Generating normal inhabitants of types with a common structure

Sabine Broda, Luís Damas

Technical Report Series: DCC-01-01

Departamento de Ciência de Computadores – Faculdade de Ciências

&

Laboratório de Inteligência Artificial e Ciência de Computadores

---

Universidade do Porto

Rua do Campo Alegre, 823 4150 Porto, Portugal

Tel: +351+2+6078830 – Fax: +351+2+6003654

http://www.ncc.up.pt/fcup/DCC/Pubs/treports.html

Generating normal inhabitants of types with a common structure

Sabine Broda, Luís Damas
DCC & LIACC, Universidade do Porto
R. do Campo Alegre 823, 4150-180 Porto, Portugal
Phone: 351 2 6078830, Fax: 351 2 6003654
e-mail: {sbb,luis}@ncc.up.pt

Abstract

In [10] it was shown that it is possible to describe the set of normal inhabitants of a given type τ, in the standard simple type system, using an infinitary extension of the concept of context-free grammar, which allows for an infinite number of non-terminal symbols as well as production rules. The set of normal inhabitants of τ corresponds then to the set of terms generated by this, possibly infinitary, grammar plus all terms obtained from those by η-reduction. In this paper we show that the set of normal inhabitants of a type τ can in fact be described using a standard (finite) context-free grammar, and more interestingly that the sets of normal inhabitants of all types with a same structure are described by context-free grammars which share one unique underlying structure. The definition of a common scheme for these grammars, which depends uniquely on the given type structure, is based on an alternative representation for types, introduced in [4], which gives us a better insight on the nature of a type's structure and its relation to the structure of the set of its normal inhabitants.

1 Introduction

The relation of terms and types in the system $\mathbf{TA}_\lambda$ of simply typed λ-calculus has been a major subject of study over the last decades. The interest in this area is due to its importance to areas of mathematical logic and more recently to computer science. In fact, it is well known that there exists a direct correspondence (cf. [8]), via Curry-Howard isomorphism, between $\mathbf{TA}_\lambda$ and the implicational fragment of intuitionistic propositional logic. As such, a type τ can be assigned to some λ-term if and only if τ is a provable formula/theorem. Furthermore, every (normal) term to which τ can be assigned, i.e. every (normal) inhabitant of τ, represents a (normal) proof of τ in the natural deduction system for the implicational fragment of intuitionistic logic. On the other hand $\mathbf{TA}_\lambda$ finds a major field of application in the area of functional programming. The concept of functional programming relies on the representation of computable functions/algorithms by λ-terms, and type assignment to λ-terms provides for partial specification of the represented algorithms. Almost all practical systems have the system $\mathbf{TA}_\lambda$ at their basis.

One direction of this study seeks for properties of (normal) inhabitants of types in $\mathbf{TA}_\lambda$. The decision problem for inhabitation as well as the infiniteness problem of the set of normal inhabitants have been shown to be polynomial-space complete, respectively by Statman in [9] and by Hirokawa in [7]. In [2], Ben-Yelles defined an algorithm, also described in [6], to count and list the normal inhabitants of a type. An adaptation of his algorithm for the subsystem λI and similar algorithms for principal normal inhabitants in both system were presented in [3]. Other algorithms, which for every type produce a normal inhabitant or guarantee the non-existence of inhabitants, were given by Bunder in [5].

In [10] it was shown that it is possible to describe the set of normal inhabitants of a type, using an infinitary extension of the concept of context-free grammar, which allows for an infinite number of non-terminal symbols as well as production rules. The set of normal inhabitants of τ corresponds then to the set of terms generated by this, possibly infinitary, grammar plus

all terms obtained from those by η-reduction. In this paper we show that the set of normal inhabitants of a type τ can in fact be described using a standard (finite) context-free grammar, and more interestingly that the sets of normal inhabitants of all types with a same structure are described by context-free grammars which share one unique underlying structure. This result partly explains an observation made by Hindley in [6], where he pointed out that "when we ask for the number of normal inhabitants of a type, the answer is often finite and interesting patterns show up which are still not completely understood." On the other hand, it provides us with a new angle to take into account in the study of the relation between the structure of formulas and their proofs. The definition of a common grammar-scheme for a given type-structure, is based on an alternative representation for types, introduced in [4], which will be recalled in section 2.

2 Background

We assume familiarity with the basic notions in λ-calculus and use standard notation from [1] and [6]. 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. This choice results from the standard notation for context-free grammars and from the fact that we will use an annotated version of the atoms in a type as non-terminal symbols of the context-free grammar which describes the normal inhabitants of this type. For type assignment we consider the system $\mathbf{TA}_\lambda$ of simply typed λ-calculus à la Curry (for an introduction see [6] or [1], section 3.1).

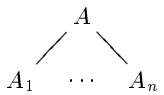
In this section we recall the definition of an alternative tree-like representation for types, called formula trees, first introduced in [4], which gives us an exact idea on the different stages during the construction of normal proofs/inhabitants of a formula/type. In fact, the formula tree of a type τ defines some kind of hierarchy over the primitive parts of τ which can be used to construct proofs of τ represented by proof trees.

2.1 Formula trees and proof trees

Formula trees are trees whose nodes consist of primitive parts which are of either one of the following forms.

(p1)

(p2)



$(n \geq 1)$

(p3)

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

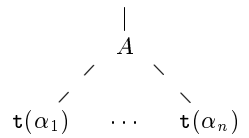
- *the root of the tree is of form (p1);*
- *every internal node of the tree is of form (p2);*
- *every leaf of the tree is of form (p2) or (p3).*

The following algorithm computes the formula tree $\texttt{tree}(\varphi)$ of a type φ. We 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

$\texttt{tree}(\varphi) =$

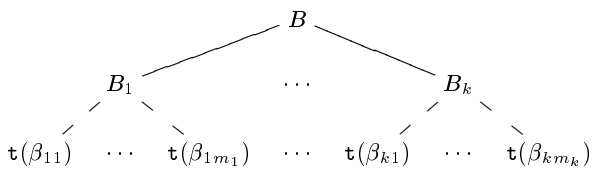
- If $n \geq 1$, then

$\mathtt{tree}(\varphi) =$ 

where

$\mathtt{t}(B) =$

and finally for $k \geq 1$ and $m_1, \ldots, m_k \geq 0$ take
$\mathtt{t}((\beta_{11} \to \ldots \to \beta_{1m_1} \to B_1) \to \ldots \to (\beta_{k1} \to \ldots \to \beta_{km_k} \to B_k) \to B) =$

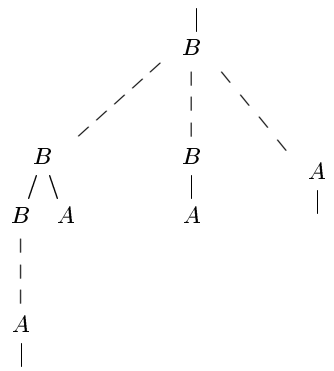


Note that one can easily define an inverse algorithm, which given a formula tree FT computes the unique type φ such that $\mathtt{tree}(\varphi) = \mathtt{FT}$.

Example 2.2 *The formula*

$$\alpha = ((A \to B) \to A \to B) \to (A \to B) \to A \to B$$

has the following formula tree



with primitive parts

$p_0 =$ $p_1 =$ $p_2 =$ $p_3 =$ $p_4 =$.

Definition 2.3 *A proof tree built from a formula tree* $\mathtt{tree}(\varphi)$ *is obtained by joining primitive parts of* $\mathtt{tree}(\varphi)$ *identifying/overlapping (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* $\mathtt{tree}(\varphi)$*, 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 of φ defines some kind of hierarchy over its primitive parts, which has to be respected by every valid proof tree built from $\texttt{tree}(\varphi)$. In particular, this means that the root of every valid proof tree built from $\texttt{tree}(\varphi)$ is the root of $\texttt{tree}(\varphi)$.

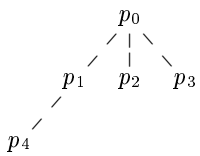
Example 2.4 *A valid proof tree for the formula in example 2.2 is*

$$\begin{array}{lll} & & | \\ p_0 & & B \\ p_2 & \textit{which looks like} & | \\ p_3 & & A \\ & & | \end{array}$$

On the other hand,

$$\begin{array}{l} p_0 \\ p_2 \\ p_4 \end{array}$$

is no valid proof tree (though it would have the same appearance), since it does not respect the hierarchy given by the formula tree



of

$$((A \to B) \to A \to B) \to (A \to B) \to A \to B$$

which requires that p_4 should only be used beneath occurrences of both p_0 and p_1. Other valid proof trees are

$$\begin{array}{l} p_0 \\ p_1 \\ p_2\ p_3 \\ p_4 \end{array} \quad \textit{and} \quad \begin{array}{l} p_0 \\ p_1 \\ p_2\ p_3 \\ p_3 \end{array} \quad \textit{both with appearance} \quad \begin{array}{ccc} & | & \\ & B & \\ & / \ \backslash & \\ B & & A \\ | & & | \\ A & & \\ | & & \end{array} .$$

2.2 Proof trees and (long) normal inhabitants

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 $(zP_1 \ldots P_n)$, $(n \geq 0)$ 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], [6]) 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], [6]) 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 [6]. Thus, when seeking for normal inhabitants of a type it is sufficient to compute the set its long normal inhabitants from which all normal inhabitants can be obtained by η-reduction.

In [4] an algorithm was defined which given a long inhabitant M of a type φ computes a valid proof tree $\texttt{PT}(M)$ of $\texttt{tree}(\varphi)$.

Proposition 2.5 *If M is a long normal inhabitant of a type φ, then $\texttt{PT}(M)$ is a valid proof tree built from $\texttt{tree}(\varphi)$.*

Also, an *inverse* algorithm was given in [4], which given the formula tree of a type φ, $\mathtt{t} = \mathtt{tree}(\varphi)$, and any valid proof tree $\mathtt{PT}$ built from it, computes a set $\mathtt{Terms}(\mathtt{PT})$ of long closed normal inhabitants of φ.

Proposition 2.6 *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* $\mathtt{Terms}(\mathtt{PT})$ *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* $M \in \mathtt{Terms}(\mathtt{PT}(M))$.

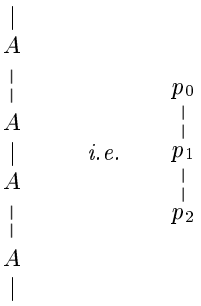
Resuming, every normal inhabitant of a type φ corresponds to one valid proof tree built from $\mathtt{tree}(\varphi)$, every valid proof tree built from $\mathtt{tree}(\varphi)$ corresponds to a finite set of normal inhabitants of φ, and distinct valid proof trees correspond to distinct and disjoint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

3 A context-free grammar description of $\mathtt{Nhabs}(\tau)$

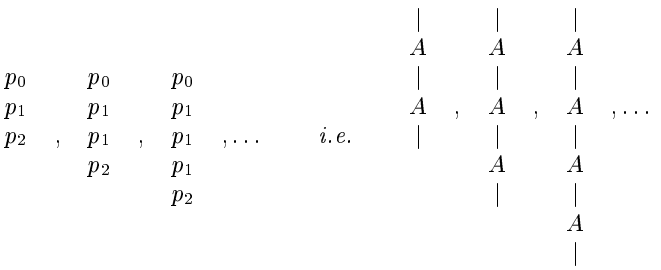
3.1 Using term-schemes instead of proof trees

Up to now we saw that the set of normal inhabitants of a type τ can be computed by generating all proof trees of τ, applying the algorithm $\mathtt{Terms}$ and collecting all terms which can be obtained from these by η-reduction. In the following we will introduce the notion of term-scheme for λ-terms, in such a way that for each proof tree $\mathtt{PT}$ there is a term-scheme T such that $\mathtt{Terms}(\mathtt{PT}) = \mathcal{C}(T)$, where $\mathcal{C}(T)$ is the set of λ-terms represented by T.

Example 3.1 *The formula* $((A \to A) \to A) \to A$ *with formula tree*



has the following proof trees



For these proof trees the algorithm $\mathtt{Terms}$ *produces respectively the following sets of long normal inhabitants:*

$\{\ \lambda x_1.x_1(\lambda x_2.x_2)\ \}$, $\{\ \lambda x_1.x_1(\lambda x_2.x_1(\lambda x_2'.x_2)),\ \lambda x_1.x_1(\lambda x_2.x_1(\lambda x_2'.x_2'))\ \}$,

$\{\ \lambda x_1.x_1(\lambda x_2.x_1(\lambda x_2'.x_1(\lambda x_2''.x_2))),\ \lambda x_1.x_1(\lambda x_2.x_1(\lambda x_2'.x_1(\lambda x_2''.x_2'))),$
$\lambda x_1.x_1(\lambda x_2.x_1(\lambda x_2'.x_1(\lambda x_2''.x_2')))\ \}$, . . .

The previous example illustrates the fact that, although a proof tree may in fact represent more than one, but a finite number of, long normal inhabitants of a type, all those inhabitants follow a common pattern which will be captured by the notion of term-scheme. As such, for every valid proof tree PT for a type τ, there will be a term-scheme representing exactly the terms computed by Terms(PT).

Definition 3.2 *Syntactically term-schemes are λ-terms. The (finite) set of λ-terms represented by a term-scheme T with variables $x_1, \ldots, x_n$ is defined by $\mathcal{C}(T) = T^{0\ldots0}_{x_1\ldots x_n}$ as follows.*

- $x_i{}^{k_1\ldots0\ldots k_n}_{x_1\ldots x_i\ldots x_n} = \{\overline{x_i}\}$ *(renaming of free occurrences of variables in T);*
- $x_i{}^{k_1\ldots k_i\ldots k_n}_{x_1\ldots x_i\ldots x_n} = \{x_i, x_i', x_i'', \ldots, x_i^{k_i-1}\}$ *for $k_i \geq 1$;*
- $(ST)^{k_1\ldots k_n}_{x_1\ldots x_n} = \{S_iT_j \mid S_i \in S^{k_1\ldots k_n}_{x_1\ldots x_n}, T_j \in T^{k_1\ldots k_n}_{x_1\ldots x_n}\}$;
- $(\lambda x_i.T)^{k_1\ldots k_n}_{x_1\ldots x_n} = \{\lambda x_i^{k_i}.T_i \mid T_i \in T^{k_1\ldots(k_i+1)\ldots k_n}_{x_1\ldots x_i\ldots x_n}\}$.

Example 3.3 *The scheme*

$$T = \lambda xy.x(\lambda xy.xy)(\lambda y.xy)$$

represents the following set of λ-terms.

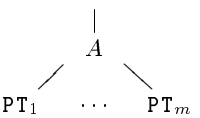
$$\begin{array}{rl}\mathcal{C}(T) = \{ & \lambda xy.x(\lambda x'y'.xy)(\lambda y'.xy),\\ & \lambda xy.x(\lambda x'y'.xy)(\lambda y'.xy'),\\ & \lambda xy.x(\lambda x'y'.xy')(\lambda y'.xy),\\ & \lambda xy.x(\lambda x'y'.xy')(\lambda y'.xy'),\\ & \lambda xy.x(\lambda x'y'.x'y)(\lambda y'.xy),\\ & \lambda xy.x(\lambda x'y'.x'y)(\lambda y'.xy'),\\ & \lambda xy.x(\lambda x'y'.x'y')(\lambda y'.xy),\\ & \lambda xy.x(\lambda x'y'.x'y')(\lambda y'.xy')\ \}\end{array}$$

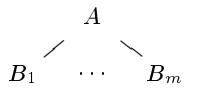
Example 3.4 *The sets of long normal inhabitants corresponding to the proof trees in example 3.1 can respectively be represented by the following term-schemes.*

$$\lambda x_1.x_1(\lambda x_2.x_2),\ \lambda x_1.x_1(\lambda x_2.x_1(\lambda x_2.x_2)),\ \lambda x_1.x_1(\lambda x_2.x_1(\lambda x_2.x_1(\lambda x_2.x_2))), \ldots$$

In the following we describe an algorithm which, given a type τ and a valid proof tree PT built from tree(τ), computes a term-scheme T = TS(PT) representing the set of long normal inhabitants that corresponds to PT.

Consider the root $\overset{|}{A}$ of PT and let $p_1, \ldots, p_n$, with $n \geq 0$, be the descendents of the corresponding occurrence of A in tree(τ).

- If PT is of the form $\overset{|}{A}$ $\overset{|}{}$, then take the corresponding primitive part $p_i = \overset{A}{|}$ in tree(τ) and let
$$\mathtt{TS}(\mathtt{PT}) = \lambda x_1 \ldots \lambda x_n.x_i.$$
- Otherwise, PT is of the form


with $m \geq 1$, and such that for $j = 1, \ldots, m$ the root of $\mathtt{PT}_j$ is $\overset{|}{B_j}$. Consider the corresponding primitive part p_i of the form


and let

$$\mathtt{TS}(\mathtt{PT}) = \lambda x_1 \ldots \lambda x_n . x_i M_1 \ldots M_m,$$

where for $1 \leq j \leq m$,

$$M_j \in \mathtt{TS}(\mathtt{PT}_j).$$

The following result establishes the correctness of the algorithm TS.

Proposition 3.5 *Let τ be a type and* PT *a valid proof tree built from* $\mathtt{tree}(\tau)$. *Then, modulo α-conversions,*

$$\mathcal{C}(\mathtt{TS}(\mathtt{PT})) = \mathtt{Terms}(\mathtt{PT}).$$

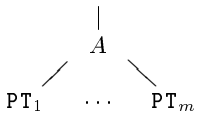
Proof The result should become quit clear after recalling the definition of the algorithm Terms in [4]. Given the formula tree of a type φ, $\mathtt{t} = \mathtt{tree}(\varphi)$, and any valid proof tree PT built from it, $\mathtt{Terms}(\mathtt{PT}) = \mathtt{Terms}(\mathtt{t}, 0, \mathtt{PT}, \emptyset)$ is defined as follows. In the following let L represent a set assigning variable names to primitive parts of t. Then, $\mathtt{Terms}(\mathtt{t}, k, \mathtt{PT}, L)$ is defined by the following.

Let $\begin{array}{c}|\\A\end{array}$ be the root of PT and consider the corresponding occurrence of A in t with (direct) descendents $p_1, \ldots, p_n$, $n \geq 0$. Now, let $L' = L \cup \{(p_i, x_i^k) \mid 1 \leq i \leq n\}$.

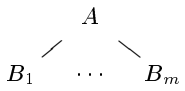
- If PT is of the form $\begin{array}{c}|\\A\\|\end{array}$, then take the corresponding primitive part $p = \begin{array}{c}A\\|\end{array}$ and let

$$\mathtt{Terms}(\mathtt{t}, k, \mathtt{PT}, L) = \{\lambda x_1^k \ldots \lambda x_n^k . x \mid (p, x) \in L'\}.$$

- Otherwise, PT is of the form



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



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

$$\{\lambda x_1^k \ldots \lambda x_n^k . x M_1 \ldots M_m \mid (p, x) \in L', M_j \in \mathtt{Terms}(\mathtt{t}, k+1, \mathtt{PT}_j, L'), 1 \leq j \leq m\}$$

Now note that, when erasing all superscripts of variables in the members of $\mathtt{Terms}(\mathtt{PT})$, one obtains exactly one term, which is $\mathtt{TS}(\mathtt{PT})$. On the other hand, the computation done by $\mathcal{C}$ is performed during the construction of $\mathtt{Terms}(\mathtt{PT})$: in fact, if an abstraction over a variable x occurs in the range of another abstraction over x, then the two occurrences of x receive different superscripts, the available superscripts for x are stored in the set L and whenever x is used, all possibilities (i.e. all possible superscripts) are explored. •

3.2 The context-free grammar $\mathtt{G}_\tau$

In this subsection we will, for every type τ, define a context-free grammar $\mathtt{G}_\tau$ which generates a set of term-schemes, that correspond exactly to the long normal inhabitants of τ.

Definition 3.6 *Let τ be a type, with atoms $A_1, \ldots, A_m$ and such that* $\mathtt{tree}(\tau)$ *has primitive parts $p_0, \ldots, p_n$, where p_0 represents the root of* $\mathtt{tree}(\tau)$. *Let* $\mathtt{G}(\tau) = (T, N, R, S)$ *be the context-free grammar with*

- *set of terminal symbols* $T = \{(,), \lambda, ., x_1, \ldots, x_n\}$,

- *set of non-terminal symbols* $N = \{S\} \cup \{A_i^P \mid 1 \leq i \leq m, P \in 2^{\{1,\ldots,n\}}\}$
- *start symbol* S

and such that R *is the smallest set satisfying the following conditions.*

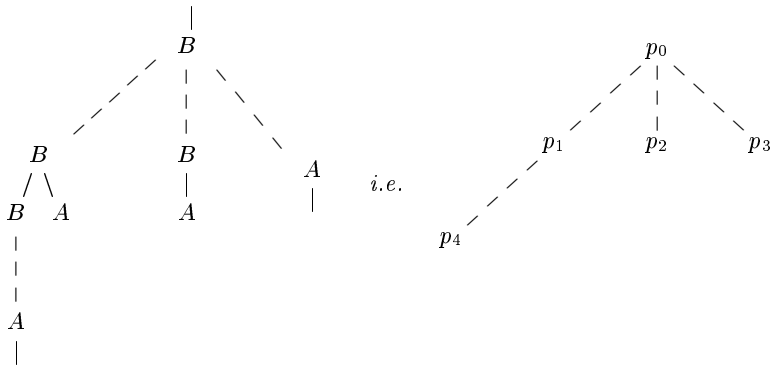
- *if* $p_0 = \begin{array}{c}|\\ A_s\end{array}$ *has descendents* $p_{i_1}, \ldots, p_{i_t}$ *in* $\mathtt{tree}(\tau)$*, then* R *has exactly one production rule for* S *which is*
$$S \to \lambda x_{i_1} \ldots x_{i_t}.A_s^{\{i_1,\ldots,i_t\}}$$
- *whenever a non-terminal symbol* A_j^P *appears on the right side of a production rule in* R*, then for every* $i \in P$ *such that* p_i *is of the form*
$$\begin{array}{c}A_j\\|\end{array}$$
there is a rule in R *of the form*
$$A_j^P \to x_i$$
- *whenever a non-terminal symbol* A_j^P *appears on the right side of a production rule in* R*, then for every* $i \in P$ *such that* p_i *is of the form*
$$\begin{array}{ccccc}&&A_j&&\\&\swarrow&&\searrow&\\A_{j_1}&&\cdots&&A_{j_s}\end{array}$$
and such that to each A_{j_l} *correspond the descendents* $p_{j_l^1}, \ldots, p_{j_l^{t_l}}$*, there is a rule in* R *of the form*
$$A_j^P \to x_i(\lambda x_{j_1^1} \ldots x_{j_1^{t_1}}.A_{j_1}^{P_1}) \ldots (\lambda x_{j_s^1} \ldots x_{j_s^{t_s}}.A_{j_s}^{P_s})$$
where $P_l = P \cup \{j_l^1, \ldots, j_l^{t_l}\}$.

Proposition 3.7 *For any type* τ *the grammar* $\mathtt{G}_\tau$ *describes* $\mathtt{Nhabs}(\tau)$ *in the sense that*

$$\mathtt{Nhabs}(\tau) = \{M \mid \exists T \in \mathcal{L}(\mathtt{G}_\tau)\ \exists N \in \mathcal{C}(T) \text{ such that } N \to_\eta M\}.$$

Proof Note, that the definition of $\mathtt{G}_\tau$ is essentially a description of the possibilities of constructing valid proof trees from $\mathtt{tree}(\tau)$. As such, the production rules for a non-terminal symbol $A^{i_1,\ldots,i_n}$ describe the possible ways of overlapping an occurrence of A with another occurrence of A in a primitive part p_i of $\mathtt{tree}(\tau)$. For this, p_i has to be available, corresponding to the condition $i \in \{i_1, ..., i_n\}$ and A has to be the root or p_i. Then, instead of generating valid proof trees, these are simultaneously translated to the corresponding term scheme they represent. •

Example 3.8 *For* α *as in example 2.2 with formula tree*



there is $\mathtt{G}_\alpha = (T, N, R, S)$

- $T = \{(,), \lambda, ., x_1, x_2, x_3, x_4\}$,

- $N = \{S\} \cup \{A^P \mid P \subseteq \{1,2,3,4\}\} \cup \{B^P \mid P \subseteq \{1,2,3,4\}\}$,
- *and production rules*

$$\begin{array}{lcl} S & \to & \lambda x_1 x_2 x_3.\ B^{123} \\ B^{123} & \to & x_1(\lambda x_4.\ B^{1234})(\ A^{123}) \\ B^{123} & \to & x_2(\ A^{123}) \\ B^{1234} & \to & x_1(\lambda x_4.\ B^{1234})(\ A^{1234}) \\ B^{1234} & \to & x_2(\ A^{1234}) \\ A^{123} & \to & x_3 \\ A^{1234} & \to & x_3 \\ A^{1234} & \to & x_4 \end{array}$$

This grammar generates the following infinite set of term-schemes.

$$\begin{array}{rl} \mathcal{L}(\mathtt{G}_\alpha) = \{ & \lambda x_1 x_2 x_3.x_2 x_3, \\ & \lambda x_1 x_2 x_3.x_1(\lambda x_4.x_2 x_3)x_3, \\ & \lambda x_1 x_2 x_3.x_1(\lambda x_4.x_2 x_4)x_3, \\ & \lambda x_1 x_2 x_3.x_1(\lambda x_4.x_1(\lambda x_4.x_2 x_3)x_3)x_3, \\ & \lambda x_1 x_2 x_3.x_1(\lambda x_4.x_1(\lambda x_4.x_2 x_3)x_4)x_3, \\ & \lambda x_1 x_2 x_3.x_1(\lambda x_4.x_1(\lambda x_4.x_2 x_4)x_3)x_3, \\ & \lambda x_1 x_2 x_3.x_1(\lambda x_4.x_1(\lambda x_4.x_2 x_4)x_4)x_3, \ldots \} \end{array}$$

Thus,

$$\begin{array}{rcl} \mathtt{Nhabs}(\alpha) & = \{ & M \mid \exists T \in \mathcal{L}\ \exists N \in \mathcal{C}(T)\ \textit{such that}\ N \to_\eta M\} \\ & = \{ & \lambda x_1 x_2 x_3.x_2 x_3, \\ & & \lambda x_1 x_2 x_3.x_1(\lambda x_4.x_2 x_3)x_3, \\ & & \lambda x_1 x_2 x_3.x_1(\lambda x_4.x_2 x_4)x_3, \\ & & \lambda x_1 x_2 x_3.x_1 x_2 x_3, \\ & & \lambda x_1 x_2 x_3.x_1(\lambda x_4.x_1(\lambda x'_4.x_2 x_3)x_3)x_3, \\ & & \lambda x_1 x_2 x_3.x_1(\lambda x_4.x_1(\lambda x'_4.x_2 x_3)x_4)x_3, \\ & & \lambda x_1 x_2 x_3.x_1(x_1(\lambda x'_4.x_2 x_3))x_3, \\ & & \lambda x_1 x_2 x_3.x_1(\lambda x_4.x_1(\lambda x'_4.x_2 x_4)x_3)x_3, \\ & & \lambda x_1 x_2 x_3.x_1(\lambda x_4.x_1(\lambda x'_4.x_2 x'_4)x_3)x_3, \\ & & \lambda x_1 x_2 x_3.x_1(\lambda x_4.x_1 x_2 x_3)x_3, \\ & & \lambda x_1 x_2 x_3.x_1(\lambda x_4.x_1(\lambda x'_4.x_2 x_4)x_4)x_3, \\ & & \lambda x_1 x_2 x_3.x_1(\lambda x_4.x_1(\lambda x'_4.x_2 x'_4)x_4)x_3, \\ & & \lambda x_1 x_2 x_3.x_1(\lambda x_4.x_1 x_2 x_4)x_3, \ldots \} \end{array}$$

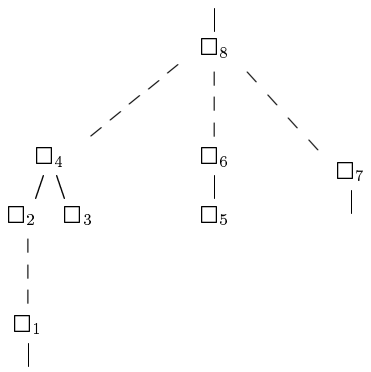
4 A general grammar scheme for a given type-structure

Given a type τ we represent the corresponding type-structure replacing occurrences of atoms by blanc successively indexed boxes as illustrated in the following example.

Example 4.1 *The type-structure corresponding to α from example 2.2 is*

$$\Theta = ((\Box_1 \to \Box_2) \to \Box_3 \to \Box_4) \to (\Box_5 \to \Box_6) \to \Box_7 \to \Box_8$$

with formula tree



Possible instances, i.e types with this structure, are for example,

1. *for* $\Box_1 = \Box_2 = \Box_3 = \Box_5 = \Box_7 = A$ *and* $\Box_4 = \Box_6 = \Box_8 = B$ *the type*

$$\alpha_1 = ((A \to A) \to A \to B) \to (A \to B) \to A \to B$$

2. *for* $\Box_1 = \Box_2 = A$, $\Box_3 = \Box_7 = B$ *and* $\Box_4 = \Box_5 = \Box_6 = \Box_8 = C$ *the type*

$$\alpha_2 = ((A \to A) \to B \to C) \to (C \to C) \to B \to C$$

3. *or for* $\Box_1 = \Box_2 = \Box_3 = \Box_4 = \Box_5 = \Box_6 = \Box_8 = A$ *and* $\Box_7 = B$ *the type*

$$\alpha_3 = ((A \to A) \to A \to A) \to (A \to A) \to B \to A$$

In the following we define an algorithm which given a type-structure Θ computes a grammar(-scheme) $\mathtt{G}_\Theta$ such that for every instance α with structure Θ a context-free grammar equivalent to $\mathtt{G}_\alpha$ can be obtained from $\mathtt{G}_\Theta$ by substitution of all occurrences of indexed boxes in this grammar by the corresponding atoms in α.

Definition 4.2 *Let* Θ *be a type-structure, with indexed boxes* $\Box_1, \ldots, \Box_n$ *and with tail* $\Box_n$, *i.e.* $\Box_n$ *is the box in the root of* $\mathtt{tree}(\Theta)$. *Let* $\mathtt{G}(\Theta) = (T, N, R, S)$ *be the context-free grammar with*

- *set of terminal symbols* $T = \{(,), \lambda, ., x_1, \ldots, x_n\}$,
- *set of non-terminal symbols* $N = \{S\} \cup \{\Box_i^P \mid 1 \leq i \leq n, P \in 2^{\{1,\ldots,n\}}\}$
- *start symbol* S

and such that R *is the smallest set satisfying the following conditions.*

- *if the root* $\begin{array}{c}|\\ \Box_n\end{array}$ *has as descendents primitive parts, whose roots are respectively* $\Box_{i_1}, \ldots, \Box_{i_t}$ *in* $\mathtt{tree}(\Theta)$, *then* R *has exactly one production rule for* S *which is*

$$S \to \lambda x_{i_1} \ldots x_{i_t}.\Box_n^{\{i_1,\ldots,i_t\}}$$

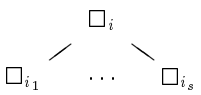
- *whenever a non-terminal symbol* $\Box_j^P$ *appears on the right side of a production rule in* R, *then for every* $i \in P$ *such that there is a (unique) primitive part in* $\mathtt{tree}(\Theta)$ *of the form*

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

there is a rule in R *of the form*

$$\Box_i^P \to x_i$$

- *whenever a non-terminal symbol* $\Box_j^P$ *appears on the right side of a production rule in* R*, then for every* $i \in P$ *such that* p_i *is of the form*



and such that each $\Box_{i_l}$ *has as descendents primitive parts respectively with roots* $\Box_{i_l^1}, \ldots, \Box_{i_l^{t_l}}$*, there is a rule in* R *of the form*

$$\Box_i^P \to x_i(\lambda x_{i_1^1} \ldots x_{i_1^{t_1}}.\Box_{i_1}^{P_1}) \ldots (\lambda x_{i_s^1} \ldots x_{i_s^{t_s}}.\Box_{i_s}^{P_s})$$

where $P_l = P \cup \{i_l^1, \ldots, i_l^{t_l}\}$.

Example 4.3 *Let* Θ *be as in example 4.1. Then,* $\mathtt{G}_\Theta = (T, N, R, S)$ *with*

- $T = \{(,), \lambda, ., x_1, \ldots, x_n\}$,
- $N = \{S\} \cup \{\Box_i^P \mid 1 \leq i \leq 8, P \in 2^{\{1,\ldots,8\}}\}$
- *start symbol* S

and R *given by*

$$\begin{array}{lcllcl}
S & \to & \lambda x_4 x_6 x_7.\ \Box_8{}^{467} & \Box_1{}^{1467} & \to & x_1 \\
\Box_4{}^{467} & \to & x_4(\lambda x_1.\ \Box_2{}^{1467})(\ \Box_3{}^{467}) & \Box_4{}^{1467} & \to & x_4(\lambda x_1.\ \Box_2{}^{1467})(\ \Box_3{}^{1467}) \\
\Box_6{}^{467} & \to & x_6(\ \Box_5{}^{467}) & \Box_6{}^{1467} & \to & x_6(\ \Box_5{}^{1467}) \\
\Box_7{}^{467} & \to & x_7 & \Box_7{}^{1467} & \to & x_7
\end{array}$$

Naturally, for every type-scheme Θ the context-free grammar $\mathtt{G}_\Theta$ generates the empty language, since no non-terminal symbol on the right side of production rules will appear on the left side of any rule in R. But, the following proposition shows that $\mathtt{G}_\Theta$ can in fact be seen at as a scheme representing grammars for any type with structure Θ.

Proposition 4.4 *Let* τ *be a type with structure* Θ*, i.e.* $\tau = \Theta[A_1/\Box_1, \ldots, A_n/\Box_n]$*, for, not necessarily distinct, atoms* $A_1, \ldots, A_n$*. Let* $\mathtt{G}_{\tau/\Theta}$ *be the context-free grammar obtained from* $\mathtt{G}_\Theta$ *by substituting all occurrences of* $\Box_1, \ldots, \Box_n$ *respectively by* $A_1, \ldots, A_n$*. Then (modulo* α*-conversion), there is*

$$\begin{array}{rl}
\mathtt{Nhabs}(\tau) & = \{M \mid \exists T \in \mathcal{L}(\mathtt{G}_\tau)\ \exists N \in \mathcal{C}(T) \text{ such that } N \to_\eta M\} \\
& = \{M \mid \exists T \in \mathcal{L}(\mathtt{G}_{\tau/\Theta})\ \exists N \in \mathcal{C}(T) \text{ such that } N \to_\eta M\}.
\end{array}$$

Proof First notice that we probably use some distinct kind of enumeration for $\mathtt{G}_\tau$ and $\mathtt{G}_\Theta$ which leads to distinct indexes of variables and superscripts of non-terminal symbols. So let us consider $\mathtt{G}'_\tau$ as the grammar $\mathtt{G}_\tau$ where every index and superscript i, referring to the primitive part p_i in $\mathtt{tree}(\tau)$ is replaced by the index of the box in the root of this primitive part p_i. Since this attribution of new values to indexes is obviously injective, it is sufficient to verify that $\mathcal{L}(\mathtt{G}'_\tau) = \mathcal{L}(\mathtt{G}_\Theta)$. For this, note first that every production rule of $\mathtt{G}'_\tau$ is a production rule of $\mathtt{G}_{\tau/\Theta}$. On the other hand, all other rules in $\mathtt{G}_{\tau/\Theta}$ will never be used during the generation of a term-scheme. •

Example 4.5 *For the types* α_1*,* α_2 *and* α_3 *in example 4.1 we obtain the following results.*

1. $\mathtt{G}_{\alpha_1/\Theta}$ *is obtained from* $\mathtt{G}_\Theta$ *substituting all occurrences of* $\Box_1, \Box_2, \Box_3, \Box_5$ *and* $\Box_7$ *by the atom* A *and all occurrences of* $\Box_4$ *and* $\Box_6$ *by* B. *The resulting set* R *has the following production rules*

$$\begin{array}{lcllcl}
S & \to & \lambda x_4 x_6 x_7.\ B^{467} & A^{1467} & \to & x_1 \\
B^{467} & \to & x_4(\lambda x_1.\ A^{1467})(\ A^{467}) & B^{1467} & \to & x_4(\lambda x_1.\ A^{1467})(\ A^{1467}) \\
B^{467} & \to & x_6(\ A^{467}) & B^{1467} & \to & x_6(\ A^{1467}) \\
A^{467} & \to & x_7 & A^{1467} & \to & x_7
\end{array}$$

which can be simplified to

$$\begin{array}{rcl}
S & \to & \lambda x_4 x_6 x_7.x_4(\lambda x_1.x_1)x_7 \\
 & | & \lambda x_4 x_6 x_7.x_4(\lambda x_1.x_7)x_7 \\
 & | & \lambda x_4 x_6 x_7.x_6 x_7
\end{array}$$

Thus,

$$\mathtt{Nhabs}(\alpha_1) = \{\lambda x_4 x_6 x_7.x_4(\lambda x_1.x_1)x_7, \lambda x_4 x_6 x_7.x_4(\lambda x_1.x_7)x_7, \lambda x_4 x_6 x_7.x_6 x_7\}.$$

2. $\mathtt{G}_{\alpha_2/\Theta}$ *has production rules*

$$\begin{array}{lcllcl}
S & \to & \lambda x_4 x_6 x_7.\ C^{467} & A^{1467} & \to & x_1 \\
C^{467} & \to & x_4(\lambda x_1.\ A^{1467})(\ B^{467}) & C^{1467} & \to & x_4(\lambda x_1.\ A^{1467})(\ B^{1467}) \\
C^{467} & \to & x_6(\ C^{467}) & C^{1467} & \to & x_6(\ C^{1467}) \\
B^{467} & \to & x_7 & B^{1467} & \to & x_7
\end{array}$$

which can be simplified to

$$\begin{array}{lcl}
S & \to & \lambda x_4 x_6 x_7.\ C^{467} \\
C^{467} & \to & x_4(\lambda x_1.x_1)x_7 \quad | \quad x_6\ (\ C^{467})
\end{array}$$

Thus,

$$\begin{array}{rl}
\mathtt{Nhabs}(\alpha_2) = \{ & \lambda x_4 x_6 x_7.x_4(\lambda x_1.x_1)x_7, \\
 & \lambda x_4 x_6 x_7.x_6(x_4(\lambda x_1.x_1)x_7), \\
 & \lambda x_4 x_6 x_7.x_6(x_6(x_4(\lambda x_1.x_1)x_7)), \ldots \ \}
\end{array}$$

3. $\mathtt{G}_{\alpha_3/\Theta}$ *has production rules*

$$\begin{array}{lcllcl}
S & \to & \lambda x_4 x_6 x_7.\ A^{467} & A^{1467} & \to & x_1 \\
A^{467} & \to & x_4(\lambda x_1.\ A^{1467})(\ A^{467}) & A^{1467} & \to & x_4(\lambda x_1.\ A^{1467})(\ A^{1467}) \\
A^{467} & \to & x_6(\ A^{467}) & A^{1467} & \to & x_6(\ A^{1467}) \\
B^{467} & \to & x_7 & B^{1467} & \to & x_7
\end{array}$$

This grammar generates the empty language, thus $\mathtt{Nhabs}(\alpha_3) = \emptyset$.

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

References

[1] H. Barendregt. Lambda calculi with types. In Abramsky, Gabbay, and Maibaum, eds., *Background: Computational Structures*, volume 2 of *Handbook of Logic in Computer Science*, pp. 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. *Fundamenta Informaticae*, 45:33–51, 2001.

[4] S. Broda, and L. Damas. On the structure of normal λ-terms having a certain type. *Proceedings of 7th WoLLIC'2000*, pp. 33–43, 2000.

[5] M.W. Bunder. Proof finding algorithms for implicational logics. *Theoretical Computer Science*, 232:165–186, 2000.

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

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

[8] 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*, pp. 479–490. Academic Press, 1980.

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

[10] M. Takahashi, Y. Akama, and S. Hirokawa. Normal Proofs and Their Grammar. *Information and Computation*, 125:144–153, 1996.