T)$.*

Proof: By induction on the size of the address. □

Lemma 6.20 *Let E be a set of terms. Let c and d be addresses. If $\varphi(c, E)$ is defined let (c', E') be its last call. Then $\varphi(c :: d, E) = \varphi(c' :: d, E')$.*

Proof: By induction on the size of the address. □

Remark 6.21 Lemma 6.20, which generalizes lemma 6.16, explains the role of the last call and will be used in the following way: In the case where $\varphi(c, F)$ gives back variables, the last call has by definition the shape $([0], \lambda\!\!\!/\overline{x}.E)$. If we want to know if some of the variables are applied, and thus appear freely in sub-terms of E, we just have to check that $\varphi(c :: 1, F)$ is defined, because $\varphi(c :: 1, F) = \varphi([0, 1], \lambda\!\!\!/\overline{x}.E)$ by lemma 6.20, and it is defined only if the variables are applied by definition of φ.

6.4 Classes

We need tools allowing us to follow some properties during the typing of a term t. The difficult part is the typing of an application, using two steps of unification. During these steps, we have many judgements of terms, and we have to unify the types of the common variables. In order to do that, one has to get the part of the types that do not match, and unify them.
In order to point out these parts of type, we will define the classes that, obviously, will use the notion of address defined before. The classes intuitively group together all the terms matching at the same address.
At the end of the unification, the classes group together all the terms, as the type of the variables are all the same.
As the proof of proposition 6.41, talking about a property of the types of η-long terms, is made by induction, we need that a similar property holds during the unification process.

Note 6.22

1. In a set J of judgements we will always write t_i $(1 \leq i \leq n)$ the terms. We will write $\tau_i(x)$ (resp. $\tau(t_i)$) the type of the variable x in the term t_i (resp. of the term t_i).

2. We do not suppose in the following that a set of judgements is a set of valid judgements. We will indeed sometime write judgements for which the variables or the term do not have the right type with respect to the other elements of the judgement.

Definition 6.23 (Point) *Let J be a set of judgements.*

1.
   - *$P = (c, x, i)$ is a (variable) point of J if x is free in t_i and c in an address in $\tau_i(x)$. We say that P is a variable point.*
   - *$P = (c, i)$ is a (term) point of J if c is an address in $\tau(t_i)$. We say that P is a term point.*
   - *A point of J is either a variable or a term point of J.*

2. *let P be a point of J. We define $H_J(P)$ (or $H(P)$ if there is no confusion), type head of P, in the following way:*
   - *$H(P) = h(c, \tau(t_i))$ if $P = (c, i)$;*
   - *$H(P) = h(c, \tau_i(x))$ if $P = (c, x, i)$.*

Definition 6.24 (Class) *Let J be a set of judgements.*

1. *On the points of J we define a relation $\simeq$ which is reflexive (in particular $(c,i) \simeq (c,i)$) and such that $(c,x,i) \simeq (d,y,j)$ if $c = d$, $x = y$ and $H(c,x,i) = H(c,x,j)$. This relation is clearly an equivalence relation.*

   A class of J is an equivalence class for this relation. We will write (c,i) instead of $\{(c,i)\}$ and (c,x,I) instead of $\{(c,x,i) \mid i \in I\}$ in order to have simple notations.

   We will write $Cl_J(P)$, or simply $Cl(P)$ if there is no confusion, the class of a point P.

2. *We define the type head of a class C, that we write $H_J(C)$ (or $H(C)$ if there is no confusion), as $H(P)$ for any point P of C.*

3. *Let C be a class of J.*

   - *if C has the shape (c,i) we say that it is a class of a term;*
   - *if C has the shape (c,x,I) we say that it is a class of x (or more generaly of a variable).*

   In both cases we say that c is the address of C.

Remark 6.25 Let J be a set of judgements. Then for all atom (resp. arrow) appearing in types, there is a class whose type head is this atom (resp. arrow). Indeed we can associate an address, a term (plus potentially a variable) that points to this atom (resp. arrow) by lemma 6.6, and so the point and then the class exists. Thus each atom or arrow can be seen as a $\mathrm{H}(C)$ for a class C.

Definition 6.26 (Associated set of a class) *Let J be a set of judgements and C be a class of J. The associated set of C, noted $E_J(C)$ (or $E(C)$ if there is no confusion) is*

- $E(C) = \{t_i\}$ *if* $C = (c,i)$;
- $E(C) = \{t_i \mid i \in I\}$ *if* $C = (c,x,I)$.

Definition 6.27 (Justifying set of a class) *Let J be a set of judgements. Let C be a class of J of address c.*

- *if C is a class of a term, $\Phi(C) = \varphi(c, E(C))$.*
- *if C is a class of a variable x, $\Phi(C) = \tilde{\varphi}(c, x, E(C))$.*
- *A class C is justified if $\Phi(C)$ is defined.*

Example 6.28 Let us consider $t_1 = \lambda y.(x\, y)$ and $t_2 = \lambda w.\lambda z.(x\, (w\, z))$. Let J be the set of judgements containing the principal typing of t_1 and t_2:

$$x : a \multimap ?b \vdash t_1 : a \to b$$

and

$$x : a \multimap b \vdash t_2 : (c \multimap a) \to c \multimap b$$

We gave intentionally some type variables shared by both terms. The variable x can be either linear or intuitionistic, let us consider it intuitionistic in the following.
then:

- $P_1 = ([], x, 1)$, $P_2 = ([0], x, 2)$ and $P_3 = ([0, 1], 2)$ are points of J;
- $C_1 = ([], x, \{1\})$, $C_2 = ([0], x, \{1, 2\})$ and $C_3 = ([0, 1], \{2\})$ are their respective classes.

Let us see it precisely:

- P_1 is a point such that $\mathrm{H}(P_1) = \multimap ?$. As $\mathrm{H}([], x, 2) = \multimap \neq \mathrm{H}(P_1)$, we obtain $\mathrm{Cl}(P_1) = C_1$. Then $\Phi(C_1) = (\{x\}, [0], \lambda x.t_1\})$.
- P_2 is a point such that $\mathrm{H}(P_2) = a$.

  As $\mathrm{H}([], x, 1) = a = \mathrm{H}(P_2)$ we obtain $\mathrm{Cl}(P_2) = C_2$.

  Thus $\Phi(C_2) = (\{(w\, z), y\}, [], \{(w\, z), y\})$.
- P_3 is a point such that $\mathrm{H}(P_3) = a$.

  We know that $\mathrm{Cl}(P_3) = ([0, 1], \{2\}) = C_3$ by definition and then $\Phi(C_3) = (\{(w\, z)\}, [0, 1], \{t_2\})$.

Definition 6.29 (Singularity) *Let J be a set of judgements. Let a be an atom (resp. $\rightsquigarrow$ an arrow) appearing in J. We say that a (resp. $\rightsquigarrow$) is singular in J if there exists a unique class C of J such that $H(C) = a$ (resp. $\rightsquigarrow$).*

Definition 6.30 (Polarity) *Let J be a set of judgements. Let C be a class of J of address c.*

- *if $C = (c, i)$, $H(C)$ is positive if c is positive, negative otherwise;*
- *if $C = (c, x, I)$, $H(C)$ is positive if c is negative, positive otherwise.*

Remark 6.31 In the case where the set of judgements contains only one term, the singularity means that the atom (or arrow) has a unique occurrence, and the polarity is equivalent to the one defined above.

Definition 6.32 *Let J and J' be two sets of judgements of the same terms t_i. We say that a class C of J is included in a class C' of J' if both have the same address, are classes of the same variable and if $E_J(C)$ is included in $E_{J'}(C')$.*

Lemma 6.33 *Let J be a set of judgements, let S be a substitution and P be a point of J. Then P is a point of SJ and the class of P in J is included in the class of P in SJ.*

Proof: It is straightforward following the definitions and using lemmas 6.6 and 6.7. □

6.5 Preliminary lemmas

We first need some definitions and lemmas.

Definition 6.34 (Class property) *Let J be a set of judgements. We say that J has the class property if the following points are satisfied:*

- *There is an η-long term t such that (where the terms t_i are the terms of J):*
  - *either $t_i = t$ for each i;*
  - *or $t = (x\, t_1 \ldots t_n)$;*
  - *or $t = (c\, t_1 \ldots t_n)$;*
  - *or $t = (t_1\, t_2 \ldots t_n)$ with $t_1 = \lambda^\star x_1 \ldots x_n.t'$ and t' is not an abstraction.*

  We say that J is associated to the η-long term t;
- *Each class C is justified;*
- *If the last call of $\Phi(C)$ has the shape $([0], \{\lambda^\star x.u\})$ with $x \notin u$ then $H(C)$ is an atom a and a is singular;*
- *If $H(C)$ is an unspecified arrow then it is singular and it is negative;*
- *If $H(C) = \rightarrow$ then it is positive.*

Remark 6.35

1. The first point of the class property implies, as t is η-long, that the t_i are also η-long. It is also important to notice that the first point does not depend on the types given in J.
2. The class property is a generalization of the SNIP property, which consists in the two last points when J is a singleton.

3. The condition on the classes such that the last call has the shape $([0], \{\not\lambda x.u\})$ may seem enigmatic. It is nevertheless very important.

   Indeed, these atoms, and only them, are susceptible to be transformed, during the unification steps, into arrow types (we always keep the fact that terms are η-long because the variables don't appear in this case). The condition of singularity ensures that applying the substitution does not break the property on the arrows. Indeed, it implies that this atom does not appear with a different polarity, which then would invert the polarity of all the arrows, and even multiply them.

   In the unification process, apart from these particular atoms, the unification always changes atoms into atoms, and arrows into arrow, because the terms are η-long, and the type of the appearing variables have always mainly the same shape.

Lemma 6.36 *Let J be a set of judgements having the class property. Assume there is a point $P_1 = (c, x, i)$ such that $f(c, \tau_i(x))$ is a type variable α and a point $P_2 = (c, x, j)$ such that $f(c, \tau_j(x))$ is an arrow type T. Let S be the substitution that changes α into T. Then SJ has the class property.*

Proof: Let us define $C_1 = \mathrm{Cl}_J(P_1)$ and $C_2 = \mathrm{Cl}_J(P_2)$, which are justified by assumption.

- Let us first prove that $\mathrm{H}(C_1) = \alpha$ is singular in J.

  In the η-long term t associated to J, the justifying subterms for C_1 and for C_2 have all the same type thanks to lemma 6.16, remark 6.17 and lemma 6.19.

  The address of the last call are the same for both classes by lemma 6.16, thus only three cases may eventually occur for the justifying subterms

  - They are η-long terms. This means that for C_2 they have the shape $\not\lambda x.u$, but not for C_1. Moreover for C_1 these terms are not applied. But in t they all have the same type and stay η-long. It is impossible.
  - They are terms of the shape $(x\, t_1 \ldots t_k)$. This means that for C_2 these terms are applied, but not for C_1. But in t they all have the same type, and as t is η-long it is impossible.
  - So they are variables.

    Thus for C_2 some of these variables are applied, because they have an arrow type. One only needs to make the address longer by adding 1 for example, in order to get a point $P_2' = (c :: [1], x, j)$ whose class will be justified by assumption and so gives back some $(y\, v_1)$. Indeed, as the terms are variables, the last call is by definition of the shape $([0], \not\lambda y.E)$. Thanks to lemma 6.16,

$\Phi(C_2) = \varphi([0], \lambda\!\!\!^{*}y.E)$. Then thanks to lemma 6.20, $\Phi(\mathrm{Cl}_J(P_2')) = \varphi([0,1], \lambda\!\!\!^{*}y.E)$. Finaly by definition, we can see that the set of justifying subterms contains some $(y\, v_1)$.

But in C_1 no variable appear. Indeed otherwise they would not be applied because they have a type variable as type, and as in t these variables all have the same type, it is necessarily an arrow type, because of C_2 (the variables being applied), but then there would be non applied variables of arrow type, which is possible for η-long terms only if they do not appear.

Thus by the class property we obtain that α is singular in J. Indeed, we have just shown that we have a class, C_1, such that the last call of $\Phi(C_1)$ has the shape $([0], \{\lambda\!\!\!^{*}x.u\})$ with $x \notin u$. Thus by assumption $\mathrm{H}(C_1) = \alpha$ is an atom, what we already know, and is singular.

- Let us prove now that for all class C_s of SJ there s a point P_a of C_s such that P_a is also a point of J and if $\mathrm{H}_{SJ}(C_s)$ is an arrow then $\mathrm{H}_J(P_a)$ is the same arrow.

  Let P be any point of C_s. If it is a point of J then it is obvious. It is the case for all the classes (e, k) because α is singular and so the type of the terms is not changed by S.

  Otherwise let us write (e, y, k) the point P. Let $\dot{e}$ be the longest sub-address of e such that for all strict sub-address e' of $\dot{e}$ (e', x, j) is a point of J (in the worst case $\dot{e} = []$). The address $\dot{e}$ is different from e. Let us consider $h(\dot{e}, \tau(t_k))$, there are three cases

  - it is an arrow. Impossible because otherwise we could take $\dot{e}$ longer because the arrow is not changed by S.
  - it is a type variable different from α. As it is not changed by S, we would have $e = \dot{e}$ because it is impossible to make this address longer into SJ, which is impossible.
  - thus it is the type variable α. Necessarily $\dot{e}$ is c and the class of the point $(\dot{e}, y, k)$ is C_1 by singularity of α and thus $y = x$. Thus $e = c :: c'$, c' being an address in the type T, because α is changed into T. The triple (e, x, j) is a point of J because $P_2 = (c, x, j)$ is a point such that $f(c, \tau_j)$ is T by assumption. Thus we can take $P_a = (e, x, j)$. This point is a point of SJ, which is in the class of P, that is C_s by definition of the class.

- Let us finally prove that SJ has the class property.

  - The first point is still true because it does not depend on the types given in J as we have already remarked.

- All the classes are justified thanks to what we have just proven. Indeed let C_s be a class in SJ. Then there is a point P_a in C_s such that P_a is also a point in J. By assumption the class of P_a in J is justified, thus also in SJ (the justification does not depend on the type).
- If C_s is such that $\Phi(C_s)$ has the shape $([0], \{\maltese x.u\})$ with $x \notin u$ then it is the same thing for the class C of P_a in J. Thus by assumption $\mathrm{H}_J(C)$ is an atom b and b is singular in J. The substitution changes it into $\mathrm{H}_{SJ}(C_s)$.

  If $\mathrm{H}_{SJ}(C_s)$ is not an atom, this means that $b = \alpha$ and thus P_2 is in C_s by singularity of α. But $\mathrm{H}(P_2)$ is an arrow by assumption, and thus one can't have $x \notin u$ for all the terms of the last call (to see that, it is enough to take as above the class of the point $(c :: 1, x, t_j)$ which is justified in order to see that some variables are applied and in particular appear).

  Thus we know that $\mathrm{H}_{SJ}(C_s)$ is the atom b (as it is different from α it is not changed by S). Assume that it is not singular in SJ. Thus there is a class C'_s different from C_s such that $\mathrm{H}_{SJ}(C_s)$=$\mathrm{H}_{SJ}(C'_s)$=b. But there is P'_a a point of C'_s which is also a point of J. As the classes of P_a and P'_a in SJ are different (they are C_s and C'_s),they are also different in J.

  But as b is singular in J, $\mathrm{H}_J(\mathrm{Cl}(P_a)) = b$ is different from $\mathrm{H}_J(\mathrm{Cl}(P'_a))$, which must be changed into b by S by assumption on C'_s. But nothing is changed into b, so this is impossible.

  Thus b is singular in SJ.
- If $\mathrm{H}(C_s)$ is an unspecified arrow then it is the same thing for $\mathrm{H}(P_a)$ (P_a still being the point which existence has been proved before) and this arrow is singular and negative in J. It is also a negative arrow in SJ because it is the same address. And it is also singular in SJ by a similar argument to the previous paragraph if we suppose that it is not singular.
- If $\mathrm{H}(C_s)$ is an intuitionistic arrow, then it is the same thing for $\mathrm{H}(P_a)$ and this arrow is positive in J, thus also in SJ. □

Here is the same lemma but in the case of the substitution of an arrow.

Lemma 6.37 *Let J be a set of judgements satisfying the class property. Assume there is a point $P_1 = (c, x, i)$ such that $H(P_1)$ is an unspecified arrow $-?_1$ and a point $P_2 = (c, x, j)$ such that $H(P_2)$ is another arrow $\rightsquigarrow_2$ (either unspecified or linear). Let S be the substitution that changes $-?_1$ into $\rightsquigarrow_2$. Then SJ has the class property.*

Proof:

- It is obvious by assumption that $-?_1$ is singular.

  Let us consider the classes C_1=$\mathrm{Cl}_J(P_1)$ and C_2=$\mathrm{Cl}_J(P_2)$, which are justified.

- Let us prove that for all class C_s of SJ there is a point P_a of C_s such that P_a is also a point of J and if $\mathrm{H}_{SJ}(C_s)$ is a arrow then $\mathrm{H}_J(P_a)$ is the same arrow.

  Let P be any point of C_s. It is necessarily a point of J, because the addresses have not changed. If $\mathrm{H}_{SJ}(C_s)$ is an arrow, assume that $\mathrm{H}_J(P)$ is not the same.

  Necessarily $S(\mathrm{H}_J(P))$=$\mathrm{H}_{SJ}(C_s)$, thus $\mathrm{H}_J(P)$=$-?_1$ and $\mathrm{H}_{SJ}(C_s)$=$\rightsquigarrow_2$, which implies that P is in C_1 by singularity of $-?_1$. Let us then take $P_a = P_2$. We have $\mathrm{H}(P_2)$=$\rightsquigarrow_2$ and P_2 is in C_s in this case.

- Let us finally prove that SJ has the class property.

  - The first point is still valid.
  - All the classes are justified by the same argument as the previous lemma.
  - If C_s is such that $\Phi(C_s)$ has the shape $([0], \{\not\forall x.u\})$ with $x \notin u$ then it is the same thing for the class C of P_a in J. Thus by assumption $\mathrm{H}_J(C)$ is an atom b and b is singular, and it is not changed by the substitution.

    It is singular by the same argument as the previous lemma.
  - If $\mathrm{H}_{SJ}(C_s)$ is an unspecified arrow $-?$ then it is the same thing for $\mathrm{H}_J(P_a)$ and this arrow is singular and negative in J. It is also a negative arrow in SJ because it is the same address.

    Assume it is not singular in SJ. Then there is a class C'_s different from C_s such that $\mathrm{H}_{SJ}(C_s)$=$\mathrm{H}_{SJ}(C'_s)$=$-?$. But there is P'_a a point of C'_s that is also a point of J, and such that $\mathrm{H}_J(P'_a) = -?$. As the classes of P_a and P'_a in SJ are different (they are C_s and C'_s), they are different in J. This contradict the singularity of $\mathrm{H}_J(P_a)$.
  - If $\mathrm{H}(C_s)$ is an intuitionistic arrow then it is the same for $\mathrm{H}(P_a)$ and this arrow is positive in J, thus also in SJ. □

Here is again the same lemma, in the case of the substitution of a type variable into another one.

Lemma 6.38 *Let J be a set of judgements which has the class property. Assume there is a point $P_1 = (c, x, i)$ such that $H(P_1)$ is a type variable α and a point $P_2 = (c, x, j)$ such that $H(P_2)$ is another type variable β. Let S be the substitution that changes α into β. Then SJ has the class property.*

Proof:
Consider the classes C_1=$\mathrm{Cl}_J(P_1)$ and C_2=$\mathrm{Cl}_J(P_2)$ which are justified.

- Notice that neither α nor β are necessarily singulars in J.
- Let us prove that for all class C_s of SJ there is a point P_a of C_s such that P_a is also a point of J and if $\mathrm{H}_{SJ}(C_s)$ is an arrow then $\mathrm{H}_J(P_a)$ is the same arrow.

  Let P be any point of C_s. It is necessarily a point of J, because the addresses did not changed. If $\mathrm{H}_{SJ}(C_s)$ is an arrow then $\mathrm{H}_J(P)$ is the same one because arrows have not changed, nor the shape of the types.
- Let us finally prove that SJ has the class property.
  - The first point is still valid.
  - All the classes are justified by the same argument as in lemma 6.36.
  - If C_s is such that $\Phi(C_s)$ has the shape $([0], \{\not\lambda x.u\})$ with $x \notin u$ then it is the same thing for the class C of P_a in J (which is included in the first one by lemma 6.33). Thus by assumption $\mathrm{H}(C)$ is an atom b and b is singular in J. It's substitution is still an atom.

    If it is the same atom then it is singular by the same argument as in lemma 6.36.

    If it is a different atom, then $b = \alpha$ and $\mathrm{H}(C_s)$=β. As α is singular in this case, necessarily C=C_1. Then C_s contains P_2 and so its class thanks to the substitution. As $\Phi(C_s)$ has the shape $([0], \{\not\lambda x.u\})$ with $x \notin u$, it is the same thing for $\Phi(C_2)$ and β is singular in J.

    Thus β is singular in SJ: the substitution changes α singular in J into β singular in J, and the class C_s contains boths classes C_1 and C_2.
  - If $\mathrm{H}(C_s)$ is an unspecified arrow then it is the same thing for $\mathrm{H}(P_a)$ and this arrow is singular and negative in J. It is also a negative arrow in SJ because it is the same address, and it is singular in SJ by the same argument as previously.
  - If $\mathrm{H}(C_s)$ is an intuitionistic arrow then it is the same thing for $\mathrm{H}(P_a)$ and this arrow is positive in J, and thus in SJ too. □

Note 6.39 When we consider a set of judgements containing only one term t, we write (c, x, t) and (c, t) the points of J. Notice that in this case, talking about points is equivalent to talking about classes. In the following we will then talk about points.

Lemma 6.40 *Let t be an η-long term, consider an η-long judgement for t. Let J be the set containing olny this judgement. Let $P = (c, x, t)$ be a point such that $H(P)$ is an atom and P is justified. Then the justifying subterms are not applied and do not have the shape $\lambda\!\!\!^{*}x.u'$.*

Proof: Straightforward. □

6.6 The Proposition

We can now give and prove the promissed proposition for η-long terms:

Proposition 6.41 *Let t be a typable term such that the algorithm 3 gives for t an η-long typing tree. Then the (set sontaining only the) judgement at the root of the tree has the class property.*

Proof: By induction on the complexity of t.

1. $t = \lambda x.t'$: by induction the judgement given after the recursive call, $\{[x : \tau, x_i : \tau_i \ ; y_j : \gamma_j] \vdash t' : \theta\}$ has the class property. The algorithm gives then $\{[x_i : \tau_i \ ; y_j : \gamma_j] \vdash \lambda x.t' : \tau \to \theta\}$. One can then check that the class property is satisfied.

2. $t = \lambda\!\!\!^{*}x.t'$: in the same way.

3. $t = (x\, t_1 \ldots t_n)$: consider J_0 the set of judgements given for the t_i. By induction each of the judgements of J_0 has the class property. We can assume that each t_i has type variables different from the others. Then J_0 has the class property.

   - The first step, the unification of the types of the variables, is made by going from set of judgements to set of judgements all expected in the lemmas 6.36, 6.37 and 6.38. After each step of unification, the property is then satisfied. Thus at the end of the unification process, we obtain a set of judgements having the class property, and the free common variables have their type unified.
   - The second step is the unification of the type of x obtained at the end of the first step with the type formed by the types of the t_i obtained and β. Consider the set of judgements containing the two judgements described below:
     - the judgement of $t = (x\, t_1 \ldots t_n)$, where all the free variables have the type obtained after the first step (if x is not free, we give a fresh type variable as type).
     - the same judgement of t but with a type for the variable x constructed from the types of the t_i obtained after the first step and using a new type variable β.

In these two judgements the type of t is β. Both judgements have the class property. Indeed the type of t being β, the only point of the shape (c, t) is $([], t)$. Then for the points of the form (c, y, t) with y different from x the justification is the same as for the set of judgements of the t_i. For the points of the shape (c, x, t)

- in the first judgement x is treated as the other variables.
- in the second judgement, the type given to x is $\tau_1 \rightsquigarrow_1 \dots \tau_n \rightsquigarrow_n \beta$. It has the SPIN property thanks to the first step. Some of the addresses are related to addresses in the types τ_i of the t_i, others are related to arrows $\rightsquigarrow_i$ or to β.

In all cases the points are justified, and the singularities are kept. The singularities come from the fact that the class of a point (c, y, t) with y being any variable is the set of all the terms t_i at the end of the first step of unification.

Thus this set has the class property. We can so apply lemmas 6.36, 6.37 and 6.38 in order to conclude that at the end we obtain twice the same judgement that has the class property. Thus the class property is satisfied.

4. $t = (c\, t_1 \dots t_n)$: the first step (unification of the types of the variables) is similar to the previous case.

   For the second step, let us consider the term t with the types of the variables obtained after the first step, t having an atom as type. The class property is satisfied. We then unify the type of c with the type constructed with the types of the t_i obtained after the first step and a new type variable. As the algorithm ends, we know that t keeps an atom as type and is η-long after the application of the unifier.

   Assume that the unification changes type variables appearing in the type of the variables into arrow types. The previous lemma shows then that the term obtained is no more η-long (because the justifying subterms having an atom as type are neither applied nor of the shape $\lambda\!\!\!^{*}x.u$), thus it is impossible.

   Thus all the points are justified. It is obvious that the unspecified arrows, as they are singular before the unification, stay unchanged and singular. Consequently all the arrows are the same and have the same polarity as before the unification. The points P for which the last call to Φ as the shape $([0], \{\lambda\!\!\!^{*}x.u\})$ with $x \notin u$ are such that $\mathrm{H}(P)$ is an atom a and a is singular before the unification. As the unification does not changes this atom, it is still true after the unification.

   Thus the class property is satisfied.

5. $t = (\lambda\!\!\!^{*}x_1 \dots x_m.(\dots)\, t_1 \dots t_m)$: it is similar to the previous case. Notice that there is $m + 1$ judgements in the set of the first step (we add

the term $\lambda^* x_1 \dots x_m.(\dots)$). □

We finish with the following corollary, which allows to use the SNIP property not only for the principal type, but for any η-long term.

Corollary 6.42 *Let t be an η-long typable term of type T. Assume there is an address c such that $h(c,T)$ is $\rightarrow$ and c is negative. Then t is (η-long) typable of type T' where T' is T where $\rightarrow$ has been replaced by $\multimap$.*

Proof: As t is η-long typable, its principal type TP is η-long. There is a substitution S such that $STP{=}T$. By the previous proposition, TP has no negative $\rightarrow$. Thus S changes an unspecified arrow $-?_1$ into $\rightarrow$, or a type variable α into a type A containing the arrow $\rightarrow$.
In the first case, as $-?_1$ has a unique occurrence, we can exchange S with S' which is S that maps $-?_1$ to $\multimap$. Then $S'TP = T'$ and T' is a valid type for t.
In the second case, let us prove that α has a unique occurrence. In TP, there is an address c for α. The point P is justified by te previous proposition and the terms of the last call are η-long. The justifying subterms have type α. The address of the last call is not [], because otherwise the justifying subterms are also η-long, but with the type T, these terms have an arrow type. Thus the justifying subterms are either $(x\, t_1 \dots t_k)$ or variables. They can't be $(x\, t_1 \dots t_k)$, because within the principal type they have type α, and so can't be applied, but with T they have an arrow type and t is η-long. Thus they are variables, the last call being $([0], \{\lambda^* x.u\})$. Necessarily the variables x do not appear in the terms u, otherwise they would not be applied, but with T they have an arrow type. The previous proposition implies then that α has a unique occurrence.
As α has a unique occurrence, we can consider S' which is S where α is changed into A', which is A where $\rightarrow$ is exchanged with $\multimap$. Thus $S'TP{=}T'$ and t has type T'. □

Future Work

Some problems are still to be explored. We could study the unification or the matching problem. This has already been studied on the similar λ-calcul of I. Cervesato (cf. [CePfe1, CePfe2]), which is however a bit easier, as there are two kinds of application, and not a single one like in the calculus introduced in this paper, which eliminates typing problems.
To add intuitionism is not the only way to extend the calculus for ACGs. Another way is to add features to atomic types. This would allow a more concise and precise description of signatures for the user. For example, for the atomic type NAME, we can add features corresponding to number (singular, plural). In cases where it is possible for a term to have any feature, it could

be possible to group cases by a quantifier, which reduces the definition of the signatures. It could then be another case of study.

References

[CePfe1] I. Cervesato, F. Pfenning, *A linear Logical Framework*, 11th Annual Symposium on Logic in Computer Science - LICS'96 (E. Clarke, editor), pp. 264-275, IEEE Computer Society Press, New Brunswick, NJ, 27-30 July 1996

[CePfe2] I. Cervesato, F. Pfenning, *Linear Higher-Order Pre-Unification*, International Workshop on Proof-Search in Type-Theoretic Languages - PSTT'96 (D. Galmiche, editor), pp. 41-50, New Brunswick, NJ, 30 July 1996

[DaMi] L. Damas, R. Milner, *Principal type-schemes for functional programs*, 9th symposium Principles of programming Languages, pp 207-212. ACM Press, 1982

[DeG] P. de Groote, *Towards abstract categorial grammars*, in Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference, pp. 148–155, 2001

[Hu] G.P. Huet, *A unification algorithm for typed λ-calculus*, in Theoretical Computer Science 1 27-57, North-Holland Publishing Company, 1975

[The] P. Thévenon, *Vers un assistant à la preuve en langue naturelle*, PhD Thesis, Université de Savoie, 2006