A Unifying Framework for Type Inhabitation

Sandra Alves[1]
DCC-Faculty of Science & CRACS, University of Porto, Portugal
sandra@dcc.fc.up.pt
https://orcid.org/0000-0001-8840-5587

Sabine Broda[2]
DCC-Faculty of Science & CMUP, University of Porto, Portugal
sbb@dcc.fc.up.pt
https://orcid.org/0000-0002-3798-9348

Abstract

In this paper we define a framework to address different kinds of problems related to type inhabitation, such as type checking, the emptiness problem, generation of inhabitants and counting, in a uniform way. Our framework uses an alternative representation for types, called the pre-grammar of the type, on which different methods for these problems are based. Furthermore, we define a scheme for a decision algorithm that, for particular instantiations of the parameters, can be used to show different inhabitation related problems to be in PSPACE.

2012 ACM Subject Classification Theory of computation → Type theory, Theory of computation → Lambda calculus, Theory of computation → Rewrite systems

Keywords and phrases simple types, type inhabitation, rewriting, PSPACE

Digital Object Identifier 10.4230/LIPIcs.FSCD.2018.5

1 Introduction

Inhabitation of simply typed λ-terms and its related problems, such as type checking, the emptiness problem, generation of inhabitants, counting algorithms, etc. have been extensively studied throughout the years [2, 3, 5, 15, 14, 10, 12, 13, 6], using a variety of formalisms such as context-free grammars [15], tree-based methods [4], automata theory [13], amongst others. Despite the diversity of methods, there are common fundamental features that emerge from the different approaches.

One of these features is the implicit relation between the structure of a type and its normal inhabitants. The *Formula-Tree Method* by Broda and Damas [4] explores this relation by looking at a tree representation of the type, identifying what are called the primitive parts, which are then combined following a set of rules determined by the structure of the type. In the case of the inhabitation machines defined by Schubert et al. [13], the states of the automata used to recognize the inhabitants of a given type, as well as the transition relation between configurations of the machines, are obtained directly from the sub-expressions of the type. More recently, while studying the complexity of the principal inhabitation problem, Dudenhefner and Rehof [6] use the structure of the type to define a path relation identifying subformulas with the same atomic type. This path relation is then used in the definition

[1] 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.
[2] Partially funded 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.

3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).
Editor: Hélène Kirchner; Article No. 5; pp. 5:1–5:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of an algorithm that addresses principal inhabitation. Types and their structure are also fundamental in the definition of the context-free grammars by Takahashi et al. [15].

In this paper we highlight the importance of the underlying structure of types in the definition of methods for type inhabitation related problems. To that end, we present an alternative unifying representation of the type's structure, which we call the *pre-grammar* of the type. From this simple, yet powerful, device we extract rewriting methods to deal with type-checking, counting and generation of inhabitants.

Secondly, we explore the uniformity of decision algorithms defined over the years to prove that different inhabitation related problems are in PSPACE. Complexity of inhabitation related problems was first addressed by Statman [14] in the realm of propositional intuitionistic logic. The decidability of the logic was proved to be PSPACE complete, and therefore also the emptiness problem for the simply typed lambda calculus, due to the well-known Curry-Howard correspondence [11]. A direct syntactic proof of the same result, for the simply typed λ-calculus, was later given by Urzyczyn [16]. PSPACE completeness of the infiniteness problem was proved by Hirokawa [10], by reducing the emptiness problem to the infiniteness problem. In [6], PSPACE completeness was proved for the problem of principal inhabitation, by means of a non-deterministic algorithm for choosing a particular path relation for a given type. Also in the case of inhabitation machines [13], a PSPACE completeness result is obtained for the emptiness problem by means of a polynomial time alternating algorithm. In fact, several of the results mentioned above rely on polynomial time alternating algorithms. Note that the class of problems decidable in alternating polynomial time (AP) corresponds to the class of problems decidable in polynomial space (PSPACE). Following that, we define a scheme for a polynomial time alternating decision algorithm, which operates on the rules of the pre-grammar of the type. By instantiating the parameters of the algorithm scheme, we obtain different PSPACE decision algorithms for the problems of emptiness, counting and principal inhabitation.

We will restrict our methods and definitions to terms in normal form. In fact, most of the interesting questions related to inhabitation can be reduced to, or even just make sense for, normal terms. For instance, an inhabited type may have only a finite number of normal inhabitants, but has always an infinite number of (not necessarily normal) inhabitants. Also, every inhabited type is the principal type of an infinite number of terms, while it may not be the principal type of a term in normal form [9].

The rest of the paper is structured as follows. In the next section we introduce some preliminary notions. In Section 3, we present the notion of pre-grammars and prove some basic results. Using the pre-grammar representation, in Section 4, we define rewriting methods to address type checking and the emptiness problem, and explore closure properties, for intersection and union types. In Section 5 we define the scheme of an alternating decision algorithm, and its instances. Finally, in Section 6, we draw some conclusions and highlight some future work.

2 Preliminaries

In this paper we assume familiarity with the simply typed λ-calculus à la Curry [8]. We denote type variables (atoms) by $a, b, c, \ldots$ and arbitrary types by lower-case Greek letters $\alpha, \beta, \gamma, \sigma, \tau, \ldots$. The set of simple types is denoted by $\mathcal{T}$. We denote λ-terms by $M, N, \ldots$, which are built from an infinite countable set of term variables $\mathcal{V}$. Unless stated otherwise, we identify terms modulo α-equivalence. For type assignment we consider the system TA_λ as described in [8] and consider its inference rules for terms in β-normal form. Note that every

3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).
Editor: Hélène Kirchner; Article No. 5; pp. 5:2–5:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

β-normal λ-term is of the form $\lambda x.N$ or $xN_1 \cdots N_s$, where $N, N_1, \ldots, N_s$ are in normal form and $s \geq 0$. Different from [8] we define the depth of a λ-term by $\mathsf{depth}(\lambda x.N) = 1 + \mathsf{depth}(N)$, and $\mathsf{depth}(xN_1 \cdots N_s) = 1 + max(\mathsf{depth}(N_1), \ldots, \mathsf{depth}(N_s))$ for $s \geq 1$, and $\mathsf{depth}(x) = 1$. With this definition the depth of a term M, such that $\Gamma \vdash M : \tau$, corresponds directly to the height of the unique TA_λ-deduction of this fact, as defined below. A *context* is a finite set Γ of declarations $x : \sigma$, where $x \in \mathcal{V}$ and $\sigma \in \mathcal{T}$, such that all term variables occurring in Γ are distinct from each other. The set of term variables occurring in Γ is denoted by $\mathsf{Subj}(\Gamma)$. The union of contexts is *consistent* if it does not contain different type declarations for the same term variable.

▶ **Definition 1.** We write $\Gamma \vdash M : \tau$ and say that type τ can be assigned to term M in context Γ, if this formula can be obtained by applying the rules below a finite number of times.

- If $\Gamma \vdash N : \sigma_2$ and $\Gamma \cup \{x : \sigma_1\}$ is consistent, then $\Gamma \setminus \{x : \sigma_1\} \vdash \lambda x.N : \sigma_1 \to \sigma_2$.
- if $\Gamma_i \vdash N_i : \sigma_i$, for $1 \leq i \leq s$ $(s \geq 0)$, then $\Gamma \vdash xN_1 \cdots N_s : \sigma$, if $\Gamma = \Gamma_1 \cup \cdots \cup \Gamma_s \cup \{x : \sigma_1 \to \cdots \to \sigma_s \to \sigma\}$ is consistent;

If $\Gamma = \emptyset$, then we also write $\vdash M : \tau$ instead of $\Gamma \vdash M : \tau$ and say that M is an *inhabitant* of type τ. The set of all (normal) inhabitants of τ is denoted by $\mathsf{Nhabs}(\tau)$.

One knows that $\Gamma \vdash M : \tau$ implies that the set of term variables in Γ coincides with the set of free variables in M, i.e. $\mathsf{Subj}(\Gamma) = \mathsf{FV}(M)$, cf. Lemma 2A10 in [8]. Furthermore, for every derivable formula $\Gamma \vdash M : \tau$ there is exactly one deduction in TA_λ.

▶ **Example 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, $M_1 = \lambda xy.x(\lambda z.y)y$ and $M_2 = \lambda x.x(\lambda y.y)$, for which one has $\mathsf{depth}(M_1) = 5$ and $\mathsf{depth}(M_2) = 4$.

▶ **Definition 3.** The polarity of occurrences of subtypes in a type τ is defined as follows.

- τ is a positive occurrence in τ;
- if $\rho \to \sigma$ occurs positively (resp. negatively) in τ, then that occurrence of ρ is negative (resp. positive) and that occurrence of σ is positive (resp. negative) in τ.

Following the notation in [8], we will on occasions write $\underline{o}$ when referring to a particular occurrence of an object o. Every type τ can be uniquely written as $\tau = \tau_1 \to \ldots \to \tau_l \to a$, where a is a type variable and $l \geq 0$. Type variable a is called the *tail* of τ and denoted by $\mathsf{tail}(\tau)$. If $l \geq 1$, then $\tau_1, \ldots, \tau_l$ are called the *arguments* of τ. An occurrence $\underline{\sigma}$ in τ is called a *negative subpremise* of τ iff it is the argument of a positive occurrence of a subtype in τ.

Consider a term M and a type τ such that $\vdash M : \tau$, as well as a formula $\Gamma \vdash N : \sigma$, appearing in the unique TA_λ-deduction of $\vdash M : \tau$. In the following, we assign to each $X \in \mathsf{Subj}(\Gamma) \cup \{N\}$ an occurrence $\mathsf{st}(X)$ of a subtype in τ. The definition of st is bottom-up, starting with $\vdash M : \tau$.

- For $\vdash M : \tau$, let $\mathsf{st}(M) = \tau$.
- Now consider $\Gamma \setminus \{x : \sigma_1\} \vdash \lambda x.N : \sigma_1 \to \sigma_2$, because $\Gamma \vdash N : \sigma_2$ and because $\Gamma \cup \{x : \sigma_1\}$ is consistent. Consider $\mathsf{st}(\lambda x.N) = \underline{\sigma_1 \to \sigma_2}$ for $\Gamma \setminus \{x : \sigma_1\} \vdash \lambda x.N : \sigma_1 \to \sigma_2$. Then, for $\Gamma \vdash N : \sigma_2$ let $\mathsf{st}(N)$ be the occurrence of σ_2 in $\mathsf{st}(\lambda x.N)$. If $x \in \mathsf{Subj}(\Gamma)$, then $\mathsf{st}(x)$ is the occurrence of σ_1 in $\underline{\sigma_1 \to \sigma_2}$. All other variables in $\mathsf{Subj}(\Gamma)$ are assigned the same occurrences as for the formula $\Gamma \setminus \{x : \sigma_1\} \vdash \lambda x.N : \sigma_1 \to \sigma_2$.
- Finally let $\Gamma \vdash xN_1 \cdots N_s : \sigma$, because $\Gamma_i \vdash N_i : \sigma_i$, for $1 \leq i \leq s$ $(s \geq 0)$, and because $\Gamma = \Gamma_1 \cup \cdots \Gamma_s \cup \{x : \sigma_1 \to \cdots \to \sigma_s \to \sigma\}$ is consistent. If $\mathsf{st}(x) = \underline{\sigma_1 \to \cdots \to \sigma_s \to \sigma}$, then $\mathsf{st}(N_i)$ is the occurrence of σ_i in $\mathsf{st}(x)$, for $\Gamma_i \vdash N_i : \sigma_i$ and $1 \leq i \leq s$ $(s \geq 0)$. The variables in $\mathsf{Subj}(\Gamma_i)$ are assigned the same occurrences as for $\Gamma \vdash xN_1 \cdots N_s : \sigma$.

3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).
Editor: Hélène Kirchner; Article No. 5; pp. 5:3–5:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The following lemma, cf. [4], establishes the relationship between occurrences of variables in abstraction sequences and occurrences of subterms in M, respectively with negative subpremises and positive occurrences of subtypes in τ and can be easily proved using the definition of st above, as well as Definition 3. The established relationship will be explored in the definition of pre-grammars in the next section.

▶ **Lemma 4.** *Consider a term M in β-normal form and a type τ such that $\vdash M : \tau$, as well as a formula $\Gamma \vdash N : \sigma$, appearing in the unique TA_λ-deduction of $\vdash M : \tau$. If $x : \sigma_x \in \Gamma$, then $\mathsf{st}(x) = \underline{\sigma_x}$ is a negative subpremise in τ. Furthermore, $\mathsf{st}(N) = \underline{\sigma}$ is a positive occurrence of subtype σ in τ.*

3 Pre-grammars

In this section we describe how to obtain for a type τ a set of rewriting rules, which we call the *pre-grammar* of τ and denote by $\mathsf{pre}(\tau)$. We start by associating to each type τ a set $\mathsf{occ}\mathsf{T}(\tau)$ that contains for each type occurrence $\underline{\sigma}$ a tuple (σ, n, l), where $n \in \mathbb{N}$, and $\mathsf{l} \in \{\mathsf{var}\} \cup \{\ n \to m \mid n, m \in \mathbb{N}\ \}$. Distinct occurrences of subtypes are assigned distinct tuples. This set is uniquely defined, up to isomorphism between integers used in the tuples.

▶ **Definition 5.** Given a type $\tau \in \mathcal{T}$ let $\mathsf{occ}\mathsf{T}(\tau)$ be the smallest set satisfying the following.

- For each occurrence of a type variable a in τ there is a tuple $(a, n, \mathsf{var}) \in \mathsf{occ}\mathsf{T}(\tau)$;
- if $\rho \to \sigma$ is an occurrence of a subtype of τ, and $(\rho, n, \mathsf{l}_\rho), (\sigma, m, \mathsf{l}_\sigma) \in \mathsf{occ}\mathsf{T}(\tau)$ are the tuples corresponding to ρ and σ in this occurrence, then $(\rho \to \sigma, k, n \to m) \in \mathsf{occ}\mathsf{T}(\tau)$;
- for each $n \in \mathbb{N}$ there is at most one tuple $(\sigma, n, \mathsf{l}) \in \mathsf{occ}\mathsf{T}(\tau)$.

Furthermore, given a particular occurrence $\underline{\sigma}$ of a subtype of τ we denote by $\mathsf{n}(\underline{\sigma})$ the unique integer n such that $(\sigma, n, \mathsf{l}) \in \mathsf{occ}\mathsf{T}(\tau)$. We frequently will refer to $\mathsf{n}(\sigma)$ as the *identifier* of $\underline{\sigma}$ w.r.t. $\mathsf{occ}\mathsf{T}(\tau)$. Finally, $\mathsf{t}(n) = \sigma$, $\mathsf{lab}(n) = \mathsf{l}$, and $\mathsf{N}(\tau) = \{\ n \mid (\sigma, n, \mathsf{l}) \in \mathsf{occ}\mathsf{T}(\tau)\ \}$.

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, 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.

▶ **Definition 6.** We say that two integers $n, m \in \mathsf{N}(\tau)$ are equivalent w.r.t. $\mathsf{occ}\mathsf{T}(\tau)$, and write $n \equiv_{\mathsf{occ}\mathsf{T}} m$, if and only if $\mathsf{t}(n) = \mathsf{t}(m)$. The binary relation $T(\tau) \subseteq \mathsf{N}(\tau) \times \mathsf{N}(\tau)$ is defined by $(p_2, p_3) \in T(\tau)$ iff $(\beta, p_3, p_1 \to p_2) \in \mathsf{occ}\mathsf{T}(\tau)$, i.e. $\beta = \beta_1 \to \beta_2$, $\mathsf{n}(\beta_1) = p_1$, $\mathsf{n}(\beta_2) = p_2$, and $\mathsf{n}(\beta) = p_3$. Furthermore, for $(p_2, p_3) \in T(\tau)$ let $\mathsf{q}(p_2, p_3) = p_1$.

▶ **Lemma 7.** *If τ contains s occurrences $\underline{a_1}, \ldots, \underline{a_s}$, of type variables, then the graph of $T(\tau)$, whose set of nodes is $\mathsf{N}(\tau)$, consists of s unary trees with roots $\mathsf{n}(a_1), \ldots, \mathsf{n}(a_s)$, respectively.*

▶ **Example 8.** For $\alpha = ((o \to o) \to o \to o) \to o \to o$ from Example 2 the set $\mathsf{occ}\mathsf{T}(\alpha)$ contains eleven tuples (β, n, l), where β, n and l are given below.

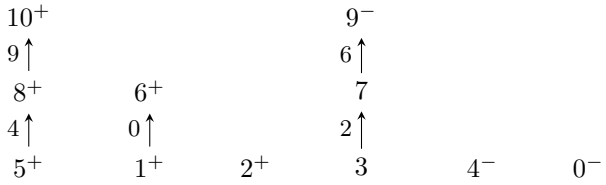
β	n	l
o	0	var
o	1	var
o	2	var
o	3	var

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	$9 \to 8$

3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).
Editor: Hélène Kirchner; Article No. 5; pp. 5:4–5:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The equivalence relation $\equiv_{\mathsf{occT}}$ partitions $\mathsf{N}(\alpha)$ 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, $\mathsf{pre}(\tau)$ can be computed from $\mathsf{occT}(\tau)$ and $T(\tau)$ as follows.

▶ **Definition 9.** Given a type τ and a set of tuples $\mathsf{occT}(\tau)$, we denote by $\mathsf{pre}(\tau)$ the smallest set of rules satisfying the following conditions.

- If $m^+, k^-, n^+ \in \mathsf{N}(\tau)$, $(\beta, m, k \to n) \in \mathsf{occT}(\tau)$, then $m := \lambda k.n \in \mathsf{pre}(\tau)$;
- if $m^+, p_0^- \in \mathsf{N}(\tau)$ and $(p_s, p_{s-1}), \ldots, (p_2, p_1), (p_1, p_0) \in T(\tau)$, for some $s \geq 0$, $m^+ \equiv_{\mathsf{occT}} p_s$, $\mathsf{q}(p_i, p_{i-1}) = n_i$ for $1 \leq i \leq s$, then $m := p_0\ n_1 \cdots n_s \in \mathsf{pre}(\tau)$.

▶ **Note 10.** If τ is inhabited, then there is exactly one rule for $\mathsf{n}(\tau)$ in $\mathsf{pre}(\tau)$. This rule is of the form $\mathsf{n}(\tau) := \lambda k.n$, for some $k^-, n^+ \in \mathsf{N}(\tau)$. Also, $\mathsf{n}(\tau)$ occurs in no other rule.

It is straightforward to verify the following two properties of $\mathsf{pre}(\tau)$.

▶ **Lemma 11.**

1. *Consider a positive occurrence of a subformula $\rho \to \sigma$ in τ and the corresponding tuple in $(\rho \to \sigma, m, k \to n) \in \mathsf{occT}(\tau)$. Then, $m := \lambda k.n \in \mathsf{pre}(\tau)$, and there is no other rule of the form $m := \lambda k'.n'$ in $\mathsf{pre}(\tau)$.*
2. *Consider a negative subpremise $\rho = \sigma_1 \to \cdots \to \sigma_s \to \sigma$ in τ and let $(\sigma_1, n_1, \mathsf{l}_1), \ldots, (\sigma_s, n_s, \mathsf{l}_s), (\sigma, n, \mathsf{l}_n), (\rho, k, \mathsf{l}_k)$ be the tuples in $\mathsf{occT}(\tau)$ corresponding to $\sigma_1, \ldots, \sigma_s, \sigma, \rho$, respectively. If $m^+ \in \mathsf{occT}(\tau)$, such that $m^+ \equiv_{\mathsf{occT}} n$, then $m := k\ n_1 \cdots n_s \in \mathsf{pre}(\tau)$. Furthermore, there is no other rule of the form $m := k\ n'_1 \cdots n'_t$ $(t \geq 0)$ in $\mathsf{pre}(\tau)$.*

▶ **Example 12.** From $\mathsf{occT}(\alpha)$ and $T(\alpha)$ in Example 8 we obtain the following set $\mathsf{pre}(\alpha)$ containing fourteen rewriting rules.

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	:=	$9\ 6\ 2 \mid 4 \mid 0$

4 Inhabitation

4.1 Type Checking

In the following we describe a rewriting algorithm that, given a type τ and a term M, verifies if $\vdash M : \tau$, i.e. checks if $M \in \mathsf{Nhabs}(\tau)$. During the rewriting process we use objects with the structure of λ-terms, but such that integers can be used as placeholders for variables. We refer to these objects as *extended terms*. We denote by $N[k/x]$ the (extended) term obtained from N by replacing all free occurrences of variable x in N by placeholder k.

▶ **Definition 13.** Given a type τ, we write $(M, m) \hookrightarrow (N_1, n_1) \cdots (N_s, n_s)$, $(s \geq 0)$, where $M, N_1, \ldots, N_s$ are extended terms and $m, n_1, \ldots, n_s \in \mathsf{N}(\tau)$, if one of the following applies.

- If $m := \lambda k.n \in \mathsf{pre}(\tau)$, then $(\lambda x.N, m) \hookrightarrow (N[k/x], n)$;
- if $m := k\ n_1 \cdots n_s \in \mathsf{pre}(\tau)$, then $(k\ N_1 \cdots N_s, m) \hookrightarrow (N_1, n_1), \ldots, (N_s, n_s)$.

3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).
Editor: Hélène Kirchner; Article No. 5; pp. 5:5–5:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The definition of $\hookrightarrow$ extends, in the usual way, to rewriting of sequences of pairs, where we assume that sequences of pairs are processed from left to right. Then, $\hookrightarrow^*$ denotes the reflexive, transitive closure of $\hookrightarrow$.

Note that, by Lemma 11, in each step of $(M, \mathsf{n}(\tau)) \hookrightarrow^* \epsilon$, at most one rule of $\mathsf{pre}(\tau)$ applies to each pair. Consequently, the type-checking algorithm is deterministic and the sequence from $(M, \mathsf{n}(\tau))$ to ϵ is unique.

▶ **Example 14.** Consider α as before and $M_1 = \lambda xy.x(\lambda z.y)y$ from Example 2. Then,

$$\begin{aligned}(\lambda xy.x(\lambda z.y)y, 10) &\hookrightarrow (\lambda y.9(\lambda z.y)y, 8) \hookrightarrow (9(\lambda z.4)4, 5)\\ &\hookrightarrow (\lambda z.4, 6), (4, 2) \hookrightarrow (4, 1), (4, 2) \hookrightarrow (4, 2) \hookrightarrow \epsilon.\end{aligned}$$

▶ **Theorem 15.** $\mathsf{Nhabs}(\tau) = \{\ M \mid (M, \mathsf{n}(\tau)) \hookrightarrow^* \epsilon\ \}$.

Proof. We show that for any term M, context $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ and type σ, if $\Gamma \vdash M : \sigma$, then $(M[\mathsf{n}(\sigma_1)/x_1, \cdots, \mathsf{n}(\sigma_n)/x_n], \mathsf{n}(\sigma)) \hookrightarrow^* \epsilon$, using $M[\Gamma]$ as an abbreviation for $M[\mathsf{n}(\sigma_1)/x_1, \cdots, \mathsf{n}(\sigma_n)/x_n]$. As a consequence it follows that $\mathsf{Nhabs}(\tau) \subseteq \{\ M \mid (M, \mathsf{n}(\tau)) \hookrightarrow^* \epsilon\ \}$. We proceed by induction on $\mathsf{depth}(M)$. First, consider $M = xN_1 \cdots N_s$ and suppose that $\Gamma \vdash xN_1 \cdots N_s : \sigma$, because $\Gamma_i \vdash N_i : \sigma_i$, for $1 \leq i \leq s$ ($s \geq 0$), and because $\Gamma = \Gamma_1 \cup \cdots \cup \Gamma_s \cup \{x : \sigma_1 \to \cdots \to \sigma_s \to \sigma\}$ is consistent. By Lemma 4, we know that there is a negative subpremise $\mathsf{st}(x) = \underline{\sigma_1 \to \cdots \to \sigma_s \to \sigma}$ in τ, as well as a positive occurrence $\mathsf{st}(xN_1 \cdots N_s)$ of σ in τ, corresponding to $\Gamma \vdash xN_1 \cdots N_s : \sigma$. Let n and m be respectively the identifier of the occurrence of σ in $\mathsf{st}(x)$, and of the positive occurrence $\mathsf{st}(xN_1 \cdots N_s)$ of σ in τ. Then, $m^+ \equiv_{\mathsf{occT}} n$ and it follows from Lemma 11 that $m := k\ \mathsf{n}(\sigma_1) \cdots \mathsf{n}(\sigma_s) \in \mathsf{pre}(\tau)$, where $k = \mathsf{n}(\sigma_1 \to \cdots \to \sigma_s \to \sigma)$. Thus $(M[\Gamma], m) \hookrightarrow (N_1[\Gamma], \mathsf{n}(\sigma_1)), \ldots, (N_s[\Gamma], \mathsf{n}(\sigma_s))$. But $N_i[\Gamma] = N_i[\Gamma_i]$, for $1 \leq i \leq s$. Consequently, the result follows from the induction hypothesis. Now, consider $M = \lambda x.N$ and suppose that we have $\Gamma \setminus \{x : \sigma_1\} \vdash \lambda x.N : \sigma_1 \to \sigma_2$, because $\Gamma \vdash N : \sigma_2$ and $\Gamma \cup \{x : \sigma_1\}$ is consistent. It follows from Lemma 4 that $\mathsf{st}(\lambda x.N) = \underline{\sigma_1 \to \sigma_2}$ is a positive occurrence of $\sigma_1 \to \sigma_2$ in τ. We consider the corresponding tuple $(\sigma_1 \to \sigma_2, m, k \to n) \in \mathsf{occT}(\tau)$, where $\mathsf{n}(\sigma_1 \to \sigma_2) = m$, $\mathsf{n}(\sigma_1) = k$, and $\mathsf{n}(\sigma_2) = n$. By Lemma 11, there is a rule $m := \lambda k.n \in \mathsf{pre}(\tau)$. Furthermore, $\mathsf{Subj}(\Gamma) = \mathsf{FV}(N)$ and $\mathsf{Subj}(\Gamma \setminus \{x : \sigma_1\}) = \mathsf{FV}(\lambda x.N)$. Thus, we have $(M[\Gamma \setminus \{x : \sigma_1\}], m) \hookrightarrow (N[\Gamma], n)$. The result follows from the induction hypothesis.

For the other inclusion consider a term M, such that $(M, \mathsf{n}(\tau)) \hookrightarrow^* \epsilon$. Let (E, p) be any pair appearing in the corresponding rewriting sequence, where E is an extended term and $p = \mathsf{n}(\sigma)$, for some type occurrence $\underline{\sigma}$ in τ, i.e. $\sigma = \mathsf{t}(p)$. Naturally, we have $(E, p) \hookrightarrow^* \epsilon$. Let $P = \{p_1, \ldots, p_l\}$ be the set of placeholders that occur in E. Furthermore, let us interpret each of the integers in P as the name of a term variable. We will show, by induction on the length of $(E, p) \hookrightarrow^* \epsilon$, that $\Gamma_P \vdash E : \mathsf{t}(p)$, where $\Gamma_P = \{p_1 : \mathsf{t}(p_1), \ldots, p_l : \mathsf{t}(p_l)\}$. In particular, it follows that $\vdash M : \tau$. First, consider $E = \lambda x.E'$ such that $(\lambda x.E', p) \hookrightarrow (E'[k/x], n)$ by rule $p := \lambda k.n \in \mathsf{pre}(\tau)$. This means that there is a positive occurrence of a subtype $\mathsf{t}(p) = \mathsf{t}(k) \to \mathsf{t}(n)$ in τ. By the induction hypothesis, we have $\Gamma_{E'} \cup \{k : \mathsf{t}(k)\} \vdash E'[k/x] : \mathsf{t}(n)$. Thus, $\Gamma_{E'} \vdash \lambda k.E'[k/x] : \mathsf{t}(p)$, but $\lambda k.E'[k/x] \equiv_\alpha \lambda x.E'$. Finally, consider $E = k\ E_1 \cdots E_s$, such that $(k\ E_1 \cdots E_s, p) \hookrightarrow (E_1, n_1), \ldots, (E_s, n_s)$ by rule $p := k\ n_1 \cdots n_s \in \mathsf{pre}(\tau)$, where $s \geq 0$. Then, $\mathsf{t}(k) = \mathsf{t}(n_1) \to \cdots \to \mathsf{t}(n_s) \to \mathsf{t}(p)$ is a negative subpremise in τ. By the induction hypothesis, we have $\Gamma_{E_i} \vdash E_i : \mathsf{t}(n_i)$, for $1 \leq i \leq s$ and $s \geq 0$. Furthermore, $\Gamma_E = \Gamma_{E_1} \cup \cdots \cup \Gamma_{E_s} \cup \{k : \mathsf{t}(k)\}$ is consistent by definition. Thus, $\Gamma_E \vdash E : \mathsf{t}(p)$. ◀

3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).
Editor: Hélène Kirchner; Article No. 5; pp. 5:6–5:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

4.2 The Emptiness Problem

In this subsection we define a rewriting algorithm to decide if a given type τ has a normal inhabitant. Contrary to the previous one, this algorithm is non-deterministic, since more than one rule may apply at each step. On the other hand, it provides us with a simple tool to show that the emptiness problem for simple types is in PSPACE, and can be used for generation as well as for counting.

▶ **Definition 16.** Given a type τ, an identifier $m \in \mathsf{N}(\tau)$ and a set $V \subseteq \mathsf{N}(\tau)$, we write $(m, V) \rightsquigarrow (n_1, V'), \ldots, (n_s, V')$ if one of the following applies.

- If $m := \lambda k.n \in \mathsf{pre}(\tau)$, then $(m, V) \rightsquigarrow (n, V \cup \{k\})$;
- if $m := k\ n_1 \cdots n_s \in \mathsf{pre}(\tau)$ and $k \in V$, then $(m, V) \rightsquigarrow (n_1, V), \ldots, (n_s, V)$.

The definition of $\rightsquigarrow$ extends, in the usual way, to rewriting of sequences of pairs. Then, $\rightsquigarrow^*$ denotes the reflexive, transitive closure of $\rightsquigarrow$.

▶ **Definition 17.** For a particular rewriting sequence of $(\mathsf{n}(\tau), \emptyset) \rightsquigarrow^* \epsilon$, we define a function pair that computes for each (m, V) in that rewriting sequence a tuple $(M, \Gamma) = \mathsf{pair}(m, V)$. For convenience we will use identifiers as indexes of term variables in such a way that the type assigned to a variable with name x_n, for $n \in \mathsf{N}(\tau)$, is always $\mathsf{t}(n)$. The function pair is recursively defined as follows.

- If $(m, V) \rightsquigarrow (n, V \cup \{k\})$ because $m := \lambda k.n \in \mathsf{pre}(\tau)$, then $\mathsf{pair}(m, V) = (\lambda x_k.N, \Gamma \setminus \{x_k : \mathsf{t}(k)\})$, where $(N, \Gamma) = \mathsf{pair}(n, V \cup \{k\})$;
- if $(m, V) \rightsquigarrow (n_1, V), \ldots, (n_s, V)$ because $m := k\ n_1 \cdots n_s \in \mathsf{pre}(\tau)$ and $k \in V$, then $\mathsf{pair}(m, V) = (x_k\ N_1 \cdots N_s, \{x_k : \mathsf{t}(k)\} \cup \Gamma_1 \cup \cdots \cup \Gamma_s)$, where $(N_i, \Gamma_i) = \mathsf{pair}(n_i, V)$, for $1 \leq i \leq s$ $(s \geq 0)$.

Note that function pair actually does not depend on set V, but on the identifier m and on the rule in $\mathsf{pre}(\tau)$, which is used in each step of the rewriting sequence. The rule is implicitly given by the pairs appearing on the right of $\rightsquigarrow$ in Definition 16, unless it is of the form $m := k$. In that case $(m, V) \rightsquigarrow \epsilon$ and there might be more than one identifier $k \in V$ such that $m := k\ \in \mathsf{pre}(\tau)$. To guarantee that pair is well-defined, we suppose that in each rewriting step, the corresponding rewriting rule is given, either implicitly or explicitly. The correctness of function pair is stated in the following lemma.

▶ **Lemma 18.** *If $(m, V) \rightsquigarrow^* \epsilon$ and $(M, \Gamma) = \mathsf{pair}(m, V)$ for some corresponding rewriting sequence, then $\Gamma \vdash M : \mathsf{t}(m)$.*

Proof. By structural induction on M. We first consider the case where $\mathsf{pair}(m, V) = (\lambda x_k.N, \Gamma \setminus \{x_k : \mathsf{t}(k)\})$, which follows from $(m, V) \rightsquigarrow (n, V \cup \{k\}) \rightsquigarrow^* \epsilon$ because $m := \lambda k.n \in \mathsf{pre}(\tau)$ and $(N, \Gamma) = pair(n, V \cup \{k\})$. By the induction hypothesis, $\Gamma \vdash N : \mathsf{t}(n)$ and by definition $\Gamma \cup \{x_k : \mathsf{t}(k)\}$ is always consistent. Therefore, $\Gamma \vdash \lambda x_k.N : \mathsf{t}(k) \to \mathsf{t}(n)$. Now consider $\mathsf{pair}(m, V) = (x_k N_1 \cdots N_s, \{x_k : \mathsf{t}(k)\} \cup \Gamma_1 \cup \cdots \cup \Gamma_s)$, which follows from $(m, V) \rightsquigarrow (n_1, V), \ldots, (n_s, V) \rightsquigarrow^* \epsilon$ because $m := k\ n_1 \cdots n_s \in \mathsf{pre}(\tau)$ and $k \in V$, $(N_i, \Gamma_i) = \mathsf{pair}(n_i, V)$, for $1 \leq i \leq s$ $(s \geq 0)$. By the induction hypothesis $\Gamma_i \vdash N_i : \mathsf{t}(n_i)$ and by definition $\{x_k : \mathsf{t}(k)\} \cup \Gamma_1 \cup \cdots \cup \Gamma_s$ is consistent. It follows from $m := k\ n_1 \cdots n_s \in \mathsf{pre}(\tau)$ that $\mathsf{t}(k) = \mathsf{t}(n_1) \to \cdots \to \mathsf{t}(n_s) \to \mathsf{t}(m)$. Therefore $\{x_k : \mathsf{t}(k)\} \cup \Gamma_1 \cup \cdots \cup \Gamma_s \vdash x_k N_1 \cdots N_s : \mathsf{t}(m)$. ◀

▶ **Example 19.** Consider α and $\mathsf{pre}(\alpha)$ from Example 8. Then,

$(10, \emptyset) \rightsquigarrow (8, \{9\}) \rightsquigarrow (5, \{4, 9\}) \rightsquigarrow \epsilon$. Similarly,

3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).
Editor: Hélène Kirchner; Article No. 5; pp. 5:7–5:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$$\begin{aligned}(10,\emptyset) &\rightsquigarrow (8,\{9\}) \rightsquigarrow (6,\{9\}) \rightsquigarrow (1,\{0,9\}) \rightsquigarrow (6,\{0,9\}),(2,\{0,9\})\\ &\rightsquigarrow (1,\{0,9\}),(2,\{0,9\}) \rightsquigarrow (2,\{0,9\}) \rightsquigarrow \epsilon.\end{aligned}$$

For the first two pairs in this last rewriting sequence we have respectively $\mathsf{pair}(10,\emptyset) = (\lambda x_9.x_9(\lambda x_0.x_9(\lambda x_0.x_0)x_0),\emptyset)$ and $\mathsf{pair}(8,\{9\}) = (x_9(\lambda x_0.x_9(\lambda x_0.x_0)x_0),\{x_9 : \mathsf{t}(9)\})$, where $\mathsf{t}(9) = (o \to o) \to o \to o$. Also, $\vdash \lambda x_9.x_9(\lambda x_0.x_9(\lambda x_0.x_0)x_0) : \mathsf{t}(10)$ and $\{x_9 : \mathsf{t}(9)\} \vdash x_9(\lambda x_0.x_9(\lambda x_0.x_0)x_0) : \mathsf{t}(8)$, where $\mathsf{t}(8) = o \to o$ and $\mathsf{t}(10) = \alpha$.

For the first rewriting sequence we have $\mathsf{pair}(10,\emptyset) = (\lambda x_9 x_4.x_4,\emptyset)$, $\mathsf{pair}(8,\{9\}) = (\lambda x_4.x_4,\emptyset)$, and $\mathsf{pair}(5,\{4,9\}) = (x_4,\{x_4 : \mathsf{t}(4)\})$, where $\mathsf{t}(4) = o$. Also, $\vdash \lambda x_9 x_4.x_4 : \mathsf{t}(10)$, $\vdash \lambda x_4.x_4 : \mathsf{t}(8)$, and $\{x_4 : \mathsf{t}(4)\} \vdash x_4 : \mathsf{t}(5)$, with $\mathsf{t}(5) = o$.

▶ **Theorem 20.** $\mathsf{Nhabs}(\tau) \neq \emptyset$ *if and only if* $(\mathsf{n}(\tau),\emptyset) \rightsquigarrow^* \epsilon$.

Proof. The 'if' part follows from Lemma 18. For the 'only if' part, we show that for any term M, context Γ and type σ, if $\Gamma \vdash M : \sigma$, then $(\mathsf{n}(\sigma), V_\Gamma) \rightsquigarrow^* \epsilon$, where $V_\Gamma = \{\, \mathsf{n}(\rho) \mid x : \rho \in \Gamma \,\}$. First, consider $M = \lambda x.N$ and suppose that we have $\Gamma \setminus \{x : \sigma_1\} \vdash \lambda x.N : \sigma_1 \to \sigma_2$, because $\Gamma \vdash N : \sigma_2$ and $\Gamma \cup \{x : \sigma_1\}$ is consistent. It follows from Lemma 4 that $\mathsf{st}(\lambda x.N) = \underline{\sigma_1 \to \sigma_2}$ is a positive occurrence of $\sigma_1 \to \sigma_2$ in τ. We consider the corresponding tuple $(\sigma_1 \to \sigma_2, m, k \to n) \in \mathsf{occT}(\tau)$, where $\mathsf{n}(\sigma_1 \to \sigma_2) = m$, $\mathsf{n}(\sigma_1) = k$, and $\mathsf{n}(\sigma_2) = n$. By Lemma 11, there is a rule $m := \lambda k.n \in \mathsf{pre}(\tau)$. Thus, $(m, V_{\Gamma\setminus\{x:\sigma_1\}}) \rightsquigarrow (n, V_{\Gamma\setminus\{x:\sigma_1\}} \cup \{k\})$. But, $V_{\Gamma\setminus\{x:\sigma_1\}} \cup \{k\} = V_\Gamma$ and $(n, V_\Gamma) \rightsquigarrow^* \epsilon$ follows from the induction hypothesis. Now, consider $M = xN_1 \cdots N_s$ and suppose that $\Gamma \vdash xN_1 \cdots N_s : \sigma$, because $\Gamma_i \vdash N_i : \sigma_i$, for $1 \leq i \leq s$ ($s \geq 0$), and because $\Gamma = \Gamma_1 \cup \cdots \cup \Gamma_s \cup \{x : \sigma_1 \to \cdots \to \sigma_s \to \sigma\}$ is consistent. By Lemma 4, $\mathsf{st}(x) = \underline{\sigma_1 \to \cdots \to \sigma_s \to \sigma}$ is a negative subpremise of τ. It follows from Lemma 11 that $m := k\ \mathsf{n}(\sigma_1) \cdots \mathsf{n}(\sigma_s) \in \mathsf{pre}(\tau)$, where $m = \mathsf{n}(\sigma)$ and $k = \mathsf{n}(\sigma_1 \to \cdots \to \sigma_s \to \sigma)$. Thus $(m, V_\Gamma) \rightsquigarrow (\mathsf{n}(\sigma_1), V_\Gamma), \ldots, (\mathsf{n}(\sigma_s), V_\Gamma)$. By the induction hypothesis, we have $(\mathsf{n}(\sigma_i), V_{\Gamma_i}) \rightsquigarrow^* \epsilon$, for $1 \leq i \leq s$. Since $V_{\Gamma_i} \subseteq V_\Gamma$, we conclude that $(\mathsf{n}(\sigma_1), V_\Gamma), \ldots, (\mathsf{n}(\sigma_s), V_\Gamma) \rightsquigarrow^* \epsilon$. ◀

4.3 Closure Properties

In this section we combine the pre-grammars of two types τ_1 and τ_2 in order to obtain pre-grammars for $\mathsf{Nhabs}(\tau_1) \cap \mathsf{Nhabs}(\tau_2)$ and for $\mathsf{Nhabs}(\tau_1) \cup \mathsf{Nhabs}(\tau_2)$, respectively. This allows us to extend our methods to a bigger range of types, such as sum types of rank 1.

▶ **Definition 21.** Given types τ_1 and τ_2, we define $\mathsf{N}(\tau_1 \cap \tau_2) = \mathsf{N}(\tau_1) \times \mathsf{N}(\tau_2)$. Furthermore, let $\mathsf{pre}(\tau_1 \cap \tau_2)$ denote the smallest set of rules satisfying the following.

- If $m_i := \lambda k_i.n_i \in \mathsf{pre}(\tau_i)$ $(i = 1, 2)$, then $(m_1, m_2) := \lambda(k_1, k_2).(n_1, n_2) \in \mathsf{pre}(\tau_1 \cap \tau_2)$;
- if $m_i := k_i\ n_1^i \cdots n_s^i \in \mathsf{pre}(\tau_i)$ for $i = 1, 2$ and $s \geq 0$, then $(m_1, m_2) := (k_1, k_2)\ (n_1^1, n_1^2) \cdots (n_s^1, n_s^2) \in \mathsf{pre}(\tau_1 \cap \tau_2)$.

▶ **Example 22.** For α from Example 2 and $\beta = ((a \to b) \to a \to b) \to (a \to b) \to a \to b$, pre-grammar $\mathsf{pre}(\beta)$ consists of the following rewriting rules.

$$\begin{array}{llllllllllll} 14 & := & \lambda 12.13 & 11 & := & \lambda 6.7 \mid 12\ 8 \mid 10 & 7 & := & 12\ 8\ 2 \mid 10\ 4 & 4 & := & 0 \mid 6 \\ 13 & := & \lambda 10.11 \mid 12 & 8 & := & \lambda 0.1 \mid 12\ 8 & 1 & := & 12\ 8\ 2 \mid 10\ 4 & 2 & := & 0 \mid 6 \end{array}$$

After removing obsolete rules we obtain the following set of rules for $\mathsf{pre}(\alpha \cap \beta)$.

$$(10,14) := \lambda(9,12).(8,13) \qquad (8,13) := \lambda(4,10).(5,11) \qquad (5,11) := (4,10)$$

It is easy to see that a term M passes the type checking algorithm for this grammar if and only if $M \equiv_\alpha \lambda xy.y$, which is the only normal term that inhabits both types.

3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).
Editor: Hélène Kirchner; Article No. 5; pp. 5:8–5:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Note that the definition above can be extended in the obvious way to a finite number of intersections, i.e. types of the form $\tau_1 \cap \cdots \cap \tau_n$, for $n \geq 1$, where $\tau_1, \ldots, \tau_n$ are simple types. This corresponds to the set of intersection types of rank 1 [7]. We prove the correctness of our construction for the case of one intersection.

▶ **Theorem 23.** *Consider two simple types τ_1, τ_2, and a term M. Then, one has $M \in \mathsf{Nhabs}(\tau_1) \cap \mathsf{Nhabs}(\tau_2)$ if and only if $(M, (\mathsf{n}(\tau_1), \mathsf{n}(\tau_2)) \hookrightarrow^* \epsilon$, with pre-grammar $\mathsf{pre}(\tau_1 \cap \tau_2)$.*

Proof. Consider two pairs (E_1, m_1) and (E_2, m_2) such that there is some λ-term Q and for $i = 1, 2$: there are placeholders $p_1^i, \ldots, p_r^i \in \mathsf{N}(\tau_i)$ and $\{x_1, \ldots, x_r\} \supseteq \mathsf{FV}(Q)$, such that $E_i = Q\theta_i$ for $\theta_i = [p_1^i/x_1, \ldots, p_r^i/x_r]$; and E_i is either of the form $\lambda x.(Q'\theta_i)$ or $p_j^i\ (Q_1\theta_i) \cdots (Q_n\theta_i)$. It follows from Definition 13 that, if some rule $\mathsf{r}_i \in \mathsf{pre}(\tau_i)$ applies to (E_i, m_i) $(i = 1, 2)$, then both pairs rewrite to a sequence of pairs of equal length, i.e. there is some $s \geq 0$ such that $(E_i, m_i) \hookrightarrow (E_1^i, m_1^i) \cdots (E_{s_i}^i, m_{s_i}^i)$ $(i = 1, 2)$, and for each $j = 1, \ldots, s$ we have that (E_j^1, m_j^1) and (E_j^2, m_j^2) verify the suppositions made on (E_1, m_1) and (E_2, m_2). Furthermore, this guarantees that $(\mathsf{r}_1, \mathsf{r}_2) \in \mathsf{pre}(\tau_1 \cap \tau_2)$, where $(\mathsf{r}_1, \mathsf{r}_2)$ denotes the rule in $\mathsf{pre}(\tau_1 \cap \tau_2)$, built from r_1 and r_2 as described in Definition 21.

If $M \in \mathsf{Nhabs}(\tau_1) \cap \mathsf{Nhabs}(\tau_2)$, then we have by Theorem 15 that $(M, \mathsf{n}(\tau_i)) \hookrightarrow^* \epsilon$, for $i = 1, 2$, in which pairs are assumed to be processed from left to right. The conditions above clearly apply to $(M, \mathsf{n}(\tau_1))$ and $(M, \mathsf{n}(\tau_2))$, and consequently to every other couple of pairs (E_1, m_1) and (E_2, m_2) in these rewriting sequences. One obtains a rewriting sequence for $(M, (\mathsf{n}(\tau_1), \mathsf{n}(\tau_2))) \hookrightarrow^* \epsilon$ using $Q(\theta_1, \theta_2)$ instead of $Q\theta_i$ $(i = 1, 2)$, where $(\theta_1, \theta_2) = [(p_1^1, p_1^2)/x_1, \ldots, (p_r^1, p_r^2)/x_r]$. The proof in the other direction is symmetrical, using the projections on the first or on the second coordinate in each step, in order to obtain rewriting sequences for $(M, \mathsf{n}(\tau_1)) \hookrightarrow^* \epsilon$ or for $(M, \mathsf{n}(\tau_2)) \hookrightarrow^* \epsilon$, respectively. ◀

In order to address sum types of rank 1 we will now define pre-grammars for the union of two languages. Consider rank 1 types τ_1 and τ_2, with sets $\mathsf{N}(\tau_1)$ and $\mathsf{N}(\tau_2)$ for which, without loss of generality, we assume we use two distinct sets of identifiers. Consequently, there is no overlapping of the corresponding grammars.

▶ **Definition 24.** Consider rank 1 types τ_1 and τ_2 and the corresponding identifiers $\mathsf{n}(\tau_i) \in \mathsf{N}(\tau_i)$, for $i = 1, 2$. Let $\mathsf{N}(\tau_1 \cup \tau_2) = \{(\mathsf{n}(\tau_1), \mathsf{n}(\tau_2))\} \cup \mathsf{N}(\tau_1) \cup \mathsf{N}(\tau_2)$. Furthermore, consider the unique rule $\mathsf{n}(\tau_i) := \lambda k_i.n_i$ in $\mathsf{pre}(\tau_i)$ and let $\mathsf{pre}(\tau_i)' = \mathsf{pre}(\tau_i) \setminus \{\mathsf{n}(\tau_i) := \lambda k_i.n_i\}$, for $i = 1, 2$. We define,

$$\mathsf{pre}(\tau_1 + \tau_2) = \{(\mathsf{n}(\tau_1), \mathsf{n}(\tau_2)) := \lambda k_1.n_1; (\mathsf{n}(\tau_1), \mathsf{n}(\tau_2)) := \lambda k_2.n_2\} \cup \mathsf{pre}(\tau_1)' \cup \mathsf{pre}(\tau_2)'.$$

Again, it is straightforward to extend this definition to finite sums of rank 1 types.

▶ **Theorem 25.** *Consider two rank 1 types τ_1, τ_2, and a term M. Then, one has $M \in \mathsf{Nhabs}(\tau_1) \cup \mathsf{Nhabs}(\tau_2)$ if and only if $M \hookrightarrow^* \epsilon$, with pre-grammar $\mathsf{pre}(\tau_1 + \tau_2)$.*

Proof. Straightforward, using Note 10 and Definition 24. ◀

5 Proving Inhabitation Related Problems to be in PSPACE

In this section we present the scheme of an alternating decision algorithm operating on tuples of the form (m, V, i), where $\{m\} \cup V \subseteq \mathsf{N}(\tau)$ and $i \in \mathbb{N}$. The algorithm takes as input a simple type τ, a positive integer depth, a vector/register reg, a function $\mathsf{f} : \mathbb{N} \times \mathsf{pre}(\tau) \times \mathsf{reg} \longrightarrow \mathsf{reg}$ manipulating the contents of reg depending on the values of $(i, \mathsf{r}) \in \mathbb{N} \times \mathsf{pre}(\tau)$, as well as an accepting condition $\mathsf{ac} : \mathsf{reg} \longrightarrow \{\top, \bot\}$. Functions f and ac are supposed to be computable

3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).
Editor: Hélène Kirchner; Article No. 5; pp. 5:9–5:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

in linear time w.r.t. the size of their input. The integer depth is the limit for recursion, such that a loop of the algorithm aborts with failure, whenever this limit is exceeded. In each step, during the execution of the algorithm, one rule $\mathsf{r} \in \mathsf{pre}(\tau)$ is applied to a tuple (m, V, i), and the values in reg are updated to $\mathsf{f}(i, \mathsf{r}, \mathsf{reg})$. Upon a terminated run, condition $\mathsf{ac}(\mathsf{reg}_f)$ determines on success or failure, where reg_f is the configuration of reg at that point. We represent the empty register by $\varnothing$. Furthermore, let $\mathsf{f}_\varnothing$ be such that $\mathsf{f}_\varnothing(i, \mathsf{r}, \mathsf{reg}) = \mathsf{reg}$ for all $(i, \mathsf{r}) \in \mathbb{N} \times \mathsf{pre}(\tau)$, and $\mathsf{ac}_\top$ such that $\mathsf{ac}_\top(\mathsf{reg}) = \top$ for any configuration of reg. Depending on the instantiation of the parameters, the algorithm can be used to show that different inhabitation related problems, such as the emptiness problem, infiniteness, or principal inhabitation, are in PSPACE.

▶ **Definition 26** (PS)**.** Consider a simple type τ, a positive integer depth, a register reg, as well as (linear) functions $\mathsf{f} : \mathbb{N} \times \mathsf{pre}(\tau) \times \mathsf{reg} \longrightarrow \mathsf{reg}$ and $\mathsf{ac} : \mathsf{reg} \longrightarrow \{\top, \bot\}$. Then, $\mathsf{PS}(\tau, \mathsf{depth}, \mathsf{reg}, \mathsf{f}, \mathsf{ac})$ operates as follows, starting with the initial tuple $(m, V, i) = (\mathsf{n}(\tau), \emptyset, 0)$:

- if $i >$ depth the loop aborts with failure;
- otherwise the algorithm:
  - non-deterministically chooses a rule in $\mathsf{r} \in \mathsf{pre}(\tau)$ such that:

$$(m, V) \rightsquigarrow (n_1, V'), \ldots, (n_s, V');$$

  - updates reg according to $\mathsf{f}(i, \mathsf{r}, \mathsf{reg})$;
  - universally applies to $(n_1, V', i+1), \ldots, (n_s, V', i+1)$.

A run is successful if $\mathsf{ac}(\mathsf{reg}_f) = \top$, where reg_f is the final configuration of reg.

Note that, other than by failure, a loop finishes if the rule chosen from $\mathsf{pre}(\tau)$ is such that $s = 0$. In order to show that PS is an alternating polynomial time algorithm w.r.t. the size $|\tau|$ of τ, we start by defining some measures on τ.

▶ **Definition 27.** Given a type τ, let $|\tau| = |\tau|_v + |\tau|_\to$, where $|\tau|_v$ represents the number of occurrences of type variables in τ and $|\tau|_\to$ the number of occurrences of $\to$ in τ. Furthermore, let $|\tau|^+$ and $|\tau|^-$ denote the number of positive occurrences of subformulas and the number of negative subpremises in τ, respectively. Similarly, we use $|\tau|_v^+$ and $|\tau|_v^-$ respectively for the number of positive and negative occurrences of type variables in τ.

The following lemma is a direct consequence of the definitions of $\mathsf{occT}(\tau)$, $\mathsf{N}(\tau)$, and $\mathsf{pre}(\tau)$.

▶ **Lemma 28.** *One has,* $|\tau|^+ \leq |\tau|$, $|\tau|^- \leq |\tau|_\to < |\tau|$, *as well as* $|\mathsf{N}(\tau)| = |\tau|$. *For the number of rules in* $\mathsf{pre}(\tau)$ *we have* $|\mathsf{pre}(\tau)| \leq |\tau|^+ \cdot |\tau|^- + |\tau|_\to$. *Furthermore, the number of elements of* $\mathsf{N}(\tau)$ *occurring in a rule of* $\mathsf{pre}(\tau)$ *is always* $\leq |\tau|^+ + 1$.

▶ **Example 29.** For $\alpha = ((o \to o) \to o \to o) \to o \to o$, we have $|\alpha| = |\alpha|_v + |\alpha|_\to = 6 + 5 = 11 = \mathsf{N}(\alpha)$. Furthermore, $|\alpha|^+ \cdot |\alpha|^- + |\alpha|_\to = 6 \cdot 3 + 5 = 23 \geq 14 = |\mathsf{pre}(\alpha)|$. Finally, the maximum number of identifiers occurring in the rules of $\mathsf{pre}(\alpha)$ is 4 and $4 \leq 6 + 1 = |\alpha|^+ + 1$.

▶ **Proposition 30.** Consider a type τ and constants $k_1, k_2 \in \mathbb{N}$. Suppose that $\mathsf{depth} \leq |\tau|^{k_1}$, $|\mathsf{reg}| \leq k_2 \cdot |\tau|$, and that functions $\mathsf{f} : \mathbb{N} \times \mathsf{pre}(\tau) \times \mathsf{reg} \longrightarrow \mathsf{reg}$ and $\mathsf{ac} : \mathsf{reg} \longrightarrow \{\top, \bot\}$ are computable in linear time w.r.t. the size of their input. Then, $\mathsf{PS}(\tau, \mathsf{depth}, \mathsf{reg}, \mathsf{f}, \mathsf{ac})$ is an alternating polynomial time algorithm w.r.t. $|\tau|$.

Proof. The algorithm is alternating by design. Polynomial time is a consequence of the conditions imposed on the complexity of depth, reg, f and ac. ◀

3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).
Editor: Hélène Kirchner; Article No. 5; pp. 5:10–5:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In the following we establish the relationship between a successful run of algorithm PS with $\mathsf{reg} = \varnothing$, $\mathsf{f}_\varnothing$ and $\mathsf{ac}_\top$, and the existence of a rewriting sequence for $(\mathsf{n}(\tau), \emptyset) \rightsquigarrow^* \epsilon$. Note, that each rewriting sequence of $(\mathsf{n}(\tau), \emptyset) \rightsquigarrow^* \epsilon$ can be represented in the usual way by a unique *derivation tree* t, whose internal nodes are labelled with pairs (m, V, i) and such that all leafs are labelled with ϵ. The root of t is $(\mathsf{N}(\tau), \emptyset, 0)$, and whenever a rule $\mathsf{r} \in \mathsf{pre}(\tau)$ is applied to a pair (m, V), such that $(m, V) \rightsquigarrow (n_1, V'), \ldots, (n_s, V')$, then the corresponding node in t, labelled with (m, V, i), has s children labelled with $(n_1, V', i+1), \ldots, (n_s, V', i+1)$ if $s > 0$, and it has one child labelled with ϵ if $s = 0$. Conversely, we can replace each pair (m, V) in the rewriting sequence by the label (m, V, i) of the corresponding node in t. Furthermore, the height of t is $\mathsf{height}(\mathsf{t}) = \mathsf{depth}(M)$, where $(M, \emptyset) = \mathsf{pair}(\mathsf{N}(\tau), \emptyset)$, corresponding to that rewriting sequence of $(\mathsf{n}(\tau), \emptyset) \rightsquigarrow^* \epsilon$.

▶ **Example 31.** The annotated version of the second rewriting sequence from Example 19 is as follows.

$$\begin{aligned}(10, \emptyset, 0) \quad &\rightsquigarrow \quad (8, \{9\}, 1) \rightsquigarrow (6, \{9\}, 2) \rightsquigarrow (1, \{0, 9\}, 3) \rightsquigarrow (6, \{0, 9\}, 4), (2, \{0, 9\}, 4) \\ &\rightsquigarrow \quad (1, \{0, 9\}, 5), (2, \{0, 9\}, 4) \rightsquigarrow (2, \{0, 9\}, 4) \rightsquigarrow \epsilon.\end{aligned}$$

The corresponding derivation tree has height 6. Also, $\mathsf{depth}(\lambda x_9.x_9(\lambda x_0.x_9(\lambda x_0.x_0)x_0)) = 6$ and $\mathsf{pair}(10, \emptyset) = (\lambda x_9.x_9(\lambda x_0.x_9(\lambda x_0.x_0)x_0), \emptyset)$.

▶ **Lemma 32.** *Consider a type τ and an integer $d > 0$. Then, $\mathsf{PS}(\tau, d, \varnothing, \mathsf{f}_\varnothing, \mathsf{ac}_\top)$ succeeds if and only if there is a rewriting sequence for $(\mathsf{n}(\tau), \emptyset) \rightsquigarrow^* \epsilon$, whose derivation tree t has height $\leq d + 1$. Furthermore, $\mathsf{height}(\mathsf{t}) = \mathsf{depth}(M)$, where $(M, \emptyset) = \mathsf{pair}(\mathsf{n}(\tau), \emptyset)$ for that rewriting sequence of $(\mathsf{n}(\tau), \emptyset) \rightsquigarrow^* \epsilon$.*

Proof. It is easy to see that $\mathsf{PS}(\tau, d, \varnothing, \mathsf{f}_\varnothing, \mathsf{ac}_\top)$ succeeds if and only if there is some tree t with root $(\mathsf{n}(\tau), \emptyset, 0)$ and such that for every node (m, V, i) in that tree, there is a rule $\mathsf{r} \in \mathsf{pre}(\tau)$ such that $(m, V) \rightsquigarrow (n_1, V'), \ldots, (n_s, V')$, and node (m, V, i) has s children $(n_1, V', i+1), \ldots, (n_s, V', i+1)$ if $s > 0$, and one child labelled with ϵ if $s = 0$. The value of i in a node of t labelled with (m, V, i) is $\leq d$, and all leaf nodes are labelled with ϵ. Thus, the height of t is at most $d + 1$. On the other hand, every tree t satisfying the conditions above corresponds to an (annotated) rewriting sequence of $(\mathsf{n}(\tau), \emptyset) \rightsquigarrow^* \epsilon$ and vice-versa. It remains to show that $\mathsf{height}(\mathsf{t}) = \mathsf{depth}(M)$, where $(M, \emptyset) = \mathsf{pair}(\mathsf{n}(\tau), \emptyset)$ for that rewriting sequence of $(\mathsf{n}(\tau), \emptyset) \rightsquigarrow^* \epsilon$. Consider a subtree t' of t, whose root is labelled with a tuple (m, V, i), and the corresponding rewriting sequence of $(m, V) \rightsquigarrow^* \epsilon$. We show by induction on the height of this subtree that $\mathsf{height}(\mathsf{t}') = \mathsf{depth}(M)$, where $(M, \Gamma) = \mathsf{pair}(m, V)$. If $\mathsf{height}(\mathsf{t}') = 1$, then $(m, V) \rightsquigarrow \epsilon$ because $m := k \ \in \mathsf{pre}(\tau)$ and $k \in V$. Thus, $\mathsf{pair}(m, V) = (x_k, \{x_k : \mathsf{t}(k)\})$ and $\mathsf{depth}(x_k) = 1$. If (m, V, i) has $s > 0$ children labelled with $(n_1, V, i+1), \ldots, (n_s, V, i+1)$ because $m := k\ n_1 \cdots n_s \in \mathsf{pre}(\tau)$ and $k \in V$, then $(m, V) \rightsquigarrow (n_1, V), \ldots, (n_s, V)$ and $\mathsf{pair}(m, V) = (x_k\ N_1 \cdots N_s, \{x_k : \mathsf{t}(k)\} \cup \Gamma_1 \cup \cdots \cup \Gamma_s)$, where $(N_i, \Gamma_i) = \mathsf{pair}(n_i, V)$, for $1 \leq i \leq s$. Furthermore, $\mathsf{height}(\mathsf{t}')$ equals 1 plus the maximum of the heights of the subtrees rooted in $(n_1, V, i+1), \ldots, (n_s, V, i+1)$, while $\mathsf{depth}(x_k\ N_1 \cdots N_s)$ equals 1 plus the maximum of the depths of $N_1, \ldots, N_s$. Thus, the result follows from the induction hypothesis. Finally, suppose that (m, V, i) has one child labelled with $(n, V \cup \{k\}, i+1)$ because $m := \lambda k.n \in \mathsf{pre}(\tau)$. Then, $\mathsf{height}(\mathsf{t}')$ equals 1 plus the height of the subtree rooted in $(n, V \cup \{k\}, i+1)$. On the other hand, $\mathsf{pair}(m, V) = (\lambda x_k.N, \Gamma \setminus \{x_k : \mathsf{t}(k)\})$, where $(N, \Gamma) = \mathsf{pair}(n, V \cup \{k\})$. We have $\mathsf{depth}(\lambda x_k.N) = 1 + \mathsf{depth}(N)$ and consequently the result follows from the induction hypothesis. ◀

3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).
Editor: Hélène Kirchner; Article No. 5; pp. 5:11–5:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

5.1 Emptiness

In the following we reprove the well-known result [14, 16], stating that the emptiness problem for TA_λ is in PSPACE, by instantiation of algorithm PS. We say that a derivation tree t corresponding to a particular rewriting sequence of $(\mathsf{N}(\tau), \emptyset) \rightsquigarrow^* \epsilon$ has a *repetition*, if and only if there is a branch in t containing two nodes with labels (m, V, i) and (m, V, i') such that $i \neq i'$. Furthermore, we have $(\mathsf{N}(\tau), \emptyset) \rightsquigarrow^* \epsilon$ if and only if there is some rewriting sequence for that fact, whose derivation tree t contains no repetition. By Lemma 32 it suffices to execute algorithm PS with a value for depth that guarantees that every derivation tree with depth $> \mathsf{depth}$ has a repetition. For this, we define $\mathsf{D}(\tau) = |\tau|^+ \cdot |\tau|^-$.

▶ **Proposition 33.** $\mathsf{PS}(\tau, \mathsf{D}(\tau), \varnothing, \mathsf{f}_\varnothing, \mathsf{ac}_\top)$ succeeds if and only if $\mathsf{Nhabs}(\tau) \neq \emptyset$.

Proof. The limit $\mathsf{D}(\tau)$ is chosen so that for a pair (m, V, d) with $d > \mathsf{D}(\tau)$, there is a repetition in the corresponding derivation tree t. Since there are at most $|\tau|^+$ different identifiers m and at most $|\tau|^-$ different sets V, there has to be a repetition in the branch leading from the root of t to (m, V, d). Thus, the result follows from Lemma 32. ◀

5.2 Counting

In [2], Ben-Yelles defined a counting algorithm that answers the question of how many normal inhabitants a given type τ has. The main focus, when asking this question, is usually on determining if $\mathsf{Nhabs}(\tau)$ is empty, finite or infinite. In [10], the infiniteness of $\mathsf{Nhabs}(\tau)$ was shown to be PSPACE complete. In the following, we show how algorithm PS can be instantiated in order to prove this problem to be in PSPACE.

We already argued that $\mathsf{Nhabs}(\tau) \neq \emptyset$ if and only if there is some derivation tree for $(\mathsf{N}(\tau), \emptyset) \rightsquigarrow^* \epsilon$ of height $\leq \mathsf{D}(\tau) + 1 = |\tau|^+ \cdot |\tau|^- + 1$. In the following we establish a lower limit $\mathsf{d}(\tau)$, such that the existence of a tree of height $> \mathsf{d}(\tau)$ guarantees that $|\mathsf{Nhabs}(\tau)| = \infty$. Consider a tree t containing a branch with two nodes $n = (m, V, d)$ and $n' = (m, V', d')$, with $d < d'$. Then, $V \subseteq V'$ and one can construct a new derivation tree by replacing in t the subtree $\mathsf{t}_{n'}$ rooted in n' by the subtree t_n rooted in n, changing every label (m'', V'', i) to $(m'', V'' \cup V', i + (d' - d))$. Repeating this process, it is possible to construct an infinite number of derivation trees of increasing height. Thus, $\mathsf{Nhabs}(\tau)$ is infinite. On the other hand, for $\mathsf{d}(\tau) = |\tau|^+$, if t has some branch of length $> \mathsf{d}(\tau)$, then this branch contains necessarily two such nodes n and n'. Now, suppose that the height of t is $\leq \mathsf{d}(\tau)$ and that some branch in t contains two nodes n and n' as above. Then $d, d' \leq \mathsf{d}(\tau)$ and $0 < (d' - d) < \mathsf{d}(\tau)$. Then, it is clear that repeating the process described above, at some point, one obtains a derivation tree of height D, with $\mathsf{d}(\tau) \leq D \leq \mathsf{D}(\tau)$, as long as $|\tau|^- > 1$. We conclude that for τ, such that $|\tau|^- > 1$, we have $\mathsf{Nhabs}(\tau) = \infty$ if and only if there is some derivation tree of height D, with $\mathsf{d}(\tau) + 1 \leq D \leq \mathsf{D}(\tau) + 1$.

▶ **Lemma 34.** *If* $|\tau|^- \leq 1$, *then* $\mathsf{Nhabs}(\tau) \neq \emptyset$ *iff* $\tau = a \to a$, *for which* $|\mathsf{Nhabs}(\tau)| = 1$.

Proof. If $|\tau|^- = 0$, then $\tau = a$ and $\mathsf{Nhabs}(\tau) = \emptyset$. For $|\tau|^- = 1$, it is easy to show, by induction on the number of implications in τ, that τ is of the form $(a_1 \to \cdots \to a_n \to b) \to a$, which is inhabited exactly if $n = 0$ and $a = b$. ◀

▶ **Proposition 35.** The counting problem for $\mathsf{Nhabs}(\tau)$ is in PSPACE.

Proof. If $|\tau|^- \leq 1$, then $|\mathsf{Nhabs}(\tau)| = 1$ if $\tau = a \to a$, and $|\mathsf{Nhabs}(\tau)| = \emptyset$ otherwise.

If $|\tau|^- > 1$, then $|\mathsf{Nhabs}(\tau)| = \infty$ if and only if there is some derivation tree of height D such that $\mathsf{d}(\tau) < D \leq \mathsf{D}(\tau) + 1$. This can be checked by instantiating algorithm PS as follows:

3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).
Editor: Hélène Kirchner; Article No. 5; pp. 5:12–5:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- $\mathsf{depth} = \mathsf{D}(\tau)$;
- $|\mathsf{reg}| = 1$ and $\mathsf{reg}[0] = 0$;
- $\mathsf{f}(i, \mathsf{r}, \mathsf{reg}) = (\text{IF } (i == \mathsf{d}(\tau)) \text{ THEN } \mathsf{reg}[0] := 1)$;
- $\mathsf{ac}(\mathsf{reg}) = (\mathsf{reg}[0] == 1)$.

If the algorithm succeeds, then $|\mathsf{Nhabs}(\tau)| = \infty$. Otherwise, according to Lemma 32 we can run $\mathsf{PS}(\tau, \mathsf{d}(\tau) - 1, \varnothing, \mathsf{f}_\varnothing, \mathsf{ac}_\top)$ in order to check if $\mathsf{Nhabs}(\tau)$ is finite, but not empty. ◀

5.3 Principal Inhabitation

We now use our algorithm to address the closely related problem of principal inhabitation, which although more complex, is still PSPACE-complete [6]. The *principal inhabitation problem* is about the existence of a normal inhabitant M of τ, such that τ is the principal type of M. A term M is a principal inhabitant of τ, if $\vdash M : \tau$ and if every type σ, such that $\vdash M : \sigma$, is an instance of τ. Then, τ is called the principal type of M. When searching for principal inhabitants, it is sufficient to consider principal inhabitants in long normal form, for which a characterisation was given in [4] in terms of proof trees, in the context of the formula-tree method. In this section we instantiate the algorithm PS to decide principal inhabitation, based on that characterisation. This characterisation was used in [1] to define deterministic principal inhabitation machines for normal inhabitants obtained from pre-grammars, following the formalism of Schubert et al. An inhabitant M of a type is called long, if every variable occurrence, which is in function position, is given as many arguments as allowed by its type. It is straightforward to change the definition of $\mathsf{pre}(\tau)$ in order to apply exactly to the set of long normal inhabitants of τ. For this, it is sufficient to drop in $\mathsf{pre}(\tau)$ all rules of the form $m := k\ n_1 \cdots n_s$ such that $\mathsf{lab}(m) \neq \mathsf{var}$. The pre-grammar thereby obtained is denoted by $\mathsf{preL}(\tau)$ and verifies the following. If $m^+ \in \mathsf{N}(\tau)$ and $\mathsf{lab}(m) = k \to n$, then there is exactly one rule for m in $\mathsf{preL}(\tau)$, which is $m := \lambda k.n$. If $m^+ \in \mathsf{N}(\tau)$ and $\mathsf{lab}(m) = \mathsf{var}$, then all rules for m are of the form $m := k\ n_1 \cdots n_s$ $(s \geq 0)$, such that $\mathsf{t}(k) = \mathsf{t}(n_1) \to \cdots \to \mathsf{t}(n_s) \to \mathsf{t}(n)$, where $\mathsf{lab}(n) = \mathsf{var}$ and $\mathsf{t}(m) = \mathsf{t}(n)$, i.e. m and n are different occurrences of the same type variable. For convenience we denote n by $\mathsf{tail}(k)$. Note that $\mathsf{tail}(k)$ is the root of the (unary) tree in graph $T(\tau)$, that contains k.

▶ **Example 36.** The pre-grammar $\mathsf{preL}(\alpha)$ for the set of long normal inhabitants of α from Example 2 is the following.

10	:=	$\lambda 9.8$	6	:=	$\lambda 0.1$	2	:=	9 6 2 | 4 | 0
8	:=	$\lambda 4.5$	5	:=	9 6 2 | 4 | 0	1	:=	9 6 2 | 4 | 0

The approach in [4] establishes that, initially all occurrences of type variables in τ have to be made different. Here, this is already achieved by the association of different identifiers to different occurrences of subtypes. During the search of an inhabitant, the application of a rule $m := k\ n_1 \cdots n_s$, as described above, forces that m and n must represent the same type variable in any type of that inhabitant[3]. When instantiating the algorithm, this information will be kept in register reg and the execution will only be successful if all occurrences of the same type variable are unified. The remaining condition in the characterisation of principal inhabitants in [4] states that all composed negative subpremises have to be used. This information will also be stored in reg. We denote the number of composed negative subpremises in τ by $|\tau|_c^-$ and define $\mathsf{P}(\tau) = |\tau|^+ \cdot |\tau|^- \cdot |\tau|_v \cdot |\tau|_c^-$ for the limit of recursion.

[3] Note that, limiting the search to long inhabitants avoids dealing with the unification of composed types, but restricts this operation to occurrences of type variables.

3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).
Editor: Hélène Kirchner; Article No. 5; pp. 5:13–5:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

For practicality, we convention that type variables and negative subpremises have identifiers $0, \ldots, |\tau|_v - 1$, and $|\tau|_v, \ldots, |\tau|_v + |\tau|_c^- - 1$, respectively[4]. Finally, we denote by $\mathsf{var}(\tau)$ the number of different type variables in τ.

▶ **Example 37.** In our running example there are three negative subpremises, respectively with identifier 9, 4 and 0. We have $\mathsf{tail}(9) = 3$, $\mathsf{tail}(4) = 4$ and $\mathsf{tail}(0) = 0$. Only $\mathsf{t}(9)$ is composed. Thus, $|\alpha|_c^- = 1$ and $\mathsf{P}(\alpha) = 6 \cdot 3 \cdot 6 \cdot 1 = 108$, while $\mathsf{var}(\alpha) = 1$.

Now, we define a function $\mathsf{f_P}$ that stores the information concerning unification of different type variable occurrences and the use of composed negative subpremises in reg. For this, initially the identifier of each variable is stored in the first $|\tau|_v$ positions of reg, each representing its own class, which at that point is a singleton. The number of different classes, which is initially $|\tau|_v$, is stored in the last position of reg and decreased whenever two classes are merged. In this case, all elements (positions in reg) of these classes are represented by the same identifier. The intermediate positions of reg are used to register the application of composed negative subpremises.

$\mathsf{f_P}(i, \mathsf{r}, \mathsf{reg})$:
IF $\mathsf{r} == (m := k\ n_1 \cdots n_s)$ THEN
- $n := \mathsf{tail}(k)$;
- MIN $:= min(\mathsf{reg}[m], \mathsf{reg}[n])$;
- MAX $:= max(\mathsf{reg}[m], \mathsf{reg}[n])$;
- IF MIN $\neq$ MAX THEN
  - $\mathsf{reg}[|\tau|_v + |\tau|_c^-] := \mathsf{reg}[|\tau|_v + |\tau|_c^-] - 1$;
  - FOR $(j = 0$ TO $|\tau|_v - 1)$ DO
    - IF $(\mathsf{reg}[j] ==$ MAX$)$ THEN $\mathsf{reg}[j] :=$ MIN;
- IF $(s > 0)$ THEN $\mathsf{reg}[k] := 1$;

In order to determine success or failure of a run, function $\mathsf{ac_P}$ checks if all $|\tau|_c^-$ composed have been used and if there are exactly as many classes of occurrences of type variables as there are different type variables in τ.

$\mathsf{ac_P}(\mathsf{reg})$:
- COUNT $:= 0$;
- FOR $(j = |\tau|_v$ TO $|\tau|_v + |\tau|_c^- - 1)$ DO
  - COUNT $:=$ COUNT $+ \mathsf{reg}[j]$;
- IF $(\text{COUNT} \neq |\tau|_c^-)$ THEN (RETURN $\bot$)
  ELSE (RETURN $(\mathsf{reg}[|\tau|_v + |\tau|_c^-] == \mathsf{var}(\tau))$);

▶ **Proposition 38.** The principal inhabitation problem for $\mathsf{Nhabs}(\tau)$ is in PSPACE.

Proof. This can be checked by instantiating algorithm PS as follows:
- $\mathsf{depth} = \mathsf{P}(\tau)$;
- $|\mathsf{reg}| = |\tau|_v + |\tau|_c^- + 1$, $\mathsf{reg}[j] = j$ (for $0 \leq j \leq |\tau|_v - 1$), $\mathsf{reg}[j] = 0$ (for $|\tau|_v \leq j \leq |\tau|_v + |\tau|_c^- - 1$), and $\mathsf{reg}[|\tau|_v + |\tau|_c^-] = |\tau|_v$;
- $\mathsf{f} = \mathsf{f_P}$ and $\mathsf{ac} = \mathsf{ac_P}$.

Function $\mathsf{f_P}$ registers (during the execution of PS) all necessary information for deciding on principality in reg, which is checked by $\mathsf{ac_P}$ after completion of a run. Thus, there is

[4] This convention does not hold for the composed negative subpremise with identifier 9 in our example.

3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).
Editor: Hélène Kirchner; Article No. 5; pp. 5:14–5:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

some principal inhabitant of τ iff there is a successful run using a limit of recursion, possibly bigger than $\mathsf{depth} = \mathsf{P}(\tau)$. We consider such a successful run (for a principal inhabitant), and the corresponding derivation tree $\mathtt{t}$. Finally, we argue that it is possible to obtain a new derivation tree from $\mathtt{t}$, corresponding to a successful run, within the established limit $\mathsf{P}(\tau)$. Consider any node $n = (m, V, i)$ in $\mathtt{t}$. We associate to node n the number Eq_n of equivalence classes, as well as the set C_n of negative composed subpremises, that are induced by the derivation steps in the subtree rooted in n. There is a repetition in a branch of $\mathtt{t}$ if it contains two nodes $n = (m, V, i)$ and $n' = (m, V, i')$ with $i < i'$, such that $Eq_n = Eq_{n'}$ and $C_n = C_{n'}$. If that is the case, one can replace the subtree rooted in n by the smaller subtree rooted in n', obtaining a tree still corresponding to a successful run. Since $i < i'$ implies that $Eq_n \leq Eq_{n'} \leq |\tau|_v$, as well as $|\tau|_c^- \geq |C_n| \geq |C_{n'}|$, there is a repetition in every branch of length $\geq \mathsf{P}(\tau)$. Consequently, the process described above can be repeated until one obtains a derivation tree, thus a successful run, within the limit established for depth. ◀

6 Conclusions

In this paper we presented a unifying framework to study type inhabitation related problems and their complexity, using the notion of pre-grammar. From the pre-grammar of a type we obtained different methods to address several inhabitation related problems. A scheme for a decision algorithm was given, which we instantiated to decide emptiness, counting and principal inhabitation. Since each instantiation produces a polynomial time alternating algorithm, this also shows these problems to be in PSPACE. For principal inhabitants we focused on terms in long normal form, for which we used a simplified and smaller set of rules. In a similar way, one could define different sets of pre-grammar rules, corresponding to particular subclasses of terms, such as terms in total discharge form, term-schemes, etc. This is left for future work, where we also would like to further develop the study of closure properties, in particular study an instantiation of our algorithm for union types of rank 1.

References

1. Sandra Alves and Sabine Broda. Inhabitation machines: determinism and principality. In *Applications, NCMA 2017*, pages 57–70, 2017.
2. Ch. Ben-Yelles. *Type Assignment in the Lambda-Calculus: Syntax and Semantics*. PhD thesis, University College of Swansea, September 1979.
3. S. Broda and L. Damas. Counting a type's (principal) inhabitants. *Fundam. Inform.*, 45(1-2):33–51, 2001.
4. S. Broda and L. Damas. On long normal inhabitants of a type. *J. Log. and Comput.*, 15:353–390, June 2005.
5. M.W. Bunder. Proof finding algorithms for implicational logics. *Theoretical Computer Science*, 232(1–2):165–186, 2000.
6. Andrej Dudenhefner and Jakob Rehof. The complexity of principal inhabitation. In *Computation and Deduction, FSCD 2017*, volume 84 of *LIPIcs*, pages 15:1–15:14, 2017.
7. Silvia Ghilezan. Inhabitation in intersection and union type assignment systems. *J. Log. Comput.*, 3(6):671–685, 1993.
8. J.R. Hindley. *Basic Simple Type Theory*, volume 42 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1997.
9. R. Hindley. The principal type-scheme of an object in combinatory logic. *Trans. Amer. Math. Soc*, 146:29–60, December 1969.
10. S. Hirokawa. Infiniteness of proof (α) is polynomial-space complete. *Theor. Comput. Sci.*, 206(1-2):331–339, 1998.

3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).
Editor: Hélène Kirchner; Article No. 5; pp. 5:15–5:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

11 W.A. Howard. The formulas-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 Y. Komori and S. Hirokawa. The number of proofs for a BCK-formula. *J. Symb. Log.*, 58(2):626–628, 1993.

13 Aleksy Schubert, Wil Dekkers, and Hendrik Pieter Barendregt. Automata theoretic account of proof search. In *CSL 2015*, pages 128–143, 2015.

14 R. Statman. Intuitionistic propositional logic is polynomial-space complete. *Theor. Comput. Sci.*, 9:67–72, 1979.

15 M. Takahashi, Y. Akama, and S. Hirokawa. Normal proofs and their grammar. *Information and Computation*, 125(2):144–153, 1996.

16 P. Urzyczyn. Inhabitation in typed lambda-calculi (a syntactic approach). In *TLCA'97*, volume 1210 of *LNCS*, pages 373–389. Springer, 1997.

3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).
Editor: Hélène Kirchner; Article No. 5; pp. 5:16–5:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany