INHABITATION MACHINES: DETERMINISM AND PRINCIPALITY

Sandra Alves Sabine Broda

CRACS & CMUP
DCC - Faculty of Science, University of Porto
{sandra,sbb}@dcc.fc.up.pt

Abstract
Type-inhabitation is a topic of major importance, due to its close relationship to provability in logical systems and has been studied from different perspectives over the years. In this paper we revise the methods presented by Schubert et al. in 2015 for handling search for inhabitants in the simply typed lambda calculus by the use of inhabitation machines. We make adjustments to the definition of these machines, that allow us to process inhabitants in a deterministic way, as well as to address the more complex problem of principality.

1. Introduction

In the simply typed λ-calculus, the problem of associating to a type a term that inhabits it, which is known as type inhabitation, has been a major focus of research over the years. Through the Curry-Howard isomorphism the problem is equivalent to provability of formulas in the implicational fragment of propositional logic [12], and has crucial implications in the area of proof-theory. Since normal forms in the λ-calculus correspond to Prawitz's [15] notion of normal deduction, algorithms for deciding type-inhabitation can be used for indirectly decide provability. Note that typed λ-calculi derived from the Curry-Howard isomorphism led to the development of theorem assistant tools, such as Coq, where proofs are formalized as programs, which can be checked and executed, and which are valuable tools in the area of formal verification. The research carried out in the area led to a vast number of results, ranging from the definition of algorithms for generating/counting terms/proofs, to the capture of complexity classes, and the establishment of conditions guaranteeing the uniqueness of normal inhabitants of a given type. For a non-exhausting list of references we point to [5, 7, 9, 18, 17, 11, 13]. The subject has also been studied from the point of view of category-theory, where the uniqueness of type inhabitants for a given typing was established through the verification of certain syntactic constraints [2, 14, 3].

Recent work presented in [16], uses techniques from automata theory by defining inhabitation machines that are able to recognise the set of all normal inhabitants of a type. An inhabitation machine is an automata equipped with registers capable of storing a set of data elements, thus allowing the machines to deal with an infinite alphabet. When processing tree representations

of terms these machines work locally, following the *spine* of the term. Therefore they are non-deterministic and may require additional analysis to eliminate paths that do not lead to successful computations. Still, the emptiness problem for the inhabitation machines, which corresponds to the type inhabitation problem for the simply typed λ-calculus, proves to be PSPACE-complete for these machines (the authors prove this result for a restricted version of their ihnabitation machines, although one expects the same result to hold for the general case). In fact, this is not surprising, since the inhabitation problem for simply typed λ-calculus is long known to be PSPACE-complete [17], but nevertheless demonstrates the adequacy of the proposed models, to deal with the inhabitation problem.

A more recent line of research [1], introduces the notion of *pre-grammar*, which is a set of rules obtained from the structure of the type. From the pre-grammar of a given type one can easily address the standard problems of type-checking as well as the emptiness problem. Furthermore, from pre-grammars one can also address the emptiness problem using context free grammars (CFGs) and corresponding deterministic pushdown automata (DPA). The PDA implement acceptance by the empty stack, consequently the languages recognised by the grammars/automata are deterministic and prefix free. This work draws inspiration from the work by Schubert et al. [16] and the grammars proposed by Takahashi et al. [18], as well as the *Formula-Tree Method* by Broda et al. [6, 8].

The aim of this paper is to build on these previous approaches, namely the notion of pre-grammar and the inhabitation machines, to deal with the more complex (but closely related) problem of principality. The problem of principal inhabitation was dealt within the context of the *Formula-Tree Method* [6, 8], but no results were established regarding its complexity. The main purpose of the present work is to revise the methods presented by Schubert et al, in order to use the previous complexity results for the inhabitation machines, to establish complexity bounds for the principality problem.

The paper is structured as follows. In the next section we introduce some preliminary notions on λ-calculus, inhabitation machines and pre-grammars. In Section 3 we describe how to obtain an inhabitation machine from a pre-grammar of a given type, and prove that it accepts exactly the set of normal inhabitants of that type. This definition differs from that in [16] and leads to automata that operate in a deterministic way. Moreover, after removing a particular type of rules in the pre-grammars, the inhabitation machines are able to describe sets of long normal inhabitants, which is the more adequate framework when one wants to tackle principality. This, is done in Section 4. There, we supply the inhabitation machines with two additional global registers, that during the run of a term capture all elements of that run necessary to address principality of the term. We provide an algorithm to compute the principal type of the term from the information contained in the additional registers and prove the correctness of our methods. Finally, in Section 5, we draw some conclusions and highlight some future work.

2. Preliminaries

2.1. Simply Typed Lambda Calculus and Inhabitation Machines

In this paper we assume familiarity with basic results on the simply typed λ-calculus as described in [10] or [4]. Following [16] we consider the Church style approach. We denote type variables (atoms) by $a, b, c, \ldots$ and arbitrary types by lower-case Greek letters $\alpha, \beta, \gamma \ldots$. A (simple) type is either a type variable, or of the form $(\alpha \to \beta)$, where α and β are types. As usual, we assume associativity to the right, writing $\alpha_1 \to \alpha_2 \to \cdots \to \alpha_n$ instead of $(\alpha_1 \to (\alpha_2 \to (\cdots \to \alpha_n)))$. The set of simple types is denoted by $\mathcal{T}$. A *context* Γ is a finite set of *typed term variables* of the form x^α, where x ranges over an infinite countable set of term variables, denoted by $\mathcal{V}$, and $\alpha \in \mathcal{T}$. *Terms of type α in the context* Γ, written $\Lambda^\Gamma(\alpha)$, are a family of sets defined as the smallest family that satisfies the following conditions.

- $x^\alpha \in \Gamma$ implies $x^\alpha \in \Lambda^\Gamma(\alpha)$;
- if $M \in \Lambda^\Gamma(\alpha \to \beta)$ and $N \in \Lambda^\Gamma(\alpha)$, then $MN \in \Lambda^\Gamma(\beta)$;
- if $M \in \Lambda^{\Gamma \cup \{x^\alpha\}}(\beta)$, then $\lambda x^\alpha.M \in \Lambda^\Gamma(\alpha \to \beta)$.

In this paper we consider λ-terms modulo α-equivalence. As such, without loss of generality, we can assume terms to use different variable names in different abstractions, i.e. without bound variable clashes. If $\Gamma = \emptyset$, we usually abbreviate $\Lambda^\Gamma(\alpha)$ by $\Lambda(\alpha)$, which denotes the set of all (closed) inhabitants of α. When writing typed terms we sometimes omit some of the types in order to keep notation simple. As such, we might just write $\lambda x^{a\to b} y^a.xy$, $\lambda xy.x^{a\to b}y$ or even $\lambda xy.xy$, if convenient and when the typing is implicitly given. Following the notation in [10], we will on occasions write $\underline{o}$ when referring to a particular occurrence of an object o. Given a term M and an occurrence of a subterm $\underline{N}$ of M, let $\mathtt{sc}(\underline{N}, M)$ denote the set of typed variables x^τ such that $\underline{N}$ is in the scope of some abstraction λx^τ in M.

Occurrences of subtypes of a type α are called *negative (resp. positive)* as follows.

- α is a positive subtype of α;
- if $\alpha = \beta \to \gamma$, then every positive (resp. negative) occurrence of a subtype in β is a negative (resp. positive) occurrence in α; and every positive (resp. negative) occurrence of a subtype in γ is a positive (resp. negative) occurrence in α.

Note that every type α can be uniquely written as $\alpha = \alpha_1 \to \ldots \alpha_n \to a$, where a is a type-variable and $n \geq 0$. The type-variable a is called the *tail* of α. If $n \geq 1$, then $\alpha_1, \ldots, \alpha_n$ are called the *arguments* of α. An occurrence $\underline{\tau}$ in α is called a *negative subpremise* of α if and only if it is the argument of a positive occurrence of a subtype in α.

It is well-known that for every normal inhabitant M of a type α, every occurrence of x^τ in M corresponds to one particular occurrence of a subtype τ in α. The following lemma establishes the relationship between occurrences of variables in abstraction sequences and occurrences of subterms in a normal inhabitant M, respectively with negative subpremises and positive occurrences of subterms in its type, and can be easily proved by induction on the length of M.

Lemma 2.1 *Let M be a closed β-normal inhabitant of a type α. Then, the fact $M \in \Lambda(\alpha)$ can be obtained (in a unique way) applying the following rules a finite number of times.*

- *If $\underline{N} = x^\tau N_1 \cdots N_m$ is an occurrence of a subterm in M and $x^\tau \in \Gamma_N$, $N_1 \in \Lambda^{\Gamma_N}(\beta_1)$, $\ldots$, $N_m \in \Lambda^{\Gamma_N}(\beta_m)$, for $m \geq 0$, $\tau = \beta_1 \to \cdots \to \beta_m \to \beta$, and $\Gamma_N = \mathtt{sc}(\underline{N}, M)$, then $N \in \Lambda^{\Gamma_N}(\beta)$. Furthermore, τ is a negative subpremise and each β_i (resp. β) in this subpremise is a positive (resp. negative) occurrence of a subtype in α.*
- *If $\underline{N} = \lambda x^{\beta_1}.N_1$ is an occurrence of a subterm in M and $N_1 \in \Lambda^{\Gamma_N \cup \{x^{\beta_1}\}}(\beta_2)$, for $\Gamma_N = \mathtt{sc}(\underline{N}, M)$, then $N \in \Lambda^{\Gamma_N}(\beta_1 \to \beta_2)$. Also, there is a positive occurrence of $\beta_1 \to \beta_2$ with α, with negative subpremise β_1.*

This unique derivation of $M \in \Lambda(\alpha)$ can be represented in the usual way as a tree with root $M \in \Lambda(\alpha)$, and we will refer to it as the *minimal derivation tree of $M \in \Lambda(\alpha)$*.

A β-normal inhabitant of a type α is called a *long normal inhabitant* iff every variable occurrence x^τ, where $\tau = \tau_1 \to \cdots \to \tau_n \to a$, with $n \geq 0$, is followed by the longest sequence of arguments allowed by its type, i.e. has exactly n arguments. The finite set of 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 [5] that the η-families of the long normal inhabitants of α partition the set of normal inhabitants of α into non-overlapping finite subsets, each η-family containing just one long member. Thus, when searching for normal inhabitants one might focus on the set of long normal inhabitants from which all normal inhabitants can be obtained by η-reduction.

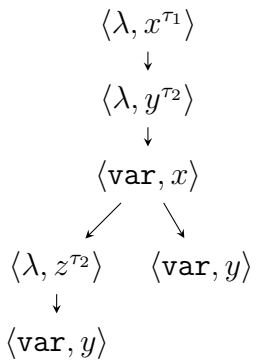
Example 2.2 *Consider type $\alpha = ((o \to o) \to o \to o) \to o \to o$ which will be our running example throughout this paper. Normal inhabitants of α are, for instance, $M_1 = \lambda x^{\tau_1} y^{\tau_2}.x(\lambda z^{\tau_2}.y)y$ and $M_2 = \lambda x^{\tau_1}.x(\lambda y^{\tau_2}.y)$, $M_3 = \lambda x^{\tau_1} y^{\tau_2}.x(\lambda z^{\tau_2}.z)(x(\lambda u^{\tau_2}.y)y)$, and $M_4 = \lambda x^{\tau_1} y^{\tau_2}.y$, where $\tau_1 = (o \to o) \to o \to o$ and $\tau_2 = o$. Note that M_2 is not a long inhabitant, since variable $x^{(o\to o)\to o\to o}$ does not have sufficient (two) arguments.*

Following [16], we also use a tree representation for terms, which is the format in which terms will be processed by our automata/inhabitation machines. However, this representation is restricted to β-normal terms and differs from that in [16], allowing us thereby to avoid the notion of spine in our approach. As a consequence runs in our inhabitation machines will be deterministic.

Definition 2.3 *Given a β-normal term M, the tree $\mathbf{t^M}$ is defined inductively by the following.*

- *The tree of $M = \lambda x^\tau.N$ has root node $\langle \lambda, x^\tau \rangle$ with a unique subtree $\mathbf{t^N}$.*
- *The tree of $M = xN_1 \ldots N_k$ has root node $\langle \mathtt{var}, x^\tau \rangle$ with $k \geq 0$ subtrees $\mathbf{t^{N_1}}, \ldots, \mathbf{t^{N_k}}$.*

Example 2.4 *For M_1 from Example 2.2 the tree $\mathbf{t_{M_1}}$ is depicted below.*



Definition 2.5 *A (single assignment)* inhabitation machine $\mathcal{A}$ *is a tuple* $\langle \Sigma, N, Q, q_I, \mathcal{R}, \delta \rangle$, *where* $\Sigma = \{\lambda, \mathtt{var}\}$ *is a finite signature,* N *is an infinite set of data elements,* Q *is a finite set of states,* $q_I \in Q$ *is the initial state,* $\mathcal{R}$ *is a finite set of register names, and* $\delta \subseteq \Sigma \times Q \times (\mathcal{R} \cup \{\varnothing\}) \times \vec{Q} \times (\mathcal{R} \cup \{\varnothing\})$ *is the transition relation containing rules written as* $a, q, \mathtt{r} \rightsquigarrow q_1, \ldots, q_n, \mathtt{r}'$ *with* $n \geq 0$, $a \in \Sigma$ *and where* $\vec{Q}$ *denotes the set of finite (possibly empty) sequences of elements in* Q.

The machine traverses tree representations of λ-terms in normal form, as defined above. There is a set $\mathtt{Reg}$ containing a register for every register name in $\mathcal{R}$. During a traversal variable names (elements of N) can be stored in the registers. We denote particular constellations of the set of registers $\mathtt{Reg}$ by $\mathtt{R}_1, \mathtt{R}_2$, etc. To access the elements stored in registers, there exists a function $\mathtt{cont}$ that given a particular constellation $\mathtt{R}$ of $\mathtt{Reg}$ and register name $\mathtt{r}$ returns the content $\mathtt{cont}(\mathtt{R}, \mathtt{r})$ of register $\mathtt{r}$ in $\mathtt{R}$. A configuration of $\mathcal{A}$ in a tree $\mathbf{t}$ is a (finite) sequence of tuples of the form $\mathtt{c}_i = (\mathbf{t}_i, q_i, \mathtt{R}_i)$, where $\mathbf{t}_i$ is a subtree of $\mathbf{t}$, $q_i \in Q$ and $\mathtt{R}_i$ is a particular constellation of $\mathtt{Reg}$.

The operational semantics for such a machine $\mathcal{A}$ and term $\mathbf{t}$ is as follows. The initial configuration is $\mathtt{c} = (\mathbf{t}, q_I, \mathtt{R}_\varnothing)$, where all registers in $\mathtt{R}_\varnothing$ are empty, i.e. $\mathtt{cont}(\mathtt{R}_\varnothing, \mathtt{r}) = \emptyset$ for all $\mathtt{r} \in \mathcal{R}$. A non-empty sequence $\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_n$ with $\mathtt{c}_1 = (\mathbf{t}_1, q_i, \mathtt{R}_1)$ transitions to sequence $\mathtt{c}^1, \ldots, \mathtt{c}^k, \mathtt{c}_2, \ldots, \mathtt{c}_n$ if one of the following applies:

- $\mathbf{t}_1$ has root $\langle \lambda, x \rangle$ and a single subtree $\mathbf{t}'$, there is some rule $\lambda, q_i, \varnothing \rightsquigarrow q^1, \mathtt{r}$ in δ, $k = 1$, and $\mathtt{c}^1 = (\mathbf{t}', q^1, \mathtt{R}')$, where $\mathtt{R}'$ is obtained from $\mathtt{R}_1$ by updating register $\mathtt{r}$ with the addition of variable x;
- $\mathbf{t}_1$ has root $\langle \mathtt{var}, x \rangle$ and $k \geq 0$ subtrees $\mathbf{t}^1, \ldots, \mathbf{t}^k$, there is some rule $\mathtt{var}, q_i, \mathtt{r} \rightsquigarrow q^1, \ldots, q^k, \varnothing$ in δ, $x \in \mathtt{cont}(\mathtt{R}_1, \mathtt{r})$, and for $1 \leq j \leq k$ one has $\mathtt{c}^j = (\mathbf{t}^j, q^j, \mathtt{R}_1)$.

The empty configuration sequence represents success. We say that $\mathcal{A}$ accepts a tree $\mathbf{t}$ if there is a run from the initial configuration (sequence) $\mathtt{c} = (\mathbf{t}, q_I, \mathtt{R}_\varnothing)$ to the empty sequence. The set of normal terms M such that $\mathcal{A}$ accepts $\mathbf{t}^M$ is denoted by $\mathcal{L}(\mathcal{A})$.

2.2. Pre-grammars

In this section we describe how to obtain for a type α a set of rewriting rules, which we call the *pre-grammar* of α, and denote by $\mathtt{pre}(\alpha)$. This device was first presented in [1] and has been shown useful for addressing different kind of problems related to type inhabitation. In fact, [1] also contains a simpler variant of a pre-grammar, based on subtypes instead of occurrences of subtypes, which would be sufficient to deal with inhabitation here and actually produces smaller inhabitation machines. Nevertheless, in Section 4 we need this more elaborate variant of a pre-grammar in order to deal with principal inhabitants.

We start by associating to each type α a set $\mathtt{OccT}(\alpha)$ that contains for each occurrence $\underline{\beta}$ of a subtype β a tuple (β, n, l), where $n \in \mathbb{N}$, and $l \in \{\mathtt{var}\} \cup \{\ n \to m \mid n, m \in \mathbb{N}\ \}$. Distinct occurrences of the same subtype are assigned distinct tuples. This set is uniquely defined, up to isomorphism between integers n used in the tuples.

Definition 2.6 *Given a type $\alpha \in \mathcal{T}$ let $\mathtt{OccT}(\alpha)$ be the smallest set satisfying the following.*

- *For each occurrence of a type variable a in α there is a tuple $(a, n, \mathtt{var}) \in \mathtt{OccT}(\alpha)$;*
- *if $\beta \to \gamma$ is an occurrence of a subtype of α, and $(\beta, n, l_\beta), (\gamma, m, l_\gamma) \in \mathtt{OccT}(\alpha)$ are the tuples corresponding to β and γ in this occurrence, then $(\beta \to \gamma, k, n \to m) \in \mathtt{OccT}(\alpha)$;*
- *for each $n \in \mathbb{N}$ there is at most one tuple $(\beta, n, l) \in \mathtt{OccT}(\alpha)$.*

Furthermore, given a particular occurrence of a subtype β of α we denote by $\mathtt{n}(\beta)$ the unique integer n such that $(\beta, n, l) \in \mathtt{OccT}(\alpha)$. We frequently will refer to $\mathtt{n}(\beta)$ as the identifier *of β w.r.t. $\mathtt{OccT}(\alpha)$. Finally, $\mathtt{t}(n) = \beta$, $\mathtt{lab}(n) = l$, and $\mathtt{N}(\alpha) = \{\ n \mid (\beta, n, l) \in \mathtt{OccT}(\alpha)\ \}$.*

In order to deal correctly with the correspondence between occurrences of subtypes and occurrences of subterms, polarities have to be taken into account. With this purpose, and whenever convenient, we might superscript an integer n with $+$ if n corresponds to a positive occurrence of a subtype, i.e. an occurrence that can be the type of a subterm of an inhabitant, and with $-$ if it corresponds to a negative subpremise of α, i.e. if it corresponds to an occurrence that can be the type of a variable in an abstraction sequence. Integers that correspond to a negative occurrence, which is no subpremise, will not be superscripted.

We say that two integers $n, m \in \mathtt{N}(\alpha)$ are equivalent w.r.t. $\mathtt{OccT}(\alpha)$, and write $n \equiv_{\mathtt{OccT}} m$, if and only if they correspond to the same subtype.

Definition 2.7 *Given a type α and the associated set $\mathtt{OccT}(\alpha)$, the binary relation $T(\alpha) \subseteq \mathtt{N}(\alpha) \times \mathtt{N}(\alpha)$ is defined by $(k, n) \in T(\alpha)$ iff $(\beta, n, m \to k) \in \mathtt{OccT}(\alpha)$, for some occurrence of a subtype β, i.e., $\beta = \beta_1 \to \beta_2$, $\mathtt{lab}(\beta_1) = m$ and $\mathtt{lab}(\beta_2) = k$. Furthermore, for $(k, n) \in T(\alpha)$ let $\mathtt{q}(k, n) = m$.*

Lemma 2.8 *If α contains p occurrences of type variables $a_1, \ldots, a_p$, then the graph of $T(\alpha)$, whose set of nodes correspond to $\mathtt{N}(\alpha)$, consists of p unary trees with roots $\mathtt{lab}(a_1), \ldots, \mathtt{lab}(a_p)$, respectively.*

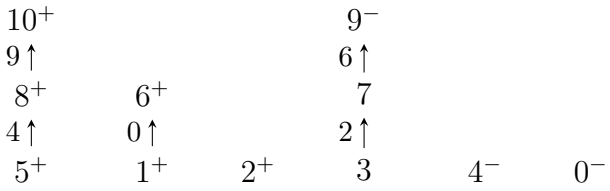
Example 2.9 *For* $\alpha = ((o \to o) \to o \to o) \to o \to o$ *from Example 2.2 the set* $\texttt{OccT}(\alpha)$ *contains eleven tuples* (β, n, l)*, where* β*,* n *and* l *are given below.*

β	n	l
o	0	var
o	1	var
o	2	var
o	3	var

β	n	l
o	4	var
o	5	var
$o \to o$	6	$0 \to 1$
$o \to o$	7	$2 \to 3$

β	n	l
$o \to o$	8	$4 \to 5$
$(o \to o) \to o \to o$	9	$6 \to 7$
α	10	$9 \to 8$

The equivalence relation $\equiv_{\texttt{OccT}}$ *partitions* $\texttt{N}(\alpha)$ *into four equivalence classes, which are* $\{10^+\}$*,* $\{9^-\}$*,* $\{6^+, 7, 8^+\}$*, and* $\{0^-, 1^+, 2^+, 3, 4^-, 5^+\}$*. The associated graph* $T(\alpha)$ *is depicted below.*



Now, $\texttt{pre}(\alpha)$ can be computed from $\texttt{OccT}(\alpha)$ and $T(\alpha)$ as follows.

Definition 2.10 *Given a type* α *and a set of tuples* $\texttt{OccT}(\alpha)$*, we denote by* $\texttt{pre}(\alpha)$ *the smallest set of rules satisfying the following conditions.*

- *If* $m^-, n^+ \in \texttt{N}(\alpha)$ *and* $n^+ \equiv_{\texttt{OccT}} m^-$*, then* $n := m \in \texttt{pre}(\alpha)$*;*
- *if* $m^-, k^+, n^+ \in \texttt{N}(\alpha)$ *and* $(\beta, n, m \to k) \in \texttt{OccT}(\alpha)$*, then* $n := \lambda m.k \in \texttt{pre}(\alpha)$*;*
- *if* $m_s^-, n^+ \in \texttt{N}(\alpha)$ *and* $(m_1, m_2), (m_2, m_3), \ldots, (m_{s-1}, m_s) \in T(\alpha)$*, for some* $s \geq 2$*,* $n^+ \equiv_{\texttt{OccT}} m_1$*, and* $\texttt{q}(m_i, m_{i+1}) = q_i$ *for* $1 \leq i \leq s-1$*, then* $n := m_s\ q_{s-1} \cdots q_1 \in \texttt{pre}(\alpha)$*.*

Example 2.11 *From the sets* $\texttt{OccT}(\alpha)$ *and* $T(\alpha)$ *from Example 2.9 we obtain the following set of rules* $\texttt{pre}(\alpha)$*.*

$$
\begin{array}{lll}
10 := \lambda 9.8 & 6 := \lambda 0.1 \mid 9\ 6 & 2 := 9\ 6\ 2 \mid 4 \mid 0 \\
8 := \lambda 4.5 \mid 9\ 6 & 5 := 9\ 6\ 2 \mid 4 \mid 0 & 1 := 9\ 6\ 2 \mid 4 \mid 0
\end{array}
$$

3. Inhabitation

In this section, given type α, we define how to obtain an inhabitation machine $\mathcal{A}_\alpha$ from $\texttt{pre}(\alpha)$, such that $\mathcal{L}(\mathcal{A}_\alpha)$ is the set of closed normal inhabitants of α.

Definition 3.1 *Given a type* α *let* $A_\alpha = \langle \Sigma, N, Q, q_I, \mathcal{R}, \delta \rangle$*, where* $\Sigma = \{\lambda, \texttt{var}\}$*,* $N = \mathcal{V}$*,* $Q = \{\ q_n \mid n^+ \in \texttt{N}(\alpha)\ \}$*,* $q_I = q_{\texttt{n}(\alpha)}$*,* $\mathcal{R} = \{\ \texttt{r}_n \mid n^- \in \texttt{N}(\alpha)\ \}$*, and* δ *is defined as follows:*

- *if* $n := \lambda m.k \in \texttt{pre}(\alpha)$*, then* $\lambda, q_n, \varnothing \rightsquigarrow q_k, \texttt{r}_m\ \in \delta$*;*
- *if* $n := m\ i_1 \cdots i_k \in \texttt{pre}(\alpha)$*, then* $\texttt{var}, q_n, \texttt{r}_m \rightsquigarrow q_{i_1}, \ldots, q_{i_k}, \varnothing\ \in \delta$*, where* $k \geq 0$*.*

Example 3.2 *The inhabitation machine for α from Example 2.2 is $A_\alpha = \langle\{\lambda, \mathtt{var}\}, \mathcal{V}, Q, q_{10}, \mathcal{R}, \delta\rangle$, where $Q = \{q_1, q_2, q_5, q_6, q_8, q_{10}\}$, $\mathcal{R} = \{\mathtt{r}_9, \mathtt{r}_4, \mathtt{r}_0\}$ and δ contains the rules below.*

$$
\begin{array}{lll}
\lambda, q_{10}, \varnothing \rightsquigarrow q_8, \mathtt{r}_9 & \mathtt{var}, q_5, \mathtt{r}_9 \rightsquigarrow q_6, q_2, \varnothing & \mathtt{var}, q_2, \mathtt{r}_4 \rightsquigarrow \varnothing \\
\lambda, q_8, \varnothing \rightsquigarrow q_5, \mathtt{r}_4 & \mathtt{var}, q_2, \mathtt{r}_9 \rightsquigarrow q_6, q_2, \varnothing & \mathtt{var}, q_2, \mathtt{r}_0 \rightsquigarrow \varnothing \\
\lambda, q_6, \varnothing \rightsquigarrow q_1, \mathtt{r}_0 & \mathtt{var}, q_1, \mathtt{r}_9 \rightsquigarrow q_6, q_2, \varnothing & \mathtt{var}, q_1, \mathtt{r}_4 \rightsquigarrow \varnothing \\
\mathtt{var}, q_8, \mathtt{r}_9 \rightsquigarrow q_6, \varnothing & \mathtt{var}, q_5, \mathtt{r}_4 \rightsquigarrow \varnothing & \mathtt{var}, q_1, \mathtt{r}_0 \rightsquigarrow \varnothing \\
\mathtt{var}, q_6, \mathtt{r}_9 \rightsquigarrow q_6, \varnothing & \mathtt{var}, q_5, \mathtt{r}_0 \rightsquigarrow \varnothing &
\end{array}
$$

Now, consider $M = \lambda xy.x(\lambda z.y)y$, whose tree representation $\mathbf{t}^M$ is isomorphic to the one in Example 2.4. Below we depict the unique run of $\mathbf{t}^M$ on $\mathcal{A}_\alpha$. We represent a change from one configuration sequence to another by $\rightsquigarrow^{\langle a,x\rangle}$, where $\langle a, x\rangle$ is the root of the tree that is processed (remember that by definition we always process the leftmost tuple in the configuration sequence first). In this way, it becomes implicit which tree pertains to each of the tuples, and we may omit them to shorten the presentation. The contents of the registers are displayed by sets indexed with register names. The run starts with the initial configuration $(\mathbf{t}^M, q_{10}, [\emptyset_9, \emptyset_4, \emptyset_0])$.

$$
\begin{array}{l}
(q_{10}, [\emptyset_9, \emptyset_4, \emptyset_0]) \rightsquigarrow^{\langle\lambda,x\rangle} (q_8, [\{x\}_9, \emptyset_4, \emptyset_0]) \rightsquigarrow^{\langle\lambda,y\rangle} (q_5, [\{x\}_9, \{y\}_4, \emptyset_0]) \rightsquigarrow^{\langle\mathtt{var},x\rangle} \\
(q_6, [\{x\}_9, \{y\}_4, \emptyset_0]), (q_2, [\{x\}_9, \{y\}_4, \emptyset_0]) \rightsquigarrow^{\langle\lambda,z\rangle} (q_1, [\{x\}_9, \{y\}_4, \{z\}_0]), \\
(q_2, [\{x\}_9, \{y\}_4, \emptyset_0]) \rightsquigarrow^{\langle\mathtt{var},y\rangle} (q_2, [\{x\}_9, \{y\}_4, \emptyset_0]) \rightsquigarrow^{\langle\mathtt{var},y\rangle} \varepsilon.
\end{array}
$$

Proposition 3.3 *Given $\alpha \in \mathcal{T}$ and a normal term M, the inhabitation machine $\mathcal{A}_\alpha$ operates in a deterministic way on $\mathbf{t}^M$.*

Proof. We will show that for any configuration $\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_n$, with $\mathtt{c}_1 = (\mathbf{t}_1, q_i, \mathtt{R}_1)$, reached from the initial configuration $\mathtt{c} = (\mathbf{t}^M, q_{\mathtt{n}(\alpha)}, \mathtt{R}_\varnothing)$, there is at most one transition rule of $\mathcal{A}_\alpha$ that applies.

If the root of $\mathbf{t}_1$ is of the form $\langle\lambda, x\rangle$, then only transition rules of the form $\lambda, q_i, \varnothing \rightsquigarrow q_k, \mathtt{r}_m$ apply. By definition, given $i \in \mathtt{N}(\alpha)$, there can be at most one tuple $(\beta, i, m \to k) \in \mathtt{OccT}(\alpha)$ and consequently at most one rule $i := \lambda m.k \in \mathtt{pre}(\alpha)$, as well as at most one transition rule of the form above in δ.

If the root of $\mathbf{t}_1$ is of the form $\langle\mathtt{var}, x\rangle$, then only transition rules of the form $\mathtt{var}, q_i, \mathtt{r}_m \rightsquigarrow q_{i_1}, \ldots, q_{i_k}, \varnothing$ apply. There is a rule of this form if and only if there exist $(m_k, m_{k-1}), \ldots, (m_1, m_0) \in T(\alpha)$ with $m_0 = m$, $m_k \equiv_{\mathtt{OccT}} i^+$ and for $1 \leq j \leq k$, $\mathtt{q}(mj, m_{j-1}) = i_j$. Suppose that there is another different transition rule of the form $\mathtt{var}, q_i, \mathtt{r}_{m'} \rightsquigarrow q_{j_1}, \ldots, q_{j_l}, \varnothing$. Consequently, we have the same conditions for this rule, in particular the condition $m'_l \equiv_{occT} i^+$. But, in each (unary) tree in the graph of $T(\alpha)$ there is at most one node m'' such that $m'' \equiv_{occT} i^+$. Thus, m_k and m'_l must occur in two different trees and consequently also m and m', which implies that $m \neq m'$. Furthermore, M is supposed to have no bound variable clashes. Thus, variable x can be present in at most one of the registers $\mathtt{R}_m$ or $\mathtt{R}_{m'}$. Consequently, by the definition of the operational semantics of inhabitation machines, at most one of these transition rules can apply. □

Theorem 3.4 *The language $\mathcal{L}(\mathcal{A}_\alpha)$ is the set of closed normal inhabitants of α.*

Proof. For the first part of the proof consider a closed normal inhabitant M of α, any occurrence $\underline{N}$ of a subterm in M and the corresponding node $N \in \Lambda^{\Gamma_N}(\beta)$ in the minimal derivation tree of $M \in \Lambda(\alpha)$, where $\Gamma_N = \mathtt{sc}(\underline{N}, M)$. Let $\mathtt{R}_N$ be the constellation of $\mathtt{Reg}$ such that for every occurrence of a subtype σ, register $r_{\mathtt{n}(\sigma)}$ contains exactly the variables in Γ_N corresponding to that occurrence of σ. We will show, by structural induction on N that $(\mathbf{t}^N, q_{\mathtt{n}(\sigma)}, \mathtt{R}_N) \rightsquigarrow^* \varepsilon$, where $\rightsquigarrow^*$ denotes the reflexive, transitive closure of $\rightsquigarrow$. Then, $(\mathbf{t}^M, q_{\mathtt{n}(\alpha)}, \mathtt{R}_\varnothing) \rightsquigarrow^* \varepsilon$ and $M \in \mathcal{L}(\mathcal{A}_\alpha)$.

- Suppose that $\underline{N} = x^\tau N_1 \cdots N_m \in \Lambda^{\Gamma_N}(\beta)$, $x^\tau \in \Gamma_N$, $N_j \in \Lambda^{\Gamma_N}(\beta_j)$ for $1 \leq j \leq m$, $m \geq 0$ and $\tau = \beta_1 \to \cdots \to \beta_m \to \beta$. Since τ is an occurrence of a subtype of α, there is a rule $\mathtt{n}(\beta) := \mathtt{n}(\tau)\ \mathtt{n}(\beta_1) \cdots \mathtt{n}(\beta_m) \in \mathtt{pre}(\alpha)$ and consequently a transition rule $\mathtt{var}, q_{\mathtt{n}(\beta)}, \mathtt{r}_{\mathtt{n}(\tau)} \rightsquigarrow q_{\mathtt{n}(\beta_1)}, \ldots, q_{\mathtt{n}(\beta_m)}, \varnothing \in \delta$ in $\mathcal{A}_\alpha$. Since $x^\tau \in \Gamma_n$, we have $x \in \mathtt{cont}(\mathtt{R}_N, \mathtt{r}_{\mathtt{n}(\tau)})$. Thus, the transition rule applies and $(\mathbf{t}^N, q_{\mathtt{n}(\beta)}, \mathtt{R}_N) \rightsquigarrow (\mathbf{t}^{N_1}, q_{\mathtt{n}(\beta_1)}, \mathtt{R}_N), \ldots, (\mathbf{t}^{N_m}, q_{\mathtt{n}(\beta_m)}, \mathtt{R}_N)$. Now, the result follows from the induction hypothesis.
- Now, suppose that $\underline{N} = \lambda x^{\beta_1}.N_1 \in \Lambda^{\Gamma_N}(\beta)$, with $\beta = \beta_1 \to \beta_2$ and $N_1 \in \Lambda^{\Gamma_N \cup \{x^{\beta_1}\}}(\beta_2)$. Then, there is a rule $\mathtt{n}(\beta) := \lambda \mathtt{n}(\beta_1).\mathtt{n}(\beta_2) \in \mathtt{pre}(\alpha)$ and consequently a transition rule $\lambda, q_{\mathtt{n}(\beta)}, \varnothing \rightsquigarrow q_{\mathtt{n}(\beta_2)}, \mathtt{r}_{\mathtt{n}(\beta_1)} \in \delta$. Thus, $(\mathbf{t}^N, q_{\mathtt{n}(\beta)}, \mathtt{R}_N) \rightsquigarrow (\mathbf{t}^{N_1}, q_{\mathtt{n}(\beta_2)}, \mathtt{R}_{N_1})$, where $\mathtt{R}_{N_1}$ is obtained from $\mathtt{R}_N$ by updating register $\mathtt{r}_{\mathtt{n}(\beta_1)}$ with the addition of variable x. Again, the result follows from the induction hypothesis.

For the second part of the proof we show that given a normal term M and $i \in \mathtt{N}(\alpha)$, if the tuple $(\mathbf{t}^M, q_i, \mathtt{R}) \rightsquigarrow^* \varepsilon$, then $M \in \Lambda^{\Gamma_\mathtt{R}}(\mathtt{t}(i))$, where $\Gamma_\mathtt{R} = \{\ x^{\mathtt{t}(n)} \mid n \in \mathtt{N}(\alpha), x \in \mathtt{cont}(\mathtt{R}, \mathtt{r}_n)\ \}$. Then for $M \in \mathcal{L}(\mathcal{A}_\alpha)$, $(\mathbf{t}^M, q_{\mathtt{n}(\alpha)}, \mathtt{R}_\varnothing) \rightsquigarrow^* \varepsilon$ implies that $M \in \Lambda(\alpha)$. This part of the proof is by induction on the length of the transition sequence from $(\mathbf{t}^M, q_i, \mathtt{R})$ to ε.

- Suppose that $\mathbf{t}^M$ has root $\langle \lambda, x \rangle$ and a single subtree $\mathbf{t}^N$, and that the first transition is by some rule $\lambda, q_i, \varnothing \rightsquigarrow q_j, \mathtt{r}_k$ to $(\mathbf{t}^N, q_j, \mathtt{R}_k)$, where $\mathtt{R}_k$ has been obtained from $\mathtt{R}$ updating register $\mathtt{r}_k$ by adding variable x. Then, $i := \lambda k.j \in \mathtt{pre}(\alpha)$ and $\mathtt{t}(i) = \mathtt{t}(k) \to \mathtt{t}(j)$. By the induction hypothesis $N \in \Lambda^{\Gamma_\mathtt{R} \cup \{x^{\mathtt{t}(k)}\}}(\mathtt{t}(j))$. It follows that $M \in \Lambda^{\Gamma_\mathtt{R}}(\mathtt{t}(i))$.
- Finally, suppose that $\mathbf{t}^M$ has root $\langle \mathtt{var}, x \rangle$ and $k \geq 0$ subtrees $\mathbf{t}^{N_1}, \ldots, \mathbf{t}^{N_k}$, and that the first transition is by some rule $\mathtt{var}, q_i, \mathtt{r}_m \rightsquigarrow q_{n_1}, \ldots, q_{n_k}, \varnothing \in \delta$, such that $x \in \mathtt{cont}(\mathtt{R}, \mathtt{r}_m)$ to $(\mathbf{t}^{N_1}, q_{n_1}, \mathtt{R}), \ldots, (\mathbf{t}^{N_k}, q_{n_k}, \mathtt{R})$. Then, there is a rule $i := m\ n_1 \cdots n_k \in \mathtt{pre}(\alpha)$, $\mathtt{t}(m) = \mathtt{t}(n_1) \to \cdots \mathtt{t}(n_k) \to \mathtt{t}(i)$ and $x^{\mathtt{t}(m)} \in \Gamma_\mathtt{R}$. By the induction hypothesis, $N_j \in \Lambda^{\Gamma_\mathtt{R}}(\mathtt{t}(n_j))$ for $1 \leq j \leq k$. Thus, $M \in \Lambda^{\Gamma_\mathtt{R}}(\mathtt{t}(i))$.

□

4. Principal Inhabitants

In this section we address the more complex problem of principal inhabitants. The methods we present here are based on the syntactic characterisation of principal proof-trees given in [6, 8]. Following that approach, and since every principal normal inhabitant expands to a unique long normal inhabitant that is also principal, we will from now on focus on long normal inhabitants.

Definition 4.1 *An inhabitant $M \in \Lambda(\alpha)$ is called a* principal inhabitant *of α, if for every type τ such that $M' \in \Lambda(\tau)$ and M' is structurally identical to M (that is, M' differs from M in the types that annotate variables), then τ is an instance of α, i.e. can be obtained from α by some type substitution. In this case, α is called the* principal type *of M. A* principal long normal inhabitant *of α is a principal inhabitant in long normal form.*

It has been shown in [1] how to change the definition of $\texttt{pre}(\alpha)$ in order to apply exactly to the set of long normal inhabitants. For this, it is sufficient to drop in $\texttt{pre}(\alpha)$ all rules of the form $n := m_s\ q_{s-1} \cdots q_1$ such that $\texttt{lab}(n) \neq \texttt{var}$. The pre-grammar thereby obtained is denoted by $\texttt{preL}(\alpha)$.

Example 4.2 *The pre-grammar* $\texttt{preL}(\alpha)$ *for the set long normal inhabitants of α from Example 2.2 is the following.*

$$\begin{array}{lll} 10 := \lambda 9.8 & 6 := \lambda 0.1 & 2 := 9\ 6\ 2 \mid 4 \mid 0 \\ 8 := \lambda 4.5 & 5 := 9\ 6\ 2 \mid 4 \mid 0 & 1 := 9\ 6\ 2 \mid 4 \mid 0 \end{array}$$

The approach given in [6, 8] establishes that, in the beginning all occurrences of type variables in α have to be made different. Here, this is already achieved by the association of different identifiers to different occurrences of subtypes in $\texttt{OccT}(\alpha)$. In order to keep track of the fulfilment of the conditions in the characterisation of principal proof-trees in [6, 8], we supply our inhabitation machines with two additional global registers. We will also need to refer to the identifier of the tail variable of an occurrence of a subtype with identifier n, which will be denoted by $\texttt{tail}(n)$. Note that $\texttt{tail}(n)$ is the root of the (unary) tree in graph $T(\alpha)$, that contains n.

Example 4.3 *For instance,* $\texttt{tail}(10) = \texttt{tail}(8) = \texttt{tail}(5) = 5$ *in our running example.*

Definition 4.4 *A* principal inhabitation machine $\mathcal{P}$ *is a tuple* $\langle \Sigma, N, Q, q_I, \mathcal{R}, \texttt{id}, \texttt{needs}, \delta \rangle$, *where $\Sigma = \{\lambda, \texttt{var}\}$ is a finite signature, N is an infinite set of data elements, Q is a finite set of states, $q_I \in Q$ is the initial state, $\mathcal{R}$ is a finite set of register names,* id *and* needs *are two seperate registers, and $\delta \subseteq \Sigma \times Q \times (\mathcal{R} \cup \{\varnothing\}) \times \vec{Q} \times (\mathcal{R} \cup \{\varnothing\})$ is the transition relation containing rules written as $a, q, \mathtt{r} \rightsquigarrow q_1, \ldots, q_n, \mathtt{r}'$ with $n \geq 0$, $a \in \Sigma$ and where $\vec{Q}$ denotes the set of finite (possibly empty) sequences of elements in Q.*

The machine traverses tree representations of λ-terms in a similarly way as above, updating in every transition step the global registers id and needs. Now, the configuration of $\mathcal{P}$ in a tree $\mathbf{t}$ is a tuple of arity 3, containing:

- a (finite) sequence of tuples of the form $\mathtt{c}_i = (\mathbf{t}_i, q_i, \mathtt{R}_i)$, where $\mathbf{t}_i$ is a subtree of $\mathbf{t}$, $q_i \in Q$ and $\mathtt{R}_i$ is a particular constellation of Reg;
- register id;
- and register needs.

The operational semantics for such a machine $\mathcal{P}$ and term $\mathbf{t}$ is as follows. The initial configuration is $((\mathbf{t}, q_I, \mathtt{R}_\varnothing); \texttt{id}; \texttt{needs})$, where all registers in $\mathtt{R}_\varnothing$ are empty, i.e. $\texttt{cont}(\mathtt{R}_\varnothing, \mathtt{r}) = \emptyset$

for all $\mathtt{r} \in \mathcal{R}$. A configuration $(\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_n; \mathtt{id}; \mathtt{needs})$ with $\mathtt{c}_1 = (\mathbf{t}_1, q_i, \mathtt{R}_1)$ transitions to configuration $(\mathtt{c}^1, \ldots, \mathtt{c}^k, \mathtt{c}_2, \ldots, \mathtt{c}_n; \mathtt{id}'; \mathtt{needs}')$ if one of the following applies:

- $\mathbf{t}_1$ has root $\langle \lambda, x \rangle$ and a single subtree $\mathbf{t}'$, there is some rule $\lambda, q_i, \varnothing \rightsquigarrow q^1, \mathtt{r}$ in δ, $k = 1$, and $\mathtt{c}^1 = (\mathbf{t}', q^1, \mathtt{R}')$, where $\mathtt{R}'$ is obtained from $\mathtt{R}_1$ by updating register $\mathtt{r}$ with the addition of variable x, $\mathtt{id}' = \mathtt{id}$ and $\mathtt{needs}' = \mathtt{needs}$;
- $\mathbf{t}_1$ has root $\langle \mathtt{var}, x \rangle$ and $k \geq 0$ subtrees $\mathbf{t}^1, \ldots, \mathbf{t}^k$, there is some rule $\mathtt{var}, q_i, \mathtt{r}_n \rightsquigarrow q^1, \ldots, q^k, \varnothing$ in δ, $x \in \mathtt{cont}(\mathtt{R}_1, \mathtt{r}_n)$, and for $1 \leq j \leq k$ one has $\mathtt{c}^j = (\mathbf{t}^j, q^j, \mathtt{R}_1)$. Here $\mathtt{id}'$ is obtained from $\mathtt{id}$ by adding the identification $i = \mathtt{tail}(n)$, and $\mathtt{needs}' = \mathtt{needs} \setminus \{n\}$.

We say that $\mathcal{P}$ accepts a tree $\mathbf{t}$ with output $(\mathtt{id}', \mathtt{needs}')$ if there is a run from the initial configuration $((\mathbf{t}, q_I, \mathtt{R}_\varnothing); \mathtt{id}; \mathtt{needs})$ to configuration $(\varepsilon; \mathtt{id}'; \mathtt{needs}')$. The set of normal terms M such that $\mathcal{P}$ accepts $\mathbf{t}^M$ is denoted by $\mathcal{P}(\mathcal{A})$.

With this modifications, we can now define the principal inhabitation machine $\mathcal{P}_\alpha$ for a type α.

Definition 4.5 *Given a type α let $P_\alpha = \langle \Sigma, N, Q, q_I, \mathcal{R}, \mathtt{id}, \mathtt{needs}, \delta \rangle$, where $\Sigma = \{\lambda, \mathtt{var}\}$, $N = \mathcal{V}$, $Q = \{\, q_n \mid n^+ \in \mathtt{N}(\alpha) \,\}$, $q_I = q_{\mathtt{n}(\alpha)}$, $\mathcal{R} = \{\, \mathtt{r}_n \mid n^- \in \mathtt{N}(\alpha) \,\}$, $\mathtt{id} = \emptyset$, $\mathtt{needs} = \{\, n \mid n^- \in \mathtt{N}(\alpha), \mathtt{lab}(n) \neq \mathtt{var} \,\}$, and δ is defined as follows:*

- *if $n := \lambda m.k \in \mathtt{preL}(\alpha)$, then $\lambda, q_n, \varnothing \rightsquigarrow q_k, \mathtt{r}_m \in \delta$;*
- *if $n := m\ i_1 \cdots i_k \in \mathtt{preL}(\alpha)$, then $\mathtt{var}, q_n, \mathtt{r}_m \rightsquigarrow q_{i_1}, \ldots, q_{i_k}, \varnothing \in \delta$, where $k \geq 0$.*

Example 4.6 *The transition rules of the principal inhabitation machine $\mathcal{P}_\alpha$, for our running example, are the ones of $\mathcal{A}_\alpha$ in Example 3.2, after removing rules $\mathtt{var}, q_8, \mathtt{r}_9 \rightsquigarrow q_6, \varnothing$ and $\mathtt{var}, q_6, \mathtt{r}_9 \rightsquigarrow q_6, \varnothing$. These two rules apply to terms that are not long and for this reason the corresponding production rules have been withdrawn from the pre-grammar. In the initial configuration registers $\mathtt{id}$ and $\mathtt{needs}$ are respectively $\emptyset$ and $\{9\}$.*

The run of $M = \lambda xy.x(\lambda z.y)y$ on $\mathcal{P}_\alpha$ is now the following:

$$
\begin{array}{ll}
 & ((q_{10}, [\emptyset_9, \emptyset_4, \emptyset_0]); \emptyset; \{9\}) \\
\rightsquigarrow^{\langle \lambda, x \rangle} & ((q_8, [\{x\}_9, \emptyset_4, \emptyset_0]); \emptyset; \{9\}) \\
\rightsquigarrow^{\langle \lambda, y \rangle} & ((q_5, [\{x\}_9, \{y\}_4, \emptyset_0]); \emptyset; \{9\}) \\
\rightsquigarrow^{\langle \mathtt{var}, x \rangle} & ((q_6, [\{x\}_9, \{y\}_4, \emptyset_0]), (q_2, [\{x\}_9, \{y\}_4, \emptyset_0]); \{5 = 3\}; \emptyset) \\
\rightsquigarrow^{\langle \lambda, z \rangle} & ((q_1, [\{x\}_9, \{y\}_4, \{z\}_0]), (q_2, [\{x\}_9, \{y\}_4, \emptyset_0]); \{5 = 3\}; \emptyset) \\
\rightsquigarrow^{\langle \mathtt{var}, y \rangle} & ((q_2, [\{x\}_9, \{y\}_4, \emptyset_0]); \{5 = 3, 1 = 4\}; \emptyset) \\
\rightsquigarrow^{\langle \mathtt{var}, y \rangle} & (\varepsilon; \{5 = 3, 1 = 2 = 4\}; \emptyset).
\end{array}
$$

Proposition 4.7 *Given type α and a normal term M, then M is a long inhabitant of α if and only if $\mathcal{P}_\alpha$ accepts M with some output $(\mathtt{id}', \mathtt{needs}')$. Furthermore, for all identifications $n = m$ in $\mathtt{id}'$ one has $\mathtt{lab}(n) = \mathtt{lab}(m) = \mathtt{var}$, and $n \in \mathtt{needs}'$ implies that $\mathtt{lab}(n) \neq \mathtt{var}$.*

Proof. It follows from the definition of the operational semantics for inhabitation machines and for principal inhabitation machines, and from the fact that $\mathtt{preL}(\alpha)$ is obtained from $\mathtt{pre}(\alpha)$ dropping all rules $n := m_s\ q_{s-1} \cdots q_1$ such that $\mathtt{lab}(n) \neq \mathtt{var}$, that a normal term M is accepted by $\mathcal{P}_\alpha$ with some output $(\mathtt{id}', \mathtt{needs}')$ if and only if M is accepted by $\mathcal{A}_\alpha$ and M is long. The condition on $\mathtt{id}'$ follows from the fact that for every transition rule $\mathtt{var}, q_i, \mathtt{r}_n \rightsquigarrow q^1, \ldots, q^k, \varnothing$

in δ one has $\mathtt{lab}(i) = \mathtt{lab}(\mathtt{tail}(n)) = \mathtt{var}$. The condition on $\mathtt{needs}'$ is a consequence of the definition of $\mathcal{P}_\alpha$ and the initial configuration. □

Definition 4.8 *Consider a type α with set $\mathtt{OccT}(\alpha)$, a partition $\mathtt{id}$ of $\{\ n \in \mathtt{N}(\alpha) \mid \mathtt{lab}(n) = \mathtt{var}\ \}$ and a subset $\mathtt{needs} \subseteq \{\ n \in \mathtt{N}(\alpha) \mid \mathtt{lab}(n) \neq \mathtt{var}\ \}$. Then, $\mathtt{raise}(\alpha, \mathtt{id}, \mathtt{needs})$ denotes the type obtained from α as follows.*

- *Every occurrence of a subtype β with $\mathtt{n}(\beta) \in \mathtt{needs}$ is substituted by a fresh type variable;*
- *all occurrences of type variables are given fresh names, in such a way that two occurrences receive the same name if and only if they belong to the same class in $\mathtt{id}$.*

Proposition 4.9 *If $\mathcal{P}_\alpha$ accepts M with output $(\mathtt{id}, \mathtt{needs})$, then $\mathtt{raise}(\alpha, \mathtt{id}, \mathtt{needs})$ is the principal type of M.*

Proof. This is a direct consequence of the construction of the principal type for M in the proof of Proposition 4.3 in [8]. This principal type is obtained from α and is precisely type $\mathtt{raise}(\alpha, \mathtt{id}, \mathtt{needs})$. □

Corollary 4.10 *M is a principal long normal inhabitant of α if and only if $\mathcal{P}_\alpha$ accepts M with some output $(\mathtt{id}, \emptyset)$ such that $\mathtt{id}$ contains one class for each type variable in α, each class containing exactly the identifiers of all occurrences of that type variable.*

Example 4.11 *The run of M in Example 4.6 is accepting with output $(\{5 = 3, 1 = 2 = 4\}, \emptyset)$, where $\{5 = 3, 1 = 2 = 4\}$ represents the partition $\{\{0\}, \{3, 5\}, \{1, 2, 4\}\}$. We conclude that the principal type of M is $\mathtt{raise}(\alpha, \{5 = 3, 1 = 2 = 4\}, \emptyset) = ((o \to o') \to o' \to o'') \to o' \to o''$.*

The unique run of $M_4 = \lambda xy.y$ from Example 2.2 is

$$\begin{array}{ll} & ((q_{10}, [\emptyset_9, \emptyset_4, \emptyset_0]); \emptyset; \{9\}) \\ \rightsquigarrow^{\langle \lambda, x\rangle} & ((q_8, [\{x\}_9, \emptyset_4, \emptyset_0]); \emptyset; \{9\}) \\ \rightsquigarrow^{\langle \lambda, y\rangle} & ((q_5, [\{x\}_9, \{y\}_4, \emptyset_0]); \emptyset; \{9\}) \\ \rightsquigarrow^{\langle \mathtt{var}, x\rangle} & (\varepsilon; \{5 = 4\}; \{9\}) \end{array}$$

Thus, $\mathtt{raise}(\alpha, \{\{0\}, \{1\}, \{2\}, \{3\}, \{4, 5\}\}, \{9\}) = o' \to o \to o$ is the principal type of M_4.

The run of $M_3 = \lambda xy.x(\lambda z.z)(x(\lambda u.y)y)$ is accepting with output $(\varepsilon; \{\{0, 1, 2, 3, 4, 5\}\}; \emptyset)$. Since $\mathtt{raise}(\alpha, \{\{0, 1, 2, 3, 4, 5\}\}; \emptyset) = \alpha$, we conclude that M_3 is a principal long inhabitant of α.

5. Conclusions

In this paper we define inhabitation machines, following the formalism of Schubert et al., that deal with terms in a deterministic way. We start by defining machines to recognise normal inhabitants (depending on the pre-grammar that we use to build the machine, this will recognise all normal inhabitants or only those in long normal form). We further develop the machines to deal with the principal inhabitation problem and prove the correctness of our machines. Unlike

Schubert et al., we do not prove any closure properties for the languages generated by our machines. However, given that our machines are built from the notion of pre-grammar, and that such closure properties were proved for the pre-grammars in [1], we believe that the same properties hold for the machines in this paper and leave the details of such results for future work.

Another topic that is left for further development in an extended version of this paper, is the complexity of the principal inhabitation problem. Schubert et al. proved that the emptiness problem is PSPACE-complete for one operation machines, which are single assignment machines that at each step either read or write the registers. This is not exactly the case for our machines that deal with principal inhabitants. The rules for variables both read and write, but they only write in the global registers `id` and `needs` an amount of information that is at most polynomial with respect to the type. Therefore, we are convinced that a PSPACE bound can also be obtained for the principal type problem, using the inhabitation machines defined in this paper.

Acknowledgment

This work is partially funded by the ERDF through the COMPETE 2020 Programme within project POCI-01-0145-FEDER-006961, and by National Funds through the FCT as part of project UID/EEA/50014/2013; and by CMUP (UID/MAT/00144/2013), which is funded by FCT (Portugal) with national (MEC) and european structural funds through the programs FEDER, under the partnership agreement PT2020.

References

[1] S. ALVES, S. BRODA, M. RAMOS, Context Free Grammars, Pushdown Automata and Type Inhabitation. *submitted* (2017).

[2] T. AOTO, Uniqueness of Normal Proofs in Implicational Intuitionistic Logic. *J. of Logic, Lang. and Inf.* 8 (1999) 2, 217–242.

[3] A. BABAEV, S. SOLOV'EV, A coherence theorem for canonical morphisms in cartesian closed categories. *Journal of Mathematical Sciences* 20 (1982), 2263–2279.

[4] H. BARENDREGT, Lambda Calculi with Types. In: *Handbook of Logic in Computer Science*. 2, Clarendon Press, Oxford, 1992, 117–309.

[5] C. BEN-YELLES, *Type Assignment in the Lambda-Calculus: Syntax and Semantics*. Ph.D. thesis, University College of Swansea, 1979.

[6] S. BRODA, L. DAMAS, On the structure of normal λ-terms having a certain type. In: *Proc. 7th WoLLIC'2000*. 2000, 33–43.

[7] S. BRODA, L. DAMAS, Counting a Type's (Principal) Inhabitants. *Fundam. Inform.* 45 (2001) 1-2, 33–51.

[8] S. BRODA, L. DAMAS, On Long Normal Inhabitants of a Type. *J. Log. and Comput.* 15 (2005), 353–390.

[9] M. BUNDER, Proof finding algorithms for implicational logics. *Theoretical Computer Science* 232 (2000) 12, 165 – 186.

[10] J. HINDLEY, *Basic Simple Type Theory*. Cambridge Tracts in Theoretical Computer Science 42, Cambridge University Press, 1997.

[11] S. HIROKAWA, Infiniteness of proof (α) is polynomial-space complete. *Theor. Comput. Sci.* 206 (1998) 1-2, 331–339.

[12] W. HOWARD, The formulas-as-types notion of construction. In: J. SELDIN, J. HINDLEY (eds.), *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*. Academic Press, 1980, 479–490.

[13] Y. KOMORI, S. HIROKAWA, The Number of Proofs for a BCK-Formula. *J. Symb. Log.* 58 (1993) 2, 626–628.

[14] G. MINTS, Closed categories and the theory of proofs. *Journal of Mathematical Sciences* 15 (1981), 45–62.

[15] D. PRAWITZ, *Natural deduction: a proof-theoretical study*. Ph.D. thesis, Almqvist & Wiksell, 1965.

[16] A. SCHUBERT, W. DEKKERS, H. P. BARENDREGT, Automata Theoretic Account of Proof Search. In: *CSL 2015*. 2015, 128–143.

[17] R. STATMAN, Intuitionistic Propositional Logic is Polynomial-Space Complete. *Theor. Comput. Sci.* 9 (1979), 67–72.

[18] M. TAKAHASHI, Y. AKAMA, S. HIROKAWA, Normal Proofs and Their Grammar. *Information and Computation* 125 (1996) 2, 144–153.