On the structure of normal λ-terms having a certain type

SABINE BRODA and LUIS DAMAS, *DCC & LIACC, Universidade do Porto, R. do Campo Alegre 823, 4150-180 Porto, Portugal.*
E-mail: {sbb,luis}@ncc.up.pt

Abstract

We define an alternative representation for types in the simply typed λ-calculus or equivalently for formulas in the implicational fragment of intuitionist propositional logic, which gives us a better insight on the nature of a types structure and its relation to the structure of the set of its normal inhabitants. Based on this representation of a type φ, we define the notion of a valid proof tree. Any such valid proof tree represents a finite set of normal inhabitants of φ and every normal inhabitant corresponds to exactly one valid proof tree, constructable with the primitive parts in the formula tree of φ. Precise algorithms are given to establish this relation. Finally, we give a simple characterisation of the proof trees that represent principal normal inhabitants of a type φ.

1 Introduction

Typed versions of systems of λ-calculus have been studied since their appearance in the beginning of this century in [6] and [5], due to their importance to areas of mathematical logic and more recently to computer science. One of the fields with direct application is the area of functional programming (e.g. programming languages ML, Miranda or Haskell). The concept of functional programming relies on the representation of computable functions/algorithms by λ-terms and the execution of programs corresponds to the reduction of λ-terms. Type assignment to λ-terms provides thus for partial specification of the represented algorithms and can be used to show partial correctness of programs. Besides that, types are used to improve the efficiency of the compilation of functional programs. Almost all practical systems have the system $\mathbf{T}A_\lambda$ of simply typed λ-calculus at their basis. On the other hand, there exists a direct correspondence (cf. [11]), via Curry-Howard isomorphism, between $\mathbf{TA}_\lambda$ and the implicational fragment of intuitionistic propositional logic. In fact, a type φ can by assigned to some λ-term if and only if φ is a provable formula/theorem. Furthermore, every (normal) term to which φ can be assigned, i.e. every (normal) inhabitant of φ, represents a (normal) proof of φ in the natural deduction system for the implicational fragment of intuitionistic logic.

Thus, it is natural to seek for algorithms which, for a given type φ, - decide if φ has any normal inhabitant, - decide if the number of normal inhabitants of φ is finite or infinite, - or compute the set of all normal (or even principal normal) inhabitants of φ. Decision-procedures for intuitionistic propositional logic have been known for a long time (e.g. [8], [7]) and the corresponding decision problem, as well as the infiniteness problem of the set of normal inhabitants have been shown to be polynomial-space complete, respectively by Statman in [12] and Hirokawa in [10]. An algorithm, also described in [9], to count (which is the difficult part) and list the normal inhabitants of a type φ has been given by Ben-Yelles in [2] and a similar algorithm for principal normal inhabitants was first presented in [3]. Both these algorithms are based on the generation of terms whose depth (a measure on the structure of a λ-term in β-normal form) does not exceed some limit which can be computed from φ. This fact makes both algorithms extremely inefficient. On the other hand, it seems to be very unsatisfactory that the algorithms mentioned do not take into account any information about φ, besides the number of type variables and of their occurrences. In fact, as pointed out by Hindley in [9], when we ask for the number of normal inhabitants [1] of a type, the answer is often finite and interesting patterns show up which are still not completely understood. The situation is even worse for principal inhabitants, since we do not know any "non-exhaustive" method for computing a principal normal inhabitant of a type φ.

The present paper is an attempt towards a better understanding of the nature of the structure of a type and its relation to the structure of the set of its normal inhabitants. In section 2 we define an alternative tree-like representation for a type φ, which we call its formula tree. The nodes of such formula tree consist of primitive parts which, following the rules given in section 3, can be combined to form valid proof trees. Any such a valid proof tree represents a finite set of normal inhabitants of φ and every normal inhabitant corresponds to exactly one valid proof tree, constructable with the primitive parts in the formula tree of φ. In fact, in section 4 we give precise algorithms to establish this relation. The first algorithm computes a valid proof tree for any [2] given normal inhabitant of a type φ, while the second "inverse" algorithm computes the set of all normal inhabitants which give origin to a given valid proof tree of φ. In section 5 we give a simple characterisation of the proof trees that represent principal normal inhabitants of a type φ. Finally, in section 6, we illustrate the use of our approach with another example.

2 The formula tree of a type

We assume familiarity with the basic notions in λ-calculus and use standard notation from [1] and [9]. Our notation differs from that in [1], since we denote type-variables (atoms) by "a,b,c,..."and arbitrary types by lower-case Greek letters. For type assignment we consider the system $\mathbf{TA}_\lambda$ of simply typed λ-calculus à la Curry (for an introduction see [9]) or [1], section 3.1).

In this section we define an alternative tree-like representation for a type φ, which we call its formula tree, denoted by $\mathtt{tree}(\varphi)$, and whose nodes consist of primitive parts of either one of the following forms.

(p1)

$$\begin{array}{c} | \\ a \end{array}$$

[1] Note that the number of (not necessarily normal) inhabitants of a type is always zero or infinite.

[2] More precisely, we define the algorithms for a subclass of normal inhabitants, called long normal inhabitants. But as will be argued, both algorithms can be trivially completed to apply to normal inhabitant in general.

(p2)

$$\begin{array}{ccc} & a & \\ a_1 & \cdots & a_n \end{array} \qquad (n \geq 1)$$

(p3)

$$\begin{array}{c} a \\ | \end{array}$$

DEFINITION 2.1
A tree with primitive parts as nodes is a formula tree iff

- the root of the tree is of form (p1);
- every internal node of the tree is of form (p2);
- every leaf of the tree is of form (p2) or (p3).

The following algorithm computes the formula tree $\texttt{tree}(\varphi)$ of a type φ. We will use dashed lines for the edges of the formula tree in order to distinguish them from the edges in the primitive parts (nodes) of the tree. Note that every type φ can be written uniquely in the form $\varphi = \alpha_1 \to \ldots \to \alpha_n \to a$, where a is an atom and $n \geq 0$. The algorithm is given by the following.

- If $n = 0$, i.e. $\varphi \equiv a$, then

$$\texttt{tree}(\varphi) = \begin{array}{c} | \\ a \end{array}$$

- If $n \geq 1$, then

$$\texttt{tree}(\varphi) = \begin{array}{ccc} & | & \\ & a & \\ \texttt{t}(\alpha_1) & \cdots & \texttt{t}(\alpha_n) \end{array}$$

  where

$$\texttt{t}(b) = \begin{array}{c} b \\ | \end{array}$$

  and finally for $k \geq 1$ and $m_1, \ldots, m_k \geq 0$ take
  $\texttt{t}((\beta_{11} \to \ldots \to \beta_{1m_1} \to b_1) \to \ldots \to (\beta_{k1} \to \ldots \to \beta_{km_k} \to b_k) \to b) =$

$$\begin{array}{ccccccc} & & & b & & & \\ & b_1 & & \cdots & & b_k & \\ \texttt{t}(\beta_{11}) & \cdots & \texttt{t}(\beta_{1m_1}) & \cdots & \texttt{t}(\beta_{k1}) & \cdots & \texttt{t}(\beta_{km_k}) \end{array}$$

Note that one can easily define an inverse algorithm, which given a formula tree $\mathtt{FT}$ computes the unique type φ such that $\mathtt{tree}(\varphi) = \mathtt{FT}$.

DEFINITION 2.2
The indexed counterpart $\varphi_{\mathtt{i}}$ of a type φ is obtained by successively indexing all occurrences of type variables in φ.

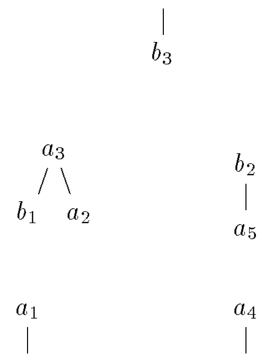
EXAMPLE 2.3
Consider the formula

$$\theta \equiv ((a \to b) \to a \to a) \to ((a \to a) \to b) \to b.$$

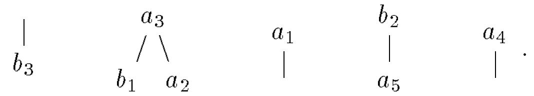
The formula tree of its indexed counterpart

$$\theta_{\mathtt{i}} \equiv ((a_1 \to b_1) \to a_2 \to a_3) \to ((a_4 \to a_5) \to b_2) \to b_3$$

is $\mathtt{tree}(\theta_{\mathtt{i}}) =$



with primitive parts



3 Proof trees

DEFINITION 3.1
Let $\varphi_{\mathtt{i}}$ be the indexed counterpart of a type φ. A proof tree built from $\mathtt{tree}(\varphi_{\mathtt{i}})$ is obtained by joining primitive parts of $\mathtt{tree}(\varphi_{\mathtt{i}})$ identifying (different) occurrences of the same type variable. We call it a valid proof tree if

- all leafs in the proof tree are of form (p3);
- whenever a primitive part p_i occurs beneath some primitive part p_j in the formula tree $\mathtt{tree}(\varphi_{\mathtt{i}})$, then above every occurrence of p_i in the proof tree there is at least one occurrence of p_j.

Note that the formula tree $\varphi_{\mathtt{i}}$ defines some kind of hierarchy over its primitive parts, which has to be respected by every valid proof tree built from $\mathtt{tree}(\varphi_{\mathtt{i}})$. In particular, this means that the root of every valid proof tree built from $\mathtt{tree}(\varphi_{\mathtt{i}})$ is the root of $\mathtt{tree}(\varphi_{\mathtt{i}})$.

Example 3.2
The following is a valid proof tree built from the primitive parts of $\mathtt{tree}(\theta_{\mathtt{i}})$ from example 2.3.

$$\begin{array}{c} | \\ b_3 = b_2 \\ | \\ a_5 = a_3 \end{array}$$

$$\begin{array}{c} b_1 = b_2 \\ | \\ a_5 = a_1 \\ | \end{array} \qquad\qquad \begin{array}{c} a_2 = a_4 \\ | \end{array}$$

with restrictions $a_1 = a_3 = a_5$, $a_2 = a_4$ and $b_1 = b_2 = b_3$. Note that we represented the identification of different occurrences of the type variables in this proof tree by equations, such as $b_3 = b_2$.

4 Proof trees and (long) normal inhabitants

In this section we establish the relation between the set of normal inhabitants of a type φ and the set of valid proof trees constructed from $\mathtt{tree}(\varphi_{\mathtt{i}})$. More precisely, we analyse the existing relationship with a special subclass of normal inhabitants of φ, called long normal inhabitants. But, as we will see, this is in fact no restriction at all. A β-normal inhabitant M of a type φ is called a long normal inhabitant of φ iff every variable-occurrence z in M is followed by the longest sequence of arguments allowed by its type, i.e. iff each component with form $(zP_1 \dots P_n)$, $(n \geq 0)$ that is not in a function position has atomic type. The (finite) set of all terms obtained by η-reducing a λ-term M is called the η-family of M and denoted by $\{M\}_\eta$. It has been shown (cf. [2], [9]) that the η-families of the long normal inhabitants of φ partition the set of normal inhabitants of φ into non-overlapping finite subsets, each η-family containing just one long member. Furthermore, Ben-Yelles (cf. [2], [9]) showed that every normal inhabitant of a type φ can be η-expanded to one unique (up to α-conversion) long normal inhabitant of φ. A simple expansion-algorithm can be found in [9].

We begin this section defining an algorithm which given a long inhabitant M of a type φ, with indexed counterpart $\varphi_{\mathtt{i}}$, computes a valid proof tree $\mathtt{PT}(M) = \mathtt{pt}(M, \varphi_{\mathtt{i}}, \emptyset)$ of $\mathtt{tree}(\varphi_{\mathtt{i}})$. For $\varphi_{\mathtt{i}} = \alpha_1 \to \dots \to \alpha_n \to a_s$ and a set of type-assignments (basis) Γ, $\mathtt{pt}(M, \varphi_{\mathtt{i}}, \Gamma)$ is defined by the following.

- For $M = \lambda x_1 \dots x_n.y$ and $y : a_t \in \Gamma \cup \{x_1 : \alpha_1, \dots, x_n : \alpha_n\}$, let

$$\mathtt{pt}(M, \varphi_{\mathtt{i}}, \Gamma) = \begin{array}{c} | \\ a_s = a_t \\ | \end{array} .$$

- For $M = \lambda x_1 \dots x_n.yM_1 \dots M_m$ with $m \geq 1$, let $\mathtt{pt}(M, \varphi_{\mathtt{i}}, \Gamma) =$

$$\begin{array}{c} | \\ a_s = a_t \\ \\ \mathtt{PT}_1 \quad \cdots \quad \mathtt{PT}_\mathtt{m} \end{array}$$

  where $\Gamma' = \Gamma \cup \{x_1 : \alpha_1, \dots, x_n : \alpha_n\}$, $y : \beta_1 \to \dots \to \beta_m \to a_t \in \Gamma'$ and for $1 \leq j \leq m$, $\mathtt{PT}_j = \mathtt{pt}(M_j, \beta_j, \Gamma')$.

PROPOSITION 4.1
If M is a long normal inhabitant of a type φ, with indexed counterpart $\varphi_{\mathtt{i}}$, then $\mathtt{PT}(M)$ is a valid proof tree built from $\mathtt{tree}(\varphi_{\mathtt{i}})$.

PROOF. By induction on the length of $\varphi_{\mathtt{i}}$, using the definition of the algorithm **tree** to construct the formula tree of $\varphi_{\mathtt{i}}$. ∎

COROLLARY 4.2
If a formula φ has a normal inhabitant M, then there is at least one valid proof tree built from $\mathtt{tree}(\varphi_{\mathtt{i}})$, where $\varphi_{\mathtt{i}}$ is the indexed counterpart of φ.

PROOF. It is sufficient to η-expand M to the unique long normal form M^+ in its η-family (using Ben-Yelles' algorithm). By proposition 4.1, $\mathtt{PT}(M^+)$ is a valid proof tree built from $\mathtt{tree}(\varphi_{\mathtt{i}})$. ∎

Now, we will define an *inverse* algorithm, which given the formula tree of the indexed counterpart of some type φ, $\mathtt{t_i} = \mathtt{tree}(\varphi_{\mathtt{i}})$, and any valid proof tree $\mathtt{PT}$ built from it, computes a set $\mathtt{Terms(PT)} = \mathtt{Terms}(\mathtt{t_i}, 0, \mathtt{PT}, \emptyset)$ of long closed normal inhabitants of φ. In the following let L represent a set assigning variable names to primitive parts of $\mathtt{t_i}$. Then, $\mathtt{Terms}(\mathtt{t_i}, k, \mathtt{PT}, L)$ is defined by the following.

Let $\begin{array}{c} | \\ a_s = a_t \end{array}$ be the root of $\mathtt{PT}$ and consider the corresponding occurrence of a_s in $\mathtt{t_i}$ of the form

$$\begin{array}{c} | \\ a_s \\ \\ p_1 \quad \cdots \quad p_n \end{array}$$

with $n \geq 0$. Now, let $L' = L \cup \{(p_i, x_i^k) \mid 1 \leq i \leq n\}$.

- If $\mathtt{PT}$ is of the form $\begin{array}{c} | \\ a_s = a_t \\ | \end{array}$, then take the corresponding primitive part $p = \begin{array}{c} a_t \\ | \end{array}$ and let

$$\mathtt{Terms}(\mathtt{t_i}, k, \mathtt{PT}, L) = \{\lambda x_1^k \ldots \lambda x_n^k . x \mid (p, x) \in L'\}.$$

- Otherwise, $\mathtt{PT}$ is of the form

$$\begin{array}{ccc} & | & \\ & a_s = a_t & \\ \mathtt{PT}_1 & \cdots & \mathtt{PT}_m \end{array}$$

with $m \geq 1$, and such that for $i = 1, \ldots, m$ the root of $\mathtt{PT}_i$ is $\begin{array}{c} | \\ b_{s_i} = b_{t_i} \end{array}$. Consider the corresponding primitive part p of the form

$$\begin{array}{ccc} & | & \\ & a_t & \\ b_{s_1} & \cdots & b_{s_m} \end{array}$$

and let $\mathtt{Terms}(\mathtt{t_i}, k, \mathtt{PT}, L) =$

$\{\lambda x_1^k \ldots \lambda x_n^k . x M_1 \ldots M_m \mid (p, x) \in L', M_j \in \mathtt{Terms}(\mathtt{t_i}, k+1, \mathtt{PT}_j, L'), 1 \leq j \leq m\}$.

PROPOSITION 4.3
Let $\mathtt{PT}$ be a valid proof tree built from the formula tree of the indexed counterpart of some type φ. Then every member of $\mathtt{Terms(PT)}$ is a closed long normal inhabitant of φ. Furthermore, the two algorithms are complementary in the sense that for every closed long normal inhabitant M of φ there is $M \in \mathtt{Terms}(\mathtt{PT}(M))$.

PROOF. In order to prove the first claim, define a basis (set of type-assignments) from L by $\Gamma_L = \{x : \varphi_p \mid (x, p) \in L\}$, where φ_p denotes the subtype of φ which corresponds to the subtree beneath the primitive part p in $\varphi_{\mathtt{i}}$. Then, prove by induction on the structure of $\mathtt{PT}$ that for every $M \in \mathtt{Terms}(\mathtt{t_i}, k, \mathtt{PT}, L)$ there is $\Gamma_L \vdash M : \tau$, where τ is the subtype of φ which can be reconstructed from the occurrence of the first variable in the root of $\mathtt{PT}$, and that M is a β-nf which is long with respect to (Γ_L, τ). Then the result follows from the fact that the algorithms starts with an empty set L. The second part of the proof is by induction on the structure of M. ∎

EXAMPLE 4.4
The algorithm $\mathtt{Terms}$ computes exactly one normal λ-term

$$M = \lambda xy.y(\lambda t.x(\lambda z.y(\lambda w.z))t)$$

corresponding to the valid proof tree in example 3.2. The η-family of M is

$$\{M, \lambda xy.y(x(\lambda z.y(\lambda w.z)))\}.$$

From corollary 4.2 and proposition 4.3 we conclude the following.

COROLLARY 4.5
A type φ is inhabitated if and only iff there is at least one valid proof tree constructed from $\mathtt{tree}(\varphi_{\mathtt{i}})$. ∎

Resuming, every normal inhabitant of a type φ corresponds to one valid proof tree built from $\mathtt{tree}(\varphi_{\mathtt{i}})$, every valid proof tree built from $\mathtt{tree}(\varphi_{\mathtt{i}})$ corresponds to a finite set of normal inhabitants of φ, and distinct valid proof trees correspond to distinct and disjoint sets of normal inhabitants.

5 Principal proof trees

DEFINITION 5.1
Let φ be a type with indexed counterpart $\varphi_{\mathtt{i}}$. A valid proof tree $\mathtt{PT}$ built from $\mathtt{tree}(\varphi_{\mathtt{i}})$ is a principal proof tree iff

- $\mathtt{PT}$ uses every primitive part of form (p2) in $\mathtt{tree}(\varphi_{\mathtt{i}})$;
- $\mathtt{PT}$ identifies all occurrences of the same type variable in $\varphi_{\mathtt{i}}$.

The following result is the correspondent, for proof trees, to Proposition 17 in [3].

PROPOSITION 5.2
A type φ has long principal normal inhabitants iff there is at least one principal proof tree $\mathtt{PPT}$ constructed from $\mathtt{tree}(\varphi_{\mathtt{i}})$. Furthermore, every term in $\mathtt{Terms(PPT)}$ is a principal long normal inhabitant of φ and whenever M is a long principal normal inhabitant of φ, then $\mathtt{PT} = \mathtt{pt}(M, \varphi_{\mathtt{i}}, \emptyset)$ is a principal proof tree.

PROOF. The proof of this result follows the lines of the proof of Proposition 17 in [3]. ∎

EXAMPLE 5.3
We conclude that the term $M = \lambda xy.y(\lambda t.x(\lambda z.y(\lambda w.z))t)$ (example 4.4) is a long normal, but no principal inhabitant of θ (example 2.3). In order to obtain a principal proof tree of θ it is necessary to further identify a_2 or a_4 with at least one of a_1, a_3 or a_5. This can be done by substituting in the proof tree of example 3.2 the (right) subtree

$$
\begin{array}{c}
\ldots = a_4 \\
|
\end{array}
\qquad \text{by} \qquad
\begin{array}{ccc}
 & \ldots = a_3 & \\
 & / \quad \backslash & \\
b_1 = b_2 & & a_2 = a_4 \\
| & & | \\
a_5 = a_1 & & \\
| & &
\end{array}
$$

obtaining thus the principal proof tree PPT =

$$\begin{array}{c} | \\ b_3 = b_2 \\ | \\ a_5 = a_3 \end{array}$$

$$\begin{array}{c} b_1 = b_2 \\ | \\ a_5 = a_1 \\ | \end{array} \qquad\qquad \begin{array}{ccc} & a_2 = a_3 & \\ & / \quad \backslash & \\ \begin{array}{c} b_1 = b_2 \\ | \\ a_5 = a_1 \\ | \end{array} & & \begin{array}{c} a_2 = a_4 \\ | \\ \\ \\ \end{array} \end{array}$$

with restrictions $a_1 = a_2 = a_3 = a_4 = a_5$ and $b_1 = b_2 = b_3$ and using every primitive part of form (p2) in $\mathtt{tree}(\theta_\mathtt{i})$. Consequently, the (unique) member of the corresponding set of terms

$$\mathtt{Terms(PPT)} = \{\lambda xy.y(\lambda t.x(\lambda z.y(\lambda w.z))(x(\lambda u.y(\lambda v.u))t))\}$$

is a principal long normal inhabitant of θ.

In the previous example, note that the occurrence b_3, which corresponds to the tail of θ, had to be identified with two other occurrences of b. Consequently, in any principal proof tree there must be at least one other node, besides the root of the tree, identifying occurrences of b. In fact, in the principal proof tree PPT from the example there are two such nodes, both of the form $b_1 = b_2$. Now it is easy to see that we obtain another (more complex) valid principal proof tree when we substitute the subtree in either one of these nodes by the whole principal proof tree. We conclude the following for the general case.

PROPOSITION 5.4
Whenever the number of occurrences of the tail of a type φ is greater than or equal to three then the number of principal normal inhabitants of τ is either zero or infinite. ■

6 Another example

Consider the type

$$\alpha = ((((a \to a) \to a) \to a) \to a) \to a \to a$$

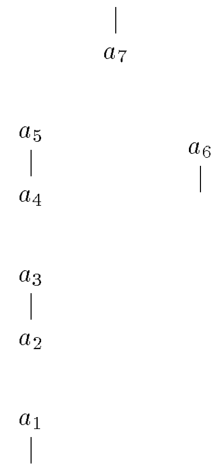
which has an infinite number of (long) normal inhabitants, such as $\lambda xy.y$, $\lambda xy.x(\lambda z.y)$, $\lambda xy.x(\lambda z.z(\lambda u.y))$ or $\lambda xy.x(\lambda z.z(\lambda u.u))$, neither of which is principal. But does α have any principal normal inhabitant at all?

Using the algorithm defined in [3] and [4], we can answer this question after generating and testing a "finite" set of type-schemes with depth $\leq |\alpha|^4$, i.e. with depth $\leq 7^4 = 2401$. To understand the real complexity of this task, note that the four terms above have respectively depth 0, 1, 2 and 2.

On the other hand, analysing the formula tree of the indexed counterpart of α

$$\alpha_{\mathtt{i}} = ((((a_1 \to a_2) \to a_3) \to a_4) \to a_5) \to a_6 \to a_7$$

which is $\mathtt{tree}(\alpha_{\mathtt{i}})$ =



with primitive parts

$$p_1 = \begin{array}{c} | \\ a_7 \end{array} \qquad p_2 = \begin{array}{c} a_5 \\ | \\ a_4 \end{array} \qquad p_3 = \begin{array}{c} a_3 \\ | \\ a_2 \end{array} \qquad p_4 = \begin{array}{c} a_1 \\ | \end{array} \qquad p_5 = \begin{array}{c} a_6 \\ | \end{array},$$

we conclude that any valid proof tree built from $\mathtt{tree}(\alpha_{\mathtt{i}})$ = uses either p_4 or p_5 and will thus not identify a_1 or a_6 with the remaining occurrences of a in α. Consequently, none of the infinite number of valid proof trees for $\mathtt{tree}(\alpha_{\mathtt{i}})$ is a principal one and we conclude that α has in fact no principal normal inhabitants at all.

It is easy to see that every valid proof tree built from $\mathtt{tree}(\alpha_{\mathtt{i}})$ is either $\begin{array}{c} p_1 \\ p_5 \end{array}$, i.e. is $\begin{array}{c} | \\ a_7 = a_6 \\ | \end{array}$, or else of the form

$$\begin{array}{c} p_1 \\ p_2 \\ p_{i_1} \\ \vdots \\ p_{i_k} \\ p_f \end{array}$$

where $k \geq 0$, $i_1, \ldots, i_k \in \{2, 3\}$ and $f \in \{4, 5\}$. Via algorithm `Terms` we thus get a concrete idea of the structure of the set of normal inhabitants of α.

Acknowledgements

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

References

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

[2] C.-B. Ben-Yelles. *Type-assignment in the lambda-calculus; syntax and semantics.* PhD thesis, Mathematics Dept., University of Wales Swansea, UK, 1979.

[3] S. Broda and L. Damas. Counting a type's principal inhabitants. Proceedings of the 4th International Conference on Typed Lambda Calculi and Applications, *TLCA'99*, *Lecture Notes in Computer Science* 1581, pág. 69-82, Springer Verlag, 1999.

[4] S. Broda and L. Damas. Counting a type's (principal) inhabitants. To appear in *Fundamenta Informaticae.*

[5] A. Church, A formulation of the simple theory of types. *Journal of Symbolic Logic*, **5**, 56–68, 1940.

[6] H. B. Curry. Functionality in combinatory logic. *Proc. Nat. Acad. Science USA*, **20**, 584–590, 1934.

[7] R. Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *Journal of Symbolic Logic*, **57**, 795–807, 1992.

[8] G. Gentzen. Untersuchungen über das logische Schliessen. *Math. Zeitschrift*, **39**, 176–210, 1935.

[9] J. R. Hindley. *Basic Simple Type Theory.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1997.

[10] S. Hirokawa. Note: Infiniteness of proof(α) is polynomial-space complete. *Theoretical Computer Science*, 206:331–339, 1998.

[11] W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980.

[12] R. Statman. Intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science*, 9:67–72, 1979.