Pregrammars and Intersection Types

Sabine Broda
CMUP, Departamento de Ciência de Computadores, Faculdade de Ciências,
University of Porto, Portugal

Abstract

A representation of intersection types in terms of pregrammars is presented. Pregrammar based rewriting relations, corresponding respectively to type checking and inhabitation are defined and the latter is used to implement a Wajsberg/Ben-Yelles style alternating semi-decision algorithm for inhabitation. The usefulness of the framework is illustrated by revisiting and partially extending standard inhabitation related results for intersection types, as well as establishing new ones. It is shown how the notion of bounded multiset dimension emerges naturally and the relation between the two settings is clarified. A meaningful rank independent superset of the set of rank 2 types is identified for which EXPSPACE-completeness for inhabitation as well as for counting is proved. Finally, a standard result on negatively non-duplicated simple types is extended to intersection types.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Logic; Theory of computation → Lambda calculus

Keywords and phrases Intersection Types, Pregrammars, Inhabitation

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.14

Funding The author was partially supported by CMUP, which is financed by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under the project with reference UIDB/00144/2020.

1 Introduction

1.1 Contribution and Related Work

Intersection type systems [12, 13] play an important role in the theory of typed λ-calculus [5]. They characterise exactly the set of strongly normalising terms and are closely related to the model theory of λ-calculus [33, 25, 14]. Applications in programming language theory cover a variety of diverse topics including the design of programming languages [36], program analysis and model checking [30, 31, 34], synthesis [15, 21], and related systems such as refinement and union types [22, 19, 20]. But the enormous expressive power of intersection types also comes with a major drawback. Standard type theoretic problems such as type checking and inhabitation are undecidable in general [5]. For inhabitation, undecidability is known for about twenty years [42]. More recently [37, 38] the problem was shown to be equivalent to the undecidable problem of λ-definability [32, 29]. Over the years attempts have been made in various directions with the purpose of identifying and studying the complexity of decidable fragments or variants of the system. Those include restrictions by rank [28, 27, 43], restrictions of admissible type inference rules [26, 35, 11], calculi of bounded dimension [16, 18], as well as systems of non-idempotent intersection types [10, 9]. In terms of rank the dividing line between decidable and undecidable cases was established in [27, 43] and complexity results were given for the decidable cases. As such, inhabitation was shown to be PSPACE-complete for rank 1, EXPSPACE-complete for rank 2, and undecidable for types of rank ≥ 3. Recently, the notion of dimension was introduced [16, 17] measuring intersection introduction as a resource. The inhabitation problem was shown to be decidable and EXPSPACE-complete in

bounded multiset dimension, exposing thereby a rank independent calculus with a decidable inhabitation problem. In particular, it was shown to subsume the same result for rank 2 inhabitation.

The present work borrows some inspiration from [40] and aims to contribute to this line of research providing a simple tool for addressing inhabitation related problems. Lower bounds of specific problems are typically established using reductions from problems for which a bound (resp. undecidability) is already known. For upper bounds the standard procedure is based on the implementation of a Wajsberg/Ben-Yelles style search algorithm [6, 23, 11, 43]. The algorithm looks for inhabitants in β-normal form and operates on a set of pairs, each consisting of a context and a type. Depending on the rule used in each step the algorithm manipulates the set of pairs accordingly. The present work was motivated by the desire to provide a simple formalisation of this transformation process, expounding changes in each execution step in a clear and organised manner. The primary goal was to facilitate reasoning about inhabitation problems for intersection types, as well as to provide a clean framework in which proofs can be presented. Additionally, a graphical representation of intersection types exposes their fine underlying structure contributing to the end of clarification. That insight enables us to add further to the knowledge on decidable fragments of the intersection type system. From a practical point of view, we believe this work, and in particular pregrammar based implementations of a Wajsberg/Ben-Yelles search algorithm, to be of importance for applications in proof/inhabitant search based program synthesis in systems with intersection [21].

Picking up on work in [1, 2, 3] we here extend the notion of pregrammar to the entire system of intersection types. In [1] pregrammars were introduced in the context of (principal) inhabitation of simple types. Later [2] PSPACE upper bounds were (re-)proved for different inhabitation related problems for simple types in the framework of pregrammars. A first attempt to generalise those concepts to apply to intersection types was made in [3] and covered a restricted fragment of rank 2 types, therein denoted by $\mathcal{T}_2^-$. Additionally, the notion of pregrammar was extended to apply to the set of finite intersections of $\mathcal{T}_2^-$-types, which properly contains the set of strict intersection types [44] of rank 2 (but not all rank 2 types). For $\mathcal{T}_2^-$-types it is sufficient to consider a type inference system without the intersection introduction rule ($\cap$I), which was the approach taken in [3]. Intersections of $\mathcal{T}_2^-$-types introduce new intersections at the top level only and could therefore be handled by considering products of pregrammars. Neither considering a system without rule ($\cap$I), nor using products of pregrammars, works if one wants to cover the entire system, for which a different approach has to be taken. In the present work we consider a type inference system for terms in β-normal form, equivalent to that in [43], with rule ($\cap$I) but without an explicit intersection elimination rule ($\cap$E). This enables us to avoid the restriction to strict types, which was the approach taken in [18], and still handle the case of application in our search algorithm strictly syntax driven, contrasting with the $\vdash$-based condition in [43].

1.2 Outline

The paper is organised as follows. Section 2 contains the necessary definitions and results on the system of intersection types and sets the ground for subsequent developments. The definition of pregrammars for intersection types is in Section 3. In Sections 4 and 5 we define sound and complete rewriting relations based on pregrammars, corresponding respectively to type checking and inhabitation. Transformation in contexts during proof search are therein captured in terms of operations (update and replication) on tuples of integers. A Wajsberg/Ben-Yelles style alternating semi-decision algorithm for inhabitation is defined in

Section 6. In the remaining of that section the usefulness of our framework is demonstrated by revisiting and partially extending standard inhabitation related results, as well as establishing new ones. It is shown how the notion of bounded multiset dimension emerges naturally in this setting and the relation between the two is clarified. A meaningful rank independent superset of the set of rank 2 types is identified for which EXPSPACE-completeness is proved. Making use of the aforementioned operations on tuples, EXPSPACE-completess of counting in this fragment is also shown. Finally, a standard result on negatively non-duplicated simple types is extended to intersection types. A compilation of complexity results for different families of types can be found in the conclusions.

It is important to note that the majority of the discussed results can be obtained by the classical method. Nonetheless, it is our belief that the work in this paper constitutes further a step towards a better understanding of inhabitation in the intersection type system by highlighting how intersections at different depth and of different polarity contribute differently to the complexity of the inhabitation problem.

2 Preliminaries

We consider the basic system of intersection types, with no constants nor subtyping, cf. [43]. Type variables are ranged over by $a, b, c, \ldots$ and arbitrary types by lower-case Greek letters. The set of simple types. i.e. types without occurrences of the intersection operator, is denoted by $\mathcal{T}$.

▶ **Definition 1** (Intersection Types). $\mathcal{T}_\cap \ni \sigma, \tau ::= a \mid \sigma \to \tau \mid \sigma_1 \cap \cdots \cap \sigma_n \ (n \geq 2)$.

We write $\sigma \in \tau$, if σ is an occurrence of a subtype of τ, where the notion of occurrence is the usual one, cf. [23, Definition 9A2]. For instance, type $\alpha = (a \to a) \to a \to a$ has three different subtypes α, $a \to a$ and a, which have respectively 1, 2 and 4 occurrences in α. We assume that the intersection operator is associative, and that it binds stronger than $\to$. Furthermore, we always identify maximal subtypes containing intersections, that is, when we write $\tau_1 \cap \cdots \cap \tau_n$, then none of the τ_i is itself an intersection. The notion of rank stems from [28] and is defined by $\mathsf{rank}(\tau) = 0$ if $\tau \in \mathcal{T}$, by $\mathsf{rank}(\sigma \to \tau) = max(1 + \mathsf{rank}(\sigma), \mathsf{rank}(\tau))$ if $\mathsf{rank}(\sigma) + \mathsf{rank}(\tau) > 0$, and by $\mathsf{rank}(\tau_1 \cap \cdots \cap \tau_n) = max(1, \mathsf{rank}(\tau_1), \ldots, \mathsf{rank}(\tau_n))$ for $n \geq 2$. Considering the syntactic tree of a type θ, its rank equals the maximal number of implications, to which an intersection occurs to the left, plus one. In general, given a particular occurrence of a subtype $\tau \in \theta$ we denote by $\mathsf{depth}(\tau, \theta)$ the number of implications in θ to which τ occurs to the left and call it the depth of τ in θ. Then, we have that $\mathsf{rank}(\theta) \geq r \geq 1$ if and only if there exists an intersection at depth $\geq r - 1$. Formally, depth can be defined (top-down) by $\mathsf{depth}(\theta, \theta) = 0$, by $\mathsf{depth}(\sigma, \theta) = \mathsf{depth}(\sigma \to \tau, \theta) + 1$ and $\mathsf{depth}(\tau, \theta) = \mathsf{depth}(\sigma \to \tau, \theta)$, and by $\mathsf{depth}(\tau_i, \theta) = \mathsf{depth}(\tau_1 \cap \cdots \cap \tau_n, \theta)$ for $1 \leq i \leq n$ and $n \geq 2$. Another meaningful notion is the degree of an occurrence $\tau \in \theta$, which measures the number of consecutive implications, ignoring intersections in that counting, to which τ occurs to the right. Formally, we have $\mathsf{deg}(\theta, \theta) = 0$, if $\mathsf{deg}(\sigma \to \tau, \theta) = n$ then $\mathsf{deg}(\sigma, \theta) = 0$ and $\mathsf{deg}(\tau, \theta) = n + 1$, and finally $\mathsf{deg}(\tau_i, \theta) = \mathsf{deg}(\tau_1 \cap \cdots \cap \tau_n, \theta)$ for $1 \leq i \leq n$ and $n \geq 2$. In particular, θ is a strict intersection type [44] if and only if all intersections in θ occur at degree 0.

The notion of polarity of an occurrence $\tau \in \theta$ is defined as usual by: τ occurs positively in τ; if τ occurs positively (resp. negatively) in θ then it occurs positively (resp. negatively) in $\sigma \to \theta$ and in $\tau_1 \cap \cdots \cap \theta \cap \cdots \cap \tau_n$, and negatively (resp. positively) in $\theta \to \sigma$. As an alternative, one can define that τ occurs positively in θ if $\mathsf{depth}(\tau, \theta)$ is even, and negatively otherwise.

$$\frac{\Gamma \cup \{x:\sigma\} \vdash_\cap N:\tau}{\Gamma \vdash_\cap \lambda x.N:\sigma\to\tau}\,(\mathrm{I}\to) \qquad \frac{\Gamma \vdash_\cap xN_1\cdots N_s:\sigma\to\tau \quad \Gamma\vdash_\cap N_{s+1}:\sigma}{\Gamma\vdash_\cap xN_1\cdots N_sN_{s+1}:\tau}\,(\mathrm{E}\to)$$

$$\frac{}{\Gamma\cup\{x:\tau\}\vdash_\cap x:\tau}\,(\mathsf{var}) \qquad \frac{\Gamma\vdash_\cap M:\tau_1 \;\cdots\; \Gamma\vdash_\cap M:\tau_n}{\Gamma\vdash_\cap M:\tau_1\cap\cdots\cap\tau_n}\,(\mathrm{I}\cap) \qquad \frac{\Gamma\vdash_\cap M:\tau_1\cap\cdots\cap\tau_n}{\Gamma\vdash_\cap M:\tau_i \;\; (1\le i\le n)}\,(\mathrm{E}\cap)$$

Figure 1 Intersection Type Assignment for Terms in Normal Form.

Every type τ can be uniquely written as $\tau=\tau_1\to\cdots\to\tau_n\to\theta$ $(n\ge 0)$, where θ is a type variable or an intersection $\theta_1\cap\cdots\cap\theta_m$ $(m\ge 2)$. If $n\ge 1$, then $\tau_1,\ldots,\tau_n$ are called the *arguments* of τ. An occurrence of σ in τ is called a *negative subpremise* of τ iff it is the argument of a positive occurrence of a subtype in τ. We write $\sigma\preceq_\cap\beta$ and say that σ is a *component* of β if and only if $\beta=\beta_1\cap\cdots\cap\beta_n$ and $\sigma=\beta_i$, for some $i\in[1..n]$ and $n\ge 1$. By our convention on intersections this implies that a component is never an intersection itself.

▶ **Example 2.** Let $\alpha=\alpha_1\cap\alpha_2$, where $\alpha_1=1\to(0\to 0)\cap(1\to 1)\to(1\to 0)\to 0$ and $\alpha_2=1\to(1\to 0)\to(0\to 1)\to 0$, and 0 and 1 denote type variables. This type belongs to the family used in the reduction of the halting problem for bus machines to rank 2 inhabitation in [43], showing thereby EXPSPACE-hardness of the problem. Furthermore, consider $\beta=o\cap\beta_1\to o$, where $\beta_1=((o\to(o\to o)\cap(o\to o))\to o)\to(o\to o)\cap(o\to o)$. We have $\mathsf{rank}(\alpha)=2$ and $\mathsf{rank}(\beta)=4$. Moreover, α is strict, while β is not, since both intersections in β_1 occur at degree 1 in β. Types α and β will be our running examples throughout the paper.

We denote λ-terms by $M,N,\ldots$, which are built from an infinite countable set of term variables $\mathcal{V}$. We identify terms modulo α-equivalence. For type assignment we consider a system equivalent to the one presented in [27, 43], but restricted to β-normal forms, which is the usual choice when addressing inhabitation related problems. Unless stated otherwise, all λ-terms considered in the remainder of this paper are supposed to be in β-normal form.

A (consistent) *context* is a finite set Γ of *declarations* of the form $x:\sigma$, where $x\in\mathcal{V}$ and $\sigma\in\mathcal{T}_\cap$, such that all term variables occurring in Γ are distinct from each other. The *domain* of Γ, denoted by $\mathsf{dom}(\Gamma)$, is the set of term variables occurring in Γ. For $(x:\sigma)\in\Gamma$, let $\Gamma(x)=\sigma$. Furthermore, $\mathsf{Types}(\Gamma)=\{\,\sigma\mid x:\sigma\in\Gamma\,\}$. The rules of the type assignment system are given in Figure 1. Formulas (judgements) in derivations are of the form $\Gamma\vdash_\cap M:\theta$. Symbol $\vdash$ will be used for the inference of formulas in a second equivalent system without explicit $\cap$-elimination, whose rules are given in Figure 2. We say that type $\theta\in\mathcal{T}_\cap$ can be assigned to a normal form M in context Γ, and write $\Gamma\vdash_\cap M:\theta$, if and only if this formula can be obtained by applying the rules in Figure 1 a finite number of times. For the parameters in these rules we suppose that $s\ge 0$, $n\ge 2$ and $i\in[1..n]$. Similarly, we write $\Gamma\vdash M:\theta$ if that formula can be derived by the inference rules in Figure 2. We draw attention to the fact that for $\vdash$, dropping the $\cap$-elimination rule was achieved by, in some sense, incorporating quasi-order $\sqsubseteq$ in [35, Lemma 20] directly into rules (var) and (E$\to$), and facilitates handling types with intersections at degree ≥ 1, i.e. non-strict types.

For $\triangleright\in\{\vdash_\cap,\vdash\}$, it is easy to verify that $\Gamma\triangleright M:\theta$ implies that the set of term variables in Γ contains the set of free variables in M, i.e. $\mathsf{dom}(\Gamma)\supseteq\mathsf{FV}(M)$. A derivation of a formula $\Gamma\triangleright M:\theta$ can be represented as a derivation tree Π, in which all nodes are labelled by formulas, such that $\Gamma\triangleright M:\theta$ is the root of Π, every internal node is obtained from its children by one of the type assignment rules different from (var), and every leaf is labelled with an instance of (var).

$$\frac{}{\Gamma \cup \{x:\tau\} \vdash x:\tau_i \quad (\tau_i \preceq_\cap \tau)}\,(\mathsf{var}) \qquad \frac{\Gamma \cup \{x:\sigma\} \vdash N:\tau}{\Gamma \vdash \lambda x.N:\sigma\to\tau}\,(\mathrm{I}{\to})$$

$$\frac{\Gamma \vdash xN_1\cdots N_s:\sigma\to\tau \quad \Gamma\vdash N_{s+1}:\sigma}{\Gamma\vdash xN_1\cdots N_sN_{s+1}:\tau_i \quad (\tau_i\preceq_\cap\tau)}\,(\mathrm{E}{\to}) \qquad \frac{\Gamma\vdash M:\tau_1 \;\cdots\; \Gamma\vdash M:\tau_n}{\Gamma\vdash M:\tau_1\cap\cdots\cap\tau_n}\,(\mathrm{I}{\cap})$$

Figure 2 Type Assignment without an explicit ∩-elimination rule.

▶ **Lemma 3.** *We have $\Gamma\vdash_\cap M:\theta$ if and only if there is a derivation of $\Gamma\vdash M:\theta$, such that for every formula $\Gamma'\vdash N:\tau$ in that derivation, either*

- *τ is an intersection and $\Gamma'\vdash N:\tau_1\cap\cdots\cap\tau_n$ ($n\geq 2$) was derived from $\Gamma'\vdash N:\tau_1,\ldots,\Gamma'\vdash N:\tau_n$ by one application of rule ($I\cap$);*
- *or $\Gamma'\vdash N:\tau$ was derived using one of the rules (var), ($I\to$), or($E\to$).*

From now on we will only consider $\vdash$-derivations satisfying the conditions in Lemma 3. The set of β-normal terms M such that $\Gamma\vdash M:\tau$ is denoted by $\mathsf{Nhabs}(\Gamma,\tau)$. 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 β-normal inhabitants of τ is denoted by $\mathsf{Nhabs}(\tau)=\mathsf{Nhabs}(\emptyset,\tau)$. The inhabitation problem for intersection types is the problem of deciding if, for a given type $\tau\in\mathcal{T}_\cap$ (input) one has $\mathsf{Nhabs}(\tau)\neq\emptyset$, and denoted by INH.

3 Pregrammars for Intersection Types

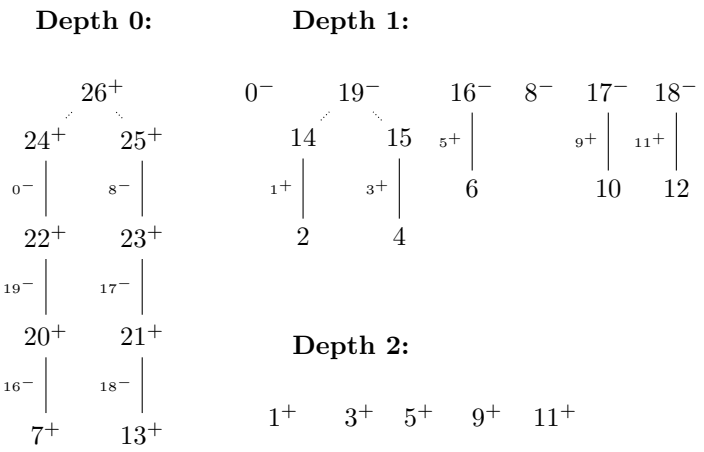
Given $\tau\in\mathcal{T}_\cap$, let $\mathsf{occT}(\tau)$ denote the set of all *occurrences* of subtypes of τ, i.e. $\mathsf{occT}(\tau)=\{\ \sigma\mid\sigma\in\tau\ \}$. Consider $\mathsf{N}(\tau)=[0..(|\mathsf{occT}(\tau)|-1)]$ as well as an (arbitrary) bijection $\mathsf{n}:\mathsf{occT}(\tau)\longrightarrow\mathsf{N}(\tau)$. We call $\mathsf{n}(\sigma)$ the identifier of $\sigma\in\tau$. The type of identifier $k\in\mathsf{N}(\tau)$ is $\mathsf{t}(k)=\mathsf{n}^{-1}(k)\in\mathsf{occT}(\tau)$. Relation $\preceq_\cap$ transfers to elements in $\mathsf{N}(\tau)$ in the obvious way, by $n\preceq_\cap m$ iff $\mathsf{t}(n)\preceq_\cap\mathsf{t}(m)$. 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 and with '−' if it corresponds to a negative subpremise. Integers that correspond to a negative occurrence, which is no subpremise, will not be superscripted. We write $m\equiv_{\mathsf{occT}}n$ if and only if the type occurrences corresponding to m and n, i.e. $\mathsf{t}(m)$ and $\mathsf{t}(n)$, are identical[1], i.e. are (not necessarily different) occurrences of the same subtype. The relation $T(\tau)\subseteq\mathsf{N}(\tau)^3$ is defined by $(n,k,m)\in T(\tau)$, abbreviated by $n\rightarrowtail^k m$, iff $\mathsf{t}(m)=\mathsf{t}(k)\to\mathsf{t}(n)$ for $m,n,k\in\mathsf{N}(\tau)$. We will display relevant information about relations T and $\preceq_\cap$ in the *dependency graph* $\mathcal{G}(\tau)$, whose set of vertices is $\mathsf{N}(\tau)$ and that consists of trees such that:

- If $\mathsf{t}(m)=\mathsf{t}(n_1)\cap\cdots\cap\mathsf{t}(n_k)$, then node m has children $n_1,\ldots,n_k$, connected by dotted edges.
- If $n\rightarrowtail^k m$, then node m has one child n and a firm edge between them labelled with k.

In light of this last observation we define $\mathsf{lab}(n,m)=k$ whenever $n\rightarrowtail^k m$, meaning that the edge between n and m is labelled by k.

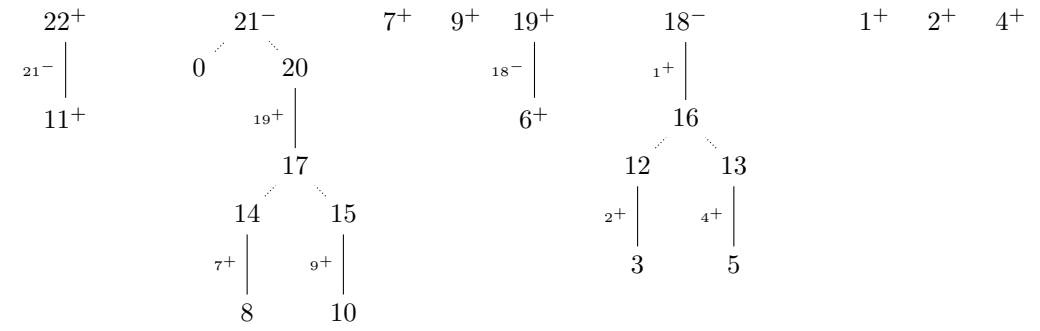
▶ **Example 4.** Consider α from Example 2, let $\mathsf{n}(\alpha)=26$, $\mathsf{n}(\alpha_1)=24$, $\mathsf{n}(\alpha_2)=25$ and the remaining identifiers of subformulas of α_1 and α_2 respectively assigned as follows: $1_0\to[((0_1\to0_2)^{14}\cap(1_3\to1_4)^{15})^{19}\to((1_5\to0_6)^{16}\to0_7)^{20}]^{22}$, and $1_8\to[(1_9\to0_{10})^{17}\to((0_{11}\to1_{12})^{18}\to0_{13})^{21}]^{23}$. The dependency graph of α is the following.

[1] Abusing on the notation, we will occasionally abbreviate this and simply write $\mathsf{t}(m)=\mathsf{t}(n)$.



Degrees of occurrences of subtypes are clearly visible in the dependency graph, since they coincide with the number of firm edges between their identifier and the top node in the subgraph of $\mathcal{G}$ in which they occur. As such, the occurrences corresponding to $22, 23, 2, 4, 6, 10, 12$ have degree 1 (their distance to the top node in the corresponding subgraph is by one firm edge), those corresponding to 20 and 21 are of degree 2 (distance by two firm edges), and those to 7 and 13 are of degree 3. All other occurrences are of degree 0, including the intersections (which correspond respectively to 26 and 19). We conclude that α is strict. On the other hand, its rank is 2 since there is an intersection at depth 1.

For β consider an attribution of identifiers to occurrences of type variable o suggested by $\beta = o_0 \cap \beta_1 \to o_{11}$, where $\beta_1 = ((o_1 \to (o_2 \to o_3) \cap (o_4 \to o_5)) \to o_6) \to (o_7 \to o_8) \cap (o_9 \to o_{10})$. Furthermore, $\mathsf{n}(\beta) = 22$, $\mathsf{n}(o_0 \cap \beta_1) = 21$, and $\mathsf{n}(\beta_1) = 20$. The remaining of this attribution n should be clear from $\mathcal{G}(\beta)$ depicted below. The graph shows clearly that β is not a strict intersection type, since both intersections $\mathsf{t}(17) = \mathsf{t}(14) \cap \mathsf{t}(15)$, as well as $\mathsf{t}(16) = \mathsf{t}(12) \cap \mathsf{t}(13)$ occur at degree 1. Since the depth of $\mathsf{t}(16)$ is 3, we have that β is of rank 4.



Positive and negative identifiers play opposite roles during the process of inhabitant search. A positive identifier m^+ represents a goal of constructing a term of type $\mathsf{t}(m)$, while a negative identifier m^- represents a (possibly available) variable of type $\mathsf{t}(m)$. As such, if one has $\mathsf{t}(m^+) = \mathsf{t}(m_1) \cap \mathsf{t}(m_2)$, then in order to construct a term of type $\mathsf{t}(m)$ one has to find an inhabitant that has simultaneously types $\mathsf{t}(m_1)$ and $\mathsf{t}(m_2)$. On the other hand, $\mathsf{t}(m^-) = \mathsf{t}(m_1) \cap \mathsf{t}(m_2)$ represents an alternative (the variable can be used either with type $\mathsf{t}(m_1)$ or $\mathsf{t}(m_2)$). Similarly, if one has $\mathsf{t}(m^+) = \mathsf{t}(k^-) \to \mathsf{t}(n^+)$, i.e. $n \rightarrowtail^k m$, then

one possibility (other than to construct an applicational term using one of the available variables) is to add a variable of type $\mathsf{t}(k^-)$ to the context and search for a term of type $\mathsf{t}(n^+)$. Conversely, an available variable of type $\mathsf{t}(m^-) = \mathsf{t}(k^+) \to \mathsf{t}(n)$, i.e. $n \rightarrowtail^k m^-$, allows to transform the goal of finding a term of type $\mathsf{t}(n)$ into a new goal of finding a term of type $\mathsf{t}(k)$. These concepts are captured in the definition of pregrammars.

▶ **Definition 5** (Pregrammar). *The pregrammar* $\mathsf{pre}(\tau)$ *of* τ *is the smallest set of rules satisfying the following conditions.*

- $m := (n_1, \ldots, n_s) \in \mathsf{pre}(\tau)$, *if* $\mathsf{t}(m^+) = \mathsf{t}(n_1) \cap \cdots \cap \mathsf{t}(n_s)$, *(*$s \geq 2$*).*
- $m := \lambda k.n \in \mathsf{pre}(\tau)$, *if* $n \rightarrowtail^k m^+$.
- $m := p_0\ n_1 \cdots n_s \in \mathsf{pre}(\tau)$, *if* $m^+ \equiv_{\mathsf{occT}} c_s$ *for* $s \geq 0$ *and there exists an ascending path from node* c_s *to node* p_0^- *in* $\mathcal{G}(\tau)$*, satisfying*

$$c_s \preceq_\cap p_s \rightarrowtail^{n_s} c_{s-1} \preceq_\cap \ldots \preceq_\cap p_1 \rightarrowtail^{n_1} c_0 \preceq_\cap p_0^-.$$

The size of a type τ, denoted by $|\tau|$, is the total number of occurrences of variables and of symbols $\to$ and $\cap$ in τ. Then, the number (of occurrences) of subformulas is $|\mathsf{N}(\tau)| \leq |\tau|$ and consequently the size of $\mathcal{G}(\tau)$, i.e. the total number of nodes and edges, is at most $2|\tau| - 1$. Let $|\tau|_\to$ and $|\tau|_\cap$ denote respectively the number of occurrences of $\to$ and of $\cap$ in τ. Then, the number of rules in $\mathsf{pre}(\tau)$ is at most $|\tau|_\cap + |\tau|_\to + |\tau| \cdot |\tau| < 3|\tau|^2$. Each of the rules contains at most $|\tau|$ identifiers. Hence, the size of $\mathsf{pre}(\tau)$, i.e. the total number of occurrences of identifiers in $\mathsf{pre}(\tau)$, is $|\mathsf{pre}(\tau)| < 3|\tau|^3$.

▶ **Example 6.** Consider α and β from Example 2. Equivalence classes in $\mathsf{N}(\alpha)/\equiv_{\mathsf{occT}}$ that are not singletons are $\{0^-, 3^+, 4, 5^+, 8^-, 9^+, 12\}$, $\{1^+, 2, 6, 7^+, 10, 11^+, 13^+\}$, and $\{16^-, 17^-\}$. The pregrammar of α has size $|\mathsf{pre}(\alpha)| = 87$ ($|\alpha| = 27$) and contains thirty-one rules:

26	:=	$(24, 25)$	20	:=	$\lambda 16.7$	21	:=	$\lambda 18.13$
24	:=	$\lambda 0.22$	25	:=	$\lambda 8.23$	$1, 7, 11, 13$	:=	19 1 | 16 5 | 17 9
22	:=	$\lambda 19.20$	23	:=	$\lambda 17.21$	$3, 5, 9$	:=	19 3 | 18 11 | 0 | 8

Moreover, the equivalence relation $\equiv_{\mathsf{occT}}$ partitions $\mathsf{N}(\beta)$ into eight classes $\{12, 13, 14, 15\}$, $\{0, 1^+, 2^+, 3, 4^+, 5, 6^+, 7^+, 8, 9^+, 10, 11^+\}$, $\{16, 17\}$, $\{18^-\}$, $\{19^+\}$, $\{20\}$, $\{21^-\}$, and $\{22^+\}$. Pregrammar $\mathsf{pre}(\beta)$ contains the following rules.

22	:=	$\lambda 21.11$	$1, 2, 4, 6, 7, 9, 11$	:=	21 | 21 19 7 | 21 19 9 | 18 1 2 | 18 1 4
19	:=	$\lambda 18.6$			

For instance, we have rule $6 :=$ 21 19 9 in $\mathsf{pre}(\beta)$, because a) $6^+ \equiv_{\mathsf{occT}} 10$, i.e. identifiers 6 and 10 represent the same type, and b) in $\mathcal{G}(\beta)$ there is an ascending path from node (component) 10 to node 21^- whose nodes satisfy $10 \preceq_\cap 10 \rightarrowtail^9 15 \preceq_\cap 17 \rightarrowtail^{19} 20 \preceq_\cap 21^-$, i.e. a variable of type $\mathsf{t}(21)$ applied successively to a term of type $\mathsf{t}(19)$ and to a term of type $\mathsf{t}(9)$ has type $\mathsf{t}(10)$ (which equals $\mathsf{t}(6)$).

4 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 tuples of integers can figure as variable names and are also referred to as *placeholders*. We refer to these objects as *extended terms*. We denote by $N[\vec{k}/x]$ the (extended) term obtained from N by replacing all free occurrences of variable x in N by placeholder $\vec{k}$.

▶ **Definition 7** (Update, Replication). *The* update of $\vec{m} = (m_1, \ldots, m_t)$ with $\vec{n} = (n_1, \ldots, n_k)$ at position i, *where* $i \in [1..t]$ *and* $t \geq 1$, *is defined by*

$$\vec{m}[\vec{n}/i] = (m_1, \ldots, m_{i-1}, n_1, \ldots, n_k, m_{i+1}, \ldots, m_t).$$

Replicating a value $k \geq 2$ times at position i in $\vec{m}$ *is denoted by* $\vec{m}[i,k]$ *and defined by*

$$\vec{m}[i,k] = (m_1, \ldots, m_{i-1}, \underbrace{m_i, \ldots, m_i}_{k}, m_{i+1}, \ldots, m_t).$$

Given an object E, possibly containing placeholders, let $E[\vec{n}/i]$ (resp. $E[i,k]$) denote the result of applying operation $[\vec{n}/i]$ (resp. $[i,k]$) to all placeholders in E. Given an extended term N, a tuple $\vec{k}$, $i \in \mathbb{N}$ and $x \in \mathcal{V}$, let $N[\vec{k}/x]$ be the result of replacing all free occurrences of x in N by placeholder $\vec{k}$, while $N[\vec{k}/i]$ is obtained by updating all placeholders in N at position i with $\vec{k}$.

▶ **Definition 8** (Type Checking Relation). *Given a type* τ, *an extended term* M, $\vec{m} = (m_1, \ldots, m_t) \in \mathbb{N}^t$, $t \geq 1$, *and* $s \geq 0$, *we write* $(M : \vec{m}) \hookrightarrow (N_1 : \vec{n_1}), \ldots, (N_s : \vec{n_s})$, *if one of the following applies.*

- *If* $m_i := \vec{n} \in \mathsf{pre}(\tau)$, *for some* $i \in [1..t]$, $\vec{n} = (n_1, \ldots, n_k)$, *and* $k \geq 2$, *then* $(M : \vec{m}) \hookrightarrow (M[i,k] : \vec{m}[\vec{n}/i])$.
- *If* $m_i := \lambda k_i.n_i \in \mathsf{pre}(\tau)$, *for all* $i \in [1..t]$, $\vec{k} = (k_1, \ldots, k_t)$, *and* $\vec{n} = (n_1, \ldots, n_t)$, *then* $(\lambda x.N : \vec{m}) \hookrightarrow (N[\vec{k}/x] : \vec{n})$.
- *If* $m_i := k_i\ n_1^i \cdots n_s^i \in \mathsf{pre}(\tau)$, *for all* $i \in [1..t]$, $\vec{n_j} = (n_j^1, \ldots, n_j^t)$, *for* $j \in [1 \ldots s]$, *and* $\vec{k} = (k_1, \ldots, k_t)$, *then* $(\vec{k}\ N_1 \cdots N_s : \vec{m}) \hookrightarrow (N_1 : \vec{n_1}), \ldots, (N_s : \vec{n_s})$.

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

▶ **Example 9.** Consider α from Example 2 and $M = \lambda xyz.y(z(yx))$. Then,

$$\begin{aligned}(M : 26) &\hookrightarrow (\lambda xyz.y(z(yx)) : (24, 25)) \hookrightarrow (\lambda yz.y(z(y(0,8))) : (22, 23)) \\ &\hookrightarrow (\lambda z.(19,17)(z((19,17)(0,8))) : (20, 21)) \hookrightarrow ((19,17)((16,18)((19,17)(0,8))) : (7, 13)) \\ &\hookrightarrow ((16,18)((19,17)(0,8)) : (1, 9)) \hookrightarrow ((19,17)(0,8) : (5, 11)) \hookrightarrow ((0,8) : (3, 9)) \hookrightarrow \epsilon.\end{aligned}$$

The proof of the following theorem, stating correctness of $\hookrightarrow$, is mainly technical and can be found in the appendix.

▶ **Theorem 10.** $\mathsf{Nhabs}(\tau) = \{\ M \mid (M : \mathsf{n}(\tau)) \hookrightarrow^* \epsilon\ \}$.

5 Inhabitation

In this section we define a rewriting relation for type inhabitation.

▶ **Definition 11** (Inhabitation Relation). *Let* $\tau \in \mathcal{T}_\cap$, $\vec{m} = (m_1, \ldots, m_t) \in \mathbb{N}^t$, $t \geq 1$, $\vec{V} \subseteq \mathbb{N}^t$, *and* $s \geq 0$. *We write*

$$(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V'} : \vec{n_1}), \ldots, (\vec{V'} : \vec{n_s}),$$

if one of the following applies.

1. *If* $m_i := \vec{n} \in \mathsf{pre}(\tau)$, *for some* $i \in [1..t]$, *where* $\vec{n} = (n_1, \ldots, n_k)$ *and* $k \geq 2$, *then*

$$(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V}[i,k] : \vec{m}[\vec{n}/i]).$$

2. *If* $m_i := \lambda k_i.n_i \in \mathsf{pre}(\tau)$, *for all* $i \in [1..t]$, *where* $\vec{k} = (k_1, \ldots, k_t)$ *and* $\vec{n} = (n_1, \ldots, n_t)$, *then*

$$(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V} \cup \{\vec{k}\} : \vec{n}).$$

3. *If* $m_i := k_i\ n_1^i \cdots n_s^i \in \mathsf{pre}(\tau)$, *for all* $i \in [1..t]$, *and* $\vec{k} = (k_1, \ldots, k_t) \in \vec{V}$, *where* $\vec{n_j} = (n_j^1, \ldots, n_j^t)$, $(1 \leq j \leq s)$, *then*

$$(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V} : \vec{n_1}), \ldots, (\vec{V} : \vec{n_s}).$$

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

▶ **Example 12.** For α from Example 2 we have the following, where $\vec{V} = \{(0,8),(19,17),(16,18)\}$.

$$\begin{aligned}(\emptyset : 26) &\rightsquigarrow (\emptyset : (24,25)) \rightsquigarrow (\{(0,8)\} : (22,23)) \rightsquigarrow (\{(0,8),(19,17)\} : (20,21)) \\ &\rightsquigarrow (\vec{V} : (7,13)) \rightsquigarrow (\vec{V} : (1,9)) \rightsquigarrow (\vec{V} : (5,11)) \rightsquigarrow (\vec{V} : (3,9)) \rightsquigarrow \epsilon,\end{aligned}$$

The following theorem, stating correctness of $\rightsquigarrow$, is proved in the appendix.

▶ **Theorem 13.** $\mathsf{Nhabs}(\tau) \neq \emptyset$ *if and only if* $(\emptyset : \mathsf{n}(\tau)) \rightsquigarrow^* \epsilon$.

5.1 Intersections at different depth or degree

We want to examine more closely how intersections at different depth or degree in a type contribute differently to the complexity of type checking and inhabitation. To that end we analyse, where in a grammar rule, their identifiers can occur and what consequences that might have in terms of applications of $\hookrightarrow$ and $\rightsquigarrow$. Let $\mathsf{r}(i)$ stand for an arbitrary identifier of a subtype at depth i. Then each rule in a pregrammar respects one of the following patterns, where $i, j \geq 0$.

(1) $\mathsf{r}(i) := (\mathsf{r}(i), \ldots, \mathsf{r}(i))$ (2) $\mathsf{r}(i) := \lambda\mathsf{r}(i+1).\mathsf{r}(i)$ (3) $\mathsf{r}(i) := \mathsf{r}(j)\ \mathsf{r}(j+1) \cdots\ \mathsf{r}(j+1)$

We observe the following:

- if m is the identifier of a positive intersection, then there is exactly one rule for m in $\mathsf{pre}(\tau)$ and that rule respects pattern (1); consequently rule 1 is incompatible with the other two rules in Definition 11;
- if m occurs at depth 0, then M cannot occur on the right side of a rule of pattern (3) in $\mathsf{pre}(\tau)$; consequently a pair $(\vec{V}, \vec{m})$, such that m is one of the coordinates of vector $\vec{m}$ (possibly $\vec{m}$ itself), occurs at most once in a rewriting sequence starting with $(\emptyset : \mathsf{n}(\tau))$; in particular intersections at depth 0 may contribute to the growth of tuples at most once;
- there can be two different rules $m := k\ n_1 \cdots n_s$ and $m' := k\ n_1' \cdots n_{s'}'$ in $\mathsf{pre}(\tau)$ such that $m = m'$ or $(m \neq m' \wedge s = s')$, only if k is the identifier of a negative subtype and there is at least one (at any degree) intersection in the tree in $\mathcal{G}(\tau)$ that is rooted in k; consequently type checking is deterministic if there are no negative intersections in τ; in terms of inhabitation negative intersections may increase the combinatorics of the problem, but don't seem to cause undecidability on their own[2];

[2] This is in accordance with the fact that there is no gap between ranks 3 and 4, which differ by negative intersections at depth 3 only. Both have undecidable inhabitation problems, shown by Urzyczyn in 1999 for rank 3 and in 2009 for rank 4 [42, 43].

- in principle, there is no relation between depths i and j in a rule of pattern (3); consequently it seems as if negative intersections at different depth contribute to the same extent to the complexity of inhabitation;
- the degree at which an intersection occurs seems to have no particular influence on the problem and similar results should be expected regardless of considering strict intersection types or not.

6 Algorithm $\mathcal{I}$

A Wajsberg/Ben-Yelles style alternating semi-decision algorithm $\mathcal{I}$ for inhabitation of intersection types, following [43], can be implemented based on relation $\rightsquigarrow$. Every rewriting sequence starting in $(\emptyset : \mathsf{n}(\tau))$ corresponds to a unique computation tree Π, whose nodes are labelled with occurrences of pairs $(\vec{V} : \vec{m})$ in the rewriting sequence. The root of Π is $(\emptyset : \mathsf{n}(\tau))$ and each node $(\vec{V} : \vec{m})$, that was rewritten by $(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V'} : \vec{n_1}), \ldots, (\vec{V'} : \vec{n_s})$, has s children, respectively labelled with $(\vec{V'} : \vec{n_1}), \ldots, (\vec{V'} : \vec{n_s})$. The rewriting sequence terminates with the empty sequence, i.e. $(\emptyset : \mathsf{n}(\tau)) \rightsquigarrow^* \epsilon$, if and only if in Π all leafs are labelled with $(\vec{V} : \vec{m})$ such that $(\vec{V} : \vec{m}) \rightsquigarrow \epsilon$. In this case the computation tree is called an *accepting computation tree for* τ.

▶ **Definition 14** (Algorithm $\mathcal{I}$). *Algorithm $\mathcal{I}$, starting with $(\emptyset : \mathsf{n}(\tau))$, aims to construct an accepting computation tree for τ, operating in each step on a pair $(\vec{V} : \vec{m})$. The procedure non-deterministically chooses a combination of rules in $\mathsf{pre}(\tau)$, such that $(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V'} : \vec{n_1}), \ldots, (\vec{V'} : \vec{n_s})$. If $s = 0$, then the algorithm accepts, otherwise it universally applies to $(\vec{V'} : \vec{n_1}), \ldots, (\vec{V'} : \vec{n_s})$. It rejects, if there is no such combination of rules in $\mathsf{pre}(\tau)$.*

Note that rules 1, 2, and 3 in Definition 11 correspond respectively to rules 1, 2, and 3 in the definition of Urzyczyn's algorithm in [43]. While rule 1 is not compatible with the other two, rules 2 and 3 may both apply at some point. If one gives priority to rule 2 whenever possible, then the algorithm is customised to find only long solutions/inhabitants in the sense of Definition 8 in [27]. We denote this variant by $\mathcal{I}^{\mathsf{long}}$.

In order to guarantee termination, i.e. convert $\mathcal{I}$ into a decision algorithm, we need to restrict the search space, for instance by limiting the maximal height of computation trees. A tree is called *non-repeating* if it has no branch containing two different nodes with the same label. Clearly, there is an accepting computation tree with root $(\emptyset : \mathsf{n}(\tau))$ iff there is a non-repeating accepting computation tree with root $(\emptyset : \mathsf{n}(\tau))$. This fact can be used to establish upper bounds for the complexity of inhabitation for different subfamilies of intersection types. Whenever there is a function $\mathsf{D} : \mathbb{N} \longrightarrow \mathbb{N}$ such that any branch in a computation tree of length $\geq \mathsf{D}(n)$ has a repetition, where $n = |\tau|$ is the size of the input, then $\mathcal{I}$ can be transformed into a decision algorithm $\mathcal{I}_{\mathsf{D}}$, which rejects when operating on nodes of depth $\geq \mathsf{D}(|\tau|)$. In that case we have an upper bound (alternating $O(\mathsf{D}(n))$-time) for deciding the inhabitation problem. This is the usual approach for obtaining upper bounds, cf. [6, 16, 27, 43].

The existence of a measure D as above depends on the maximal number of distinct nodes in a path. Such a value does not exist in general for types of rank ≥ 3 because of the possibility of having continuously growing dimension of nodes in a path. Here the *dimension* of a node labelled with $(\vec{V} : \vec{m})$ is a synonym for the arity of $\vec{m}$ (or of tuples in $(\vec{V} : \vec{m})$) and denoted by $\mathsf{dim}(\vec{V} : \vec{m}) = \mathsf{dim}(\vec{V})$. The dimension of a tree, $\mathsf{dim}(\Pi)$, is the maximal dimension of its nodes. We write $(\vec{V} : \vec{m}) \ll (\vec{V'} : \vec{n})$ if $(\vec{V'} : \vec{n})$ is a descendant of $(\vec{V} : \vec{m})$ in Π. In that case, it follows directly from Definition 11, that either $\mathsf{dim}(\vec{V} : \vec{m}) < \mathsf{dim}(\vec{V'} : \vec{n})$ or $\vec{V} \subseteq \vec{V'}$.

If $\mathsf{dim}(\vec{V} : \vec{m}) = p$ then $\vec{V} \cup \{\vec{m}\} \subseteq \mathsf{N}(\tau)^p$ and the number of tuples of arity $\leq p$ is less than $(|\mathsf{N}(\tau)| + 1)^p$. Since $|\mathsf{N}(\tau)| = |\tau|$, we conclude that the length of a non-repeating branch with a leaf of dimension $p \geq 1$ is $< (|\tau| + 1)^{2p}$. In light of the above we consider the problem of deciding, if for a given type $\tau \in \mathcal{T}_\cap$ (input) there exists an accepting computation tree of dimension $\leq p$, and denote this decision problem by INH_p.

▶ **Proposition 15.** *For each $p \geq 1$ the problem INH_p is* PSPACE*-complete.*

Proof. PSPACE-hardness is a consequence of that result for simple types, whose computation trees have dimension 1. For the upper bound one may consider decision procedure $\mathcal{I}_\mathsf{D}$, where $\mathsf{D}(n) = (n+1)^{2p}$. ◀

As foreseeable, the dimension of a computation tree relates to the notion of *dimension*, considered in [16, 18] for strict types. More precisely, to the multiset setting for which we have the following result. The proof of this result relies heavily on notions defined in [16] and is therefore included in the appendix.

▶ **Proposition 16.** *Let τ be a strict intersection type. Then, there exists M such that M can be typed with τ at bounded multiset dimension $\leq p$ if and only if $\tau \in \mathsf{INH}_p$.*

On account of this result, Proposition 15 is in fact a reformulation of Proposition 32 in [16], here extended to the entire set of intersection types.

We want to identify causes for the existing gap between the complexity of the inhabitation problem for different families of intersection types in terms of structural properties of their dependency graphs. The first observation is that the growing of dimension along a path is exclusively due to positive subtypes of the form $\tau_1 \cap \cdots \cap \tau_k$ with $k \geq 2$. Let $\mathcal{T}_\cap^-$ denote the set of intersection types without positive occurrences of intersections, i.e. all intersections occur at an odd depth. Computation trees for types in $\mathcal{T}_\cap^-$ have always dimension 1. Since this fragment falls within the scope of Proposition 15 for $p = 1$, and since $\mathcal{T} \subseteq \mathcal{T}_\cap^-$, we obtain PSPACE-completeness for $\mathcal{T}_\cap^-$, which is a generalisation of the same result for simple types [39, 41] and includes $\mathcal{T}_2^-$-types considered in [3], i.e. intersection types where all intersections occur at depth 1. More generally we have the following.

▶ **Corollary 17.** *For each $p \geq 1$ the inhabitation problem for types of the form $\tau = \cap_{i=1}^p \tau_i$, where $\tau_i \in \mathcal{T}_\cap^-$ for $i \in [1..p]$, is* PSPACE*-complete.*

Proof. PSPACE-hardness follows from the same result for $\mathcal{T} \subseteq \mathcal{T}_\cap^-$. Just consider for $\sigma \in \mathcal{T}$ the intersection type $\cap_{i=1}^p \sigma_i$, where each σ_i is a fresh copy of σ (obtained for instance by indexing all variables in σ with i). On the other hand, for $\tau = \cap_{i=1}^p \tau_i$, we have $\mathsf{n}(\tau) := (\mathsf{n}(\tau_1), \ldots, \mathsf{n}(\tau_p))$, but no other rule of that form in $\mathsf{pre}(\tau)$. So rule 1 will be applied exactly once, producing pair $(\emptyset : (\mathsf{n}(\tau_1), \ldots, \mathsf{n}(\tau_p))$. Hence, any accepting computation tree is of dimension p and it suffices to apply decision procedure $\mathcal{I}_\mathsf{D}$, where $\mathsf{D}(n) = n^{2p}$. ◀

Urzyczyn's proof for rank 2 inhabitation in [43] also shows EXPSPACE-completeness for finite intersections of $\mathcal{T}_\cap^-$-types. We denote the rank independent set of these types by

$$\bigcap \mathcal{T}_\cap^- = \{\, \cap_{i=1}^p \tau_i \mid p \geq 1 \ \wedge\ \forall i \in [1..p]\ \tau_i \in \mathcal{T}_\cap^- \,\}.$$

This set properly contains the set of strict rank 2 types.

▶ **Proposition 18.** *The inhabitation problem for $\bigcap \mathcal{T}_\cap^-$ is* EXPSPACE*-complete.*

Proof (from Urzyczyn [43]). The set of strict rank 2 types is a proper subset of $\bigcap\mathcal{T}_\cap^-$ and EXPSPACE-hardness for rank 2 was shown by reduction from the halting problem for bus machines to strict rank 2 types. For the upper bound one may consider decision procedure $\mathcal{I}_{\mathsf{D}}$, where $\mathsf{D}(n) = n + n^n \cdot n^n$. This stems from the fact that for a $\bigcap\mathcal{T}_\cap^-$-type τ algorithm $\mathcal{I}$ starting with $(\emptyset : \mathsf{n}(\tau))$ will initially apply rule 1 at most n times, expanding the dimension of nodes up to at most n, where $n = |\tau|$. After that phase rule 1 will no longer apply. Given two nodes of dimension n such that $(\vec{V} : \vec{m}) \ll (\vec{V}' : \vec{m}')$ we have necessarily $\vec{V} \subseteq \vec{V}'$. Thus, there are at most n^n different sets of dimension n in a path of a computation tree, as well as at most n^n different tuples $\vec{m}$ of dimension n. Finally, $n + n^n \cdot n^n = n + 2^{2n \log n} \leq n + 2^{n^2}$, for $n \geq 4$. ◀

In [16, Proposition 24] a not rank-bounded family of types, ranged over by T and U, where

$$T ::= a \mid U \to T \qquad \text{and} \qquad U ::= A \mid (\cap_{i=1}^n T_i) \to U$$

was considered. More precisely it was shown that every normal inhabitant of an intersection of the form $\cap_{i=1}^n U_i$ can be typed at multiset dimension n. Note that the set of types ranged over by U is a proper subset of $\mathcal{T}_\cap^-$. On the other hand, the set of types of the form $\cap_{i=1}^n U_i$ properly includes the family of types used to show EXPSPACE-hardness for rank 2 [43]. Consequently, the proof of Proposition 18 still works and we have EXPSPACE-hardness also for this proper subclass. We apply this line of reasoning to another rank independent set of types, which is a superset of $\bigcap\mathcal{T}_\cap^-$ and contains types with positive intersections at depth ≥ 2.

▶ **Definition 19.** *For $m, n \in \mathsf{N}(\tau)$ we define $m \succ n$ iff one of the following holds:*

- *$m := (n_1, \ldots, n_s) \in \mathsf{pre}(\tau)$ and $n = n_i$ for some $i \in [1..s]$;*
- *$m := \lambda k.n \in \mathsf{pre}(\tau)$;*
- *$m := k\ n_1 \cdots n_s \in \mathsf{pre}(\tau)$ and $n = n_i$ for some $i \in [1..s]$.*

Then, $\succ^+$ denotes the transitive closure of $\succ$. A type $\tau \in \mathcal{T}_\cap$ is called growth restrained *if there is no identifier m with $m := (n_1, \ldots, n_s) \in \mathsf{pre}(\tau)$ and such that $m \succ^+ m$.*

▶ **Proposition 20.** *Inhabitation for growth restrained types is* EXPSPACE*-complete.*

Proof. If τ has rank 2, then $m := (n_1, \ldots, n_s) \in \mathsf{pre}(\tau)$ implies that $\mathsf{t}(m)$ occurs at depth 0 in τ and consequently m appears nowhere else in $\mathsf{pre}(\tau)$. Hence, τ is growth restrained and EXPSPACE-hardness follows from that result for rank 2. If τ is growth restrained then any rule $m := (n_1, \ldots, n_s) \in \mathsf{pre}(\tau)$ applies at most once in a path of a computation tree for τ, whose dimension can for that reason not exceed n, where $n = |\tau|$. Again, there are at most $(n+1)^n$ tuples of dimension $\leq n$. We conclude that the length of a non-repeating branch with a leaf of dimension $\leq n$ is $< (n+1)^{2n}$ and consider decision procedure $\mathcal{I}_{\mathsf{D}}$, where $\mathsf{D}(n) = (n+1)^{2n}$. But $(n+1)^{2n} = 2^{2n \log(n+1)} < 2^{n^2}$, for $n \geq 6$. ◀

The set of growth restrained types contains properly the set of types in which positive occurrences of intersections are only allowed at depth 0. This set, on the other hand contains properly the set of rank 2 types, as well as $\bigcap\mathcal{T}_\cap^-$. Since all the aforementioned type classes contain the set of strict rank 2 types, for which Urzyczyn showed EXPSPACE-hardness, one obtains as a corollary EXPSPACE-completeness for all these classes.

Wajsberg/Ben-Yelles style search algorithms are the established vehicle to implement counting algorithms [6, 23, 24, 8]. We follow that direction and use algorithm $\mathcal{I}$ to show that the problem of counting for growth restrained types is EXPSPACE-complete. Counting

means to decide, if for a given type τ, the set of normal inhabitants $\mathsf{Nhabs}(\tau)$ is empty, finite or infinite. Following the usual approach we provide limits $\mathsf{d}(|\tau|)$ and $\mathsf{D}(|\tau|)$, such that: (i) $|\mathsf{Nhabs}(\tau)| \neq 0$ iff there is an accepting computation tree of height lower than $\mathsf{D}(|\tau|)$; (ii) $|\mathsf{Nhabs}(\tau)| = \infty$ iff there is an accepting computation tree of height between $\mathsf{d}(|\tau|)$ and $\mathsf{D}(|\tau|)$. Condition (i) is decided by $\mathcal{I}_{\mathsf{D}(|\tau|)}$. For (ii) we use a customised version of $\mathcal{I}_{\mathsf{D}(|\tau|)}$, that in each step remembers the highest depth the algorithm operated on so far and accepts the whole computation only if that value is $\geq \mathsf{d}(|\tau|)$.

▶ **Proposition 21.** *Counting for growth restrained types is* EXPSPACE*-complete.*

Proof. EXPSPACE-hardness follows from Proposition 20. To show that the problem can be solved in exponential space we consider $\mathsf{d}(n) = (n+1)^n$ and $\mathsf{D}(n) = (n+1)^{3n}$. Let τ be a growth restrained type with $|\tau| = n$. The proof of Proposition 20 shows that $\mathsf{Nhabs}(\tau) \neq \emptyset$ iff there is an accepting computation tree for τ of height $\leq (n+1)^{2n} < \mathsf{D}(n)$. It remains to show that $|\mathsf{Nhabs}(\tau)| = \infty$ iff there is an accepting computation tree of height between $\mathsf{d}(n)$ and $\mathsf{D}(n)$. Consider an accepting computation tree Π for τ, which we know to have dimension $\leq n$. Given two nodes $\iota \ll \iota'$, labelled respectively with $(\vec{V} : \vec{m})$ and $(\vec{V}' : \vec{n})$, there is a possibly empty sequence $\zeta_{\iota,\iota'}$ of replications that have been applied on the path from ι to ι' (corresponding to applications of rule 1, Definition 11). Let $\vec{V}\zeta_{\iota,\iota'}$ denote the result of applying the replications in $\zeta_{\iota,\iota'}$ successively to $\vec{V}$. Then, $\vec{V}\zeta_{\iota,\iota'} \subseteq \vec{V}'$. Now, suppose that Π has a branch of length $\geq \mathsf{d}(n) = (n+1)^n$, which is the maximal number of tuples of dimension $\leq n$. Thus, there must be two distinct nodes (of equal dimension) such that $(\vec{V}_1 : \vec{m}) \ll (\vec{V}_2 : \vec{m})$ and $\vec{V}_1 \subseteq \vec{V}_2$. Let Π_1 be the subtree of Π rooted in node ι labelled with $(\vec{V}_1 : \vec{m})$. For each other node ι' in Π_1 consider the corresponding sequence $\zeta_{\iota,\iota'}$ of replications from ι to ι'. We denote by $\Pi_1\zeta$ the tree obtained from Π_1 by replacing the label of each node $\iota' = (\vec{V}' : \vec{n})$ by $(\vec{V}' \cup \vec{V}_2\,\zeta_{\iota,\iota'} : \vec{n})$. An accepting tree of bigger height can now be obtained replacing in Π the subtree rooted in $(\vec{V}_2 : \vec{m})$ by $\Pi_1\zeta$. $|\mathsf{Nhabs}(\tau)| = \infty$ follows by repetition. This part of the proof corresponds to the Stretching Lemma in [23]. It remains to show that the existence of a tree Π of height $\geq \mathsf{D}(n) = (n+1)^{3n}$ implies that there is a tree of height in $[\mathsf{d}(n), \mathsf{D}(n)[$. This last part corresponds to the Shrinking Lemma [23] and is achieved by successively shortening subtrees in Π by a controlled amount, such that the final result is an accepting tree of height $< \mathsf{D}(n)$, but still $\geq \mathsf{d}(n)$. Consider a branch in Π of length $> \mathsf{D}(n) > \mathsf{d}(n)$. There must be two distinct nodes (a repetition) such that $\iota = (\vec{V} : \vec{m}) \ll (\vec{V} : \vec{m}) = \iota'$ and such that the path from ι to ι' has length $l \leq (n+1)^{2n}$. Now, consider the tree Π' obtained by replacing in Π the subtree rooted in ι by the subtree rooted in ι'. All paths starting in ι are now shortened by length $l \geq (n+1)^{2n}$. However, $(n+1)^{3n} - l \geq (n+1)^n$ and the result is a tree of height $\geq \mathsf{d}(n)$. It remains to repeat this process as long as there are paths of length $> \mathsf{D}(n)$. ◀

Changing the limits in the previous proof to $\mathsf{d}(n) = (n+1)^p$ and $\mathsf{D}(n) = (n+1)^{3p}$ suffices to show a similar result for counting in bounded dimensions.

▶ **Corollary 22.** *For each $p \geq 1$, counting the number of accepting computation trees of dimension $\leq p$ is* PSPACE*-complete.*

Some studies consider fragments where the amount of positive and/or negative occurrences of subtypes is constrained. In [4] it was proved that in the simple type system all inhabitants of a given *negatively non-duplicated* type τ are $\beta\eta$-equivalent. Here, negatively non-duplicated means that every type variable occurs at most once negatively in τ. Bourreaux and Salvati adapted this correspondence to the case of λ-terms containing occurrences of constants in [7]. The following result shows that the same is true for the intersection type system.

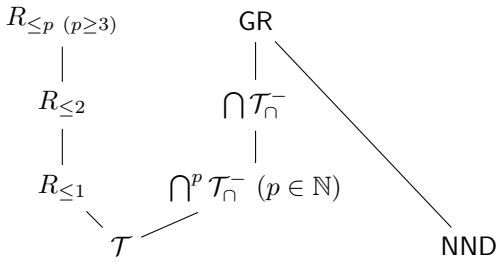


Figure 3 Intersection Type Families.

▶ **Proposition 23.** *Deciding inhabitation for negatively non-duplicated intersection types is in* PSPACE. *Furthermore, if* $M, N \in \mathsf{Nhabs}(\tau)$*, then* $M =_{\beta\eta} N$.

Proof. We know that τ has a normal inhabitant if and only if it has a long one. Suppose that Π is an accepting non-repeating computation tree for τ obtained by algorithm $\mathcal{I}^{\mathsf{long}}$. We show that Π has height $\leq |\tau|^2$, i.e. Π can be obtained by $\mathcal{I}^{\mathsf{long}}_{\mathsf{D}}$, where $\mathsf{D}(n) = n^2$. If τ is negatively non-duplicating, then for every $m_i^+ \in \mathsf{N}(\tau)$ there is at most one rule of the form $m_i := k_i\ n_1^i \cdots n_s^i \in \mathsf{pre}(\tau)$. Given a pair $(\vec{V} : \vec{m})$, such that neither rule 1 nor rule 2 in Definition 11 apply, there is at most one $\vec{k} = (k_1, \ldots, k_t) \in \vec{V}$ such that $m_i := k_i\ n_1^i \cdots n_s^i \in \mathsf{pre}(\tau)$, for all $i \in [1..|\vec{m}|]$. We conclude that any run of $\mathcal{I}^{\mathsf{long}}$ on $(\emptyset, \mathsf{n}(\tau))$ is deterministic (up to the order in which identifiers in a tuple, to which rule 1 applies, are chosen) and that there is at most one accepting computation tree (up to that order), implying that the number of long inhabitants is finite. Consider any branch $(\vec{V}_1 : \vec{m}_1), \ldots, (\vec{V}_s : \vec{m}_s)$ in Π, where $(\vec{V}_1 : \vec{m}_1) = (\emptyset, \mathsf{n}(\tau))$. We associate a tree π to this branch whose root is labelled by $\mathsf{n}(\tau)$ and such that at depth $i-1$ nodes are labelled by $m_1^i, \ldots, m_{t_i}^i$, where $\vec{m}_i = (m_1^i, \ldots, m_{t_i}^i)$ and have children defined by the following. If $(\vec{V}_{i+1} : \vec{m}_{i+1})$ was obtained by rule 1, because $m_j^i := (n_1, \ldots, n_k) \in \mathsf{pre}(\tau)$, for some $j \in [1..t_i]$, $\vec{n} =$ and $k \geq 2$, then m_j^i has k children labelled respectively with $n_1, \ldots, n_k$. Every other node m_l^i at depth $i-1$ has one child labelled with m_l^i. If $(\vec{V}_{i+1} : \vec{m}_{i+1})$ was obtained by rule 2 or by rule 3, then each node has one child labelled respectively with $k_1, \ldots, k_{t_i}$, for $\vec{k} = (k_1, \ldots, k_{t_i})$ as in Definition 11 (rules 2 and 3). Consider two nodes at different depth in a branch of π, which are labelled with the same identifier m. Since Π is accepting and the rewriting process deterministic (up to order of application of rule 1) all nodes between them are also labelled by m, corresponding to successive applications of rule 1. Otherwise, the algorithm would have entered an infinite loop. But rule 1 can be applied successively at most $|\tau|$ times. On the other hand, there are at most $|\tau|$ different identifiers in $\mathsf{N}(\tau)$. We conclude that $s \leq |\tau|^2$. On the other hand, this means that whenever $(\vec{V}_i : \vec{m}_i) \rightsquigarrow (\vec{V}_i \cup \{\vec{k}\} : \vec{m}_{i+1})$ by rule 2, then $\vec{k} \notin \vec{V}_i$. Consequently, an accepting tree Π corresponds to exactly one long normal inhabitant M and any other normal inhabitant can be obtained from M by η-expansion, cf. proof of Lemma 9 in [27]. ◀

7 Conclusions

We presented a pregrammar based framework for addressing inhabitation related problems in the intersection type system. In that setting types are first represented by dependency graphs, which expose their underlying structure and thereby reveal properties related to (the complexity of) intersection type inhabitation. Rewriting relations corresponding to type checking and inhabitation for normal forms were given. These operate solely on sets of tuples in terms of simple operations, update and replication. After proving the correctness of those

relations, which involves some bureaucracy, the method became available to be implemented in a Wajsberg/Ben-Yelles style search algorithm and any forthcoming reasoning could be expressed in terms of rewriting of pairs (consisting of a set of tuples and a tuple). That was illustrated by revisiting and partially extending some well-known problems. An overview of results is given in Table 1. The relation between the different sets of type families, regarding inclusion, is displayed in an Hasse diagram in Figure 3. There $\mathcal{T}$ denotes the set of simple types, $R_{\leq n}$ the set of types of rank at most n, $\bigcap^p \mathcal{T}_\cap^-$ the set of types of the form $\cap_{i=1}^p \tau_i$ for some fixed p, GR the set of groth restrained bypes, and finally NND the set of negatively non-duplicated intersection types. The set of types inhabited in (some) bounded multiset dimension is orthogonal to the remaining families, c.f. [16], and therefore not included in the diagram.

Studying further applications of the method is left for future work. For instance, the amount of complexity caused by negative intersections should be placed under more careful observation. As such, one could wonder about strictly positive fragments.

Table 1 Complexity Results for Inhabitation and Counting.

Problem	Complexity
Inhabitation of Simple Types	PSPACE-complete [39, 41]
Counting of Simple Types	PSPACE-complete [24]
Rank 1 Inhabitation	PSPACE-complete [43]
Rank 2 Inhabitation	EXPSPACE-complete [27, 43]
Rank ≥ 3 Inhabitation	undecidable [43, 42]
Inhabitation in Bounded Multiset Dimension	EXPSPACE-complete [16]
Inhabitation in Fixed Bounded Multiset Dimension	PSPACE-complete [16]
INH_p for fixed p	PSPACE-complete[3] [Proposition 15]
Counting in Fixed Bounded Multiset Dimension	PSPACE-complete [Corollary 22]
Inhabitation of $\tau = \cap_{i=1}^p \tau_i$ ($\tau_i \in \mathcal{T}_\cap^-$, fixed p)	PSPACE-complete [Corollary 17]
$\bigcap \mathcal{T}_\cap^-$ Inhabitation	EXPSPACE-complete [43, Proposition 18]
Inhabitation for growth restrained types	EXPSPACE-complete [Proposition 20]
Counting for growth restrained types	EXPSPACE-complete [Proposition 21]
Inhabitation for negatively non-duplicated intersection types	PSPACE [Proposition 23]

References

1 S. Alves and S. Broda. Inhabitation machines: determinism and principality. In *Ninth Workshop on Non-Classical Models of Automata and Applications, NCMA 2017*, pages 57–70, 2017.

2 S. Alves and S. Broda. A unifying framework for type inhabitation. In *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018)*, Leibniz International Proceedings in Informatics (LIPIcs), 2018.

[3] The problem is EXPSPACE-complete if p isn't fixed beforehand, corresponding then to inhabitation in bounded multiset dimension.

3 Sandra Alves and Sabine Broda. Pre-grammars and inhabitation for a subset of rank 2 intersection types. *Electr. Notes Theor. Comput. Sci.*, 344:25–45, 2019.

4 T. Aoto. Uniqueness of normal proofs in implicational intuitionistic logic. *Journal of Logic, Language and Information*, 8(2):217–242, 1999.

5 Hendrik Pieter Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Perspectives in logic. Cambridge University Press, 2013.

6 Ch. Ben-Yelles. *Type Assignment in the Lambda-Calculus: Syntax and Semantics*. PhD thesis, University College of Swansea, September 1979.

7 P. Bourreau and S. Salvati. A datalog recognizer for almost affine -cfgs. In *MOL*, pages 21–38, 2011.

8 Sabine Broda and Luís Damas. Counting a type's principal inhabitants. In *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings*, pages 69–82, 1999.

9 Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. Inhabitation for non-idempotent intersection types. *Logical Methods in Computer Science*, 14(3), 2018.

10 Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Logic Journal of the IGPL*, 25(4):431–464, 2017.

11 Martin W. Bunder. The inhabitation problem for intersection types. In *Theory of Computing 2008. Proc. Fourteenth Computing: The Australasian Theory Symposium (CATS 2008), Wollongong, NSW, Australia, January 22-25, 2008. Proceedings*, pages 7–14, 2008.

12 Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for λ-terms. *Arch. Math. Log.*, 19(1):139–156, 1978.

13 Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980.

14 Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Functional characters of solvable terms. *Math. Log. Q.*, 27(2-6):45–58, 1981.

15 Boris Düdder, Moritz Martens, and Jakob Rehof. Staged composition synthesis. In *Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, pages 67–86, 2014.

16 Andrej Dudenhefner and Jakob Rehof. Intersection type calculi of bounded dimension. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 653–665, 2017.

17 Andrej Dudenhefner and Jakob Rehof. Typability in bounded dimension. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017.

18 Andrej Dudenhefner and Jakob Rehof. Principality and approximation under dimensional bound. *PACMPL*, 3(POPL):8:1–8:29, 2019.

19 Joshua Dunfield. Elaborating intersection and union types. *J. Funct. Program.*, 24(2-3):133–165, 2014.

20 Joshua Dunfield and Frank Pfenning. Type assignment for intersections and unions in call-by-value languages. In *Foundations of Software Science and Computational Structures, 6th International Conference, FOSSACS 2003 Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings*, pages 250–266, 2003.

21 Jonathan Frankle, Peter-Michael Osera, David Walker, and Steve Zdancewic. Example-directed synthesis: a type-theoretic interpretation. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 802–815, 2016.

22 Timothy S. Freeman and Frank Pfenning. Refinement types for ML. In *Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation (PLDI), Toronto, Ontario, Canada, June 26-28, 1991*, pages 268–277, 1991.

23 J.R. Hindley. *Basic Simple Type Theory*, volume 42 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1997.

24 S. Hirokawa. Infiniteness of proof (α) is polynomial-space complete. *Theoretical Computer Science*, 206(1-2):331–339, 1998.

25 Jean-Louis Krivine. *Lambda-calculus, types and models*. Ellis Horwood series in computers and their applications. Masson, 1993.

26 Toshihiko Kurata and Masako Takahashi. Decidable properties of intersection type systems. In *Typed Lambda Calculi and Applications, Second International Conference on Typed Lambda Calculi and Applications, TLCA '95, Edinburgh, UK, April 10-12, 1995, Proceedings*, pages 297–311, 1995.

27 D. Kusmierek. The inhabitation problem for rank two intersection types. In *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, pages 240–254, 2007.

28 D. Leivant. Polymorphic type inference. In *Proceedings of Principles of Programming Languages (POPL'83)*, pages 88–98, New York, NY, USA, 1983. ACM Press.

29 Ralph Loader. The undecidability of lambda-definability.

30 Christian Mossin. Exact flow analysis. *Mathematical Structures in Computer Science*, 13(1):125–156, 2003.

31 Jens Palsberg and Christina Pavlopoulou. From polyvariant flow information to intersection and union types. *J. Funct. Program.*, 11(3):263–317, 2001.

32 G. Plotkin. Lambda definability and logical relations, Technical Report Memorandum SAI-RM-4, School of Artificial Intelligence, University of Edingburgh, (1973).

33 G. Pottinger. A type assignment for the strongly normalizable lambda-terms. In J. Hindley and J. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 561–577. Academic Press, 1980.

34 Steven J. Ramsay. Exact intersection type abstractions for safety checking of recursion schemes. In *Proceedings of the 16th International Symposium on Principles and Practice of Declarative Programming, Kent, Canterbury, United Kingdom, September 8-10, 2014*, pages 175–186, 2014.

35 Jakob Rehof and Pawel Urzyczyn. The complexity of inhabitation with explicit intersection. In *Logic and Program Semantics - Essays Dedicated to Dexter Kozen on the Occasion of His 60th Birthday*, pages 256–270, 2012.

36 John C. Reynolds. *Design of the Programming Language FORSYTHE*, page 173–233. Birkhauser Boston Inc., USA, 1997.

37 Sylvain Salvati. Recognizability in the simply typed lambda-calculus. In *Logic, Language, Information and Computation, 16th International Workshop, WoLLIC 2009, Tokyo, Japan, June 21-24, 2009. Proceedings*, pages 48–60, 2009.

38 Sylvain Salvati, Giulio Manzonetto, Mai Gehrke, and Henk Barendregt. Loader and urzyczyn are logically related. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, pages 364–376, 2012.

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

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

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

42 P. Urzyczyn. The emptiness problem for intersection types. *Journal of Symbolic Logic*, 64(3):1195–1215, 1999.

43 P. Urzyczyn. Inhabitation of low-rank intersection types. In *TLCA'09*, volume 5608 of *LNCS*, pages 356–370. Springer, 2009.

44 Steffen van Bakel. Strict intersection types for the lambda calculus. *ACM Comput. Surv.*, 43(3):20:1–20:49, 2011.

A Appendix

Proof of Lemma 3

Lemma 3. We have $\Gamma \models M : \theta$ if and only if there is a derivation of $\Gamma \vdash M : \theta$, such that for every formula $\Gamma' \vdash N : \tau$ in that derivation, either

- τ is an intersection and $\Gamma' \vdash N : \tau_1 \cap \cdots \cap \tau_n$ $(n \geq 2)$ was derived from $\Gamma' \vdash N : \tau_1, \ldots, \Gamma' \vdash N : \tau_n$ by one application of rule $(I\cap)$;
- or $\Gamma' \vdash N : \tau$ was derived using one of the rules (var), (I$\to$), or(E$\to$).

Proof. For the *if*-part of the proof note that the resulting formula of an application of rule (var) (resp. (E$\to$)) in a $\vdash$-derivation can be obtained by one derivation step with rule (var) (resp. (E$\to$)), followed by zero or one application of rule (E$\cap$) in a $\models$-derivation. On the other hand, rules (I$\to$) and (I$\cap$) are identical in both systems.

For the *only if*-part of the proof we proceed by induction on the length of the $\models$-derivation. Suppose that $\Gamma' \models N : \tau_1 \cap \cdots \cap \tau_n$ was obtained by rule (var) because $(x : \tau_1 \cap \cdots \cap \tau_n) \in \Gamma'$. If $n = 1$, then $\Gamma' \vdash N : \tau_1$ can also be obtained by rule (var). Otherwise, $\Gamma' \vdash N : \tau_1 \cap \cdots \cap \tau_n$ can be obtained by n applications of rule (var) followed by one application of (I$\cap$). Since rules (I$\to$) and (I$\cap$) are identical in both systems, it remains to consider a formula of the form $\Gamma' \models xN_1 \cdots N_s N_{s+1} : \rho_1 \cap \cdots \cap \rho_n$ obtained from $\Gamma' \models xN_1 \cdots N_s : \sigma \to \rho_1 \cap \cdots \cap \rho_n$ and $\Gamma' \models N_{s+1} : \sigma$ by rule (E$\to$). The corresponding formula $\Gamma' \vdash xN_1 \cdots N_s N_{s+1} : \rho_1 \cap \cdots \cap \rho_n$ can be obtained by considering n copies of the derivations for $\Gamma' \vdash xN_1 \cdots N_s : \sigma \to \rho_1 \cap \cdots \cap \rho_n$ and for $\Gamma' \vdash N_{s+1} : \sigma$. To each pair rule (E$\to$) is applied, leading to derivations of $\Gamma' \vdash xN_1 \cdots N_s N_{s+1} : \rho_1, \ldots, \Gamma' \vdash xN_1 \cdots N_s N_{s+1} : \rho_n$. Finally, $\Gamma' \vdash xN_1 \cdots N_s N_{s+1} : \rho_1 \cap \cdots \cap \rho_n$ follows in one step using (I$\cap$) for $n > 1$. ◀

In order to prove correctness of our method we first need to establish the precise correspondence that exists between occurrences of subtypes and of variables and subterms in inhabitants.

Variables, Subterms and Occurrences of Subtypes

Consider a term M, a type τ, and a derivation of $\vdash M : \tau$. In the following we assign occurrences of subtypes of τ to variables and terms in formulas $\Gamma \vdash N : \sigma$ in that derivation. This assignment will be used to establish the correctness of the pregrammars, that are defined in the next section. Every $x \in \mathsf{dom}(\Gamma)$ is assigned a negative subpremise $\mathsf{nsp}(x)$ of τ. Term N is assigned a positive subtype $\mathsf{pst}(N)$ of τ. Additionally, if $\Gamma \vdash N : \sigma$ was derived by rule (var) or (E$\to$), then N is also assigned a negative subtype $\mathsf{nst}(N)$ of τ. The assignment is such that $\mathsf{pst}(N)$ and $\mathsf{nst}(N)$ are occurrences of σ and for $x \in \mathsf{dom}(\Gamma)$ we have $\mathsf{nsp}(x) = \Gamma(x)$.

▶ **Definition 24** ($\mathsf{pst}, \mathsf{nst}, \mathsf{nsp}$). *Consider a term M, a type $\tau \in \mathcal{T}_\cap$, and a derivation of $\vdash M : \tau$ with derivation tree Π. The assignment of* nsp, nst *and* pst *to occurrences of variables and terms in the formulas, that appear in Π, is bottom-up, starting with $\vdash M : \tau$.*

- *For the root $\vdash M : \tau$ of Π, let $\mathsf{pst}(M) = \tau$*[3]*.*
- *Consider $\Gamma \vdash x : \tau_i$ obtained by (var), where $\tau_1 \cap \cdots \cap \tau_n = \Gamma(x)$, $n \geq 1$ and $i \in [1..n]$, with $\mathsf{nsp}(x) = \tau_1 \cap \cdots \cap \tau_n$ and $\mathsf{pst}(x) = \tau_i$. We assign $\mathsf{nst}(x)$ to x on the right side of $\vdash$, choosing the negative occurrence of τ_i in $\mathsf{nsp}(x)$.*

[3] M is necessarily of the form $\lambda x.N$, thus $\mathsf{nst}(M)$ is not defined.

- *Next, consider a formula* $\Gamma \vdash xN_1 \cdots N_k : \tau_i$ *derived by* $(I{\to})$*, with* $k > 0$*. Let* Π' *be the subtree of* Π *that derives that formula,* $\mathsf{pst}(xN_1 \cdots N_k) = \tau_i$ *and suppose that* $\mathsf{nsp}(y)$ *is defined for all* $y \in \mathsf{dom}(\Gamma)$*. Consider the declaration* $x : \rho_0 \in \Gamma$*. Then, there are subtypes* $\sigma_1, \ldots, \sigma_k, \rho_1, \ldots, \rho_k$*, such that* $\sigma_j \to \rho_j \preceq_\cap \rho_{j-1}$ *for* $j \in [1..k]$ *and* $\tau_i \preceq_\cap \rho_k$*. Furthermore, formula* $\Gamma \vdash x : \sigma_1 \to \rho_1$ *is first combined with a derived formula* $\Gamma \vdash N_1 : \sigma_1$ *(by some subtree* Π_1 *of* Π'*). The resulting formula* $\Gamma \vdash xN_1 : \sigma_2 \to \rho_2$ *is then combined with a derived formula* $\Gamma \vdash N_2 : \sigma_2$ *(by some subtree* Π_2 *of* Π'*), etc. Contexts of formulas in* Π' *always contain* Γ *and the negative subpremises to variables in* Γ *in these formulas will be those assigned to variables in the root of* Π'*, which is* $\Gamma \vdash xN_1 \cdots N_k : \tau_i$*. We now successively assign, operating top down, negative subtypes to terms* $x, xN_1, \ldots, xN_1 \cdots N_k$*, as well as positive subtypes to terms* $N_1, \ldots, N_k$*, in the formulas in this part of the tree. For* x *on the right side of* $\Gamma \vdash x : \sigma_1 \to \rho_1$ *let* $\mathsf{nst}(x)$ *be the occurrence of* $\sigma_1 \to \rho_1$ *in* $\mathsf{nsp}(x) = \rho_0$*. Now, suppose that formula* $\Gamma \vdash xN_1 \cdots N_{j-1} : \sigma_j \to \rho_j$ *is combined with* $\Gamma \vdash N_j : \sigma_j$*, deriving* $\Gamma \vdash xN_1 \cdots N_j : \rho'_j$*, where* $\rho'_j \preceq_\cap \rho_j$ *and* $j \in [1..k]$*. Consider* $\mathsf{nst}(xN_1 \cdots N_{j-1}) = \sigma_j \to \rho_j$ *(which is already assigned). Then, let* $\mathsf{pst}(N_j)$ *be the occurrence of* σ_j *in* $\mathsf{nst}(xN_1 \cdots N_{j-1})$ *and let* $\mathsf{nst}(xN_1 \cdots N_j)$ *be the negative occurrence of the component* ρ'_j *in* $\sigma_j \to \rho_j$ *(note that* $\rho'_j \preceq_\cap \rho_j$*).*
- *Consider* $\Gamma \vdash \lambda x.N : \sigma \to \rho$ *derived from* $\Gamma \cup \{x : \sigma\} \vdash N : \rho$ *by* $(I{\to})$ *and* $\mathsf{pst}(\lambda x.N) = \sigma \to \rho$*. Then, for* $\Gamma \cup \{x : \sigma\} \vdash N : \rho$ *let* $\mathsf{pst}(N)$ *be the occurrence of* ρ *in* $\mathsf{pst}(\lambda x.N)$*. If* $x \in \mathsf{dom}(\Gamma)$*, then* $\mathsf{nsp}(x)$ *is already defined. Otherwise, let* $\mathsf{nsp}(x)$ *be the negative occurrence of* σ *in* $\mathsf{pst}(\lambda x.N)$*.*
- *Now, consider* $\Gamma \vdash N : \tau_1 \cap \cdots \cap \tau_n$ *derived from* $\Gamma \vdash N : \tau_1, \ldots, \Gamma \vdash N : \tau_n$*,* $(n > 1)$ *by rule* $(I\cap)$*. Then,* $\mathsf{nsp}(x)$ *is already defined for all* $x \in \mathsf{dom}(\Gamma)$*, as well as* $\mathsf{pst}(N) = \tau_1 \cap \cdots \cap \tau_n$*. To the occurrence of* N *in* $\Gamma \vdash N : \tau_i$ *we assign the positive occurrence of* τ_i *in* $\mathsf{pst}(N)$*.*

Besides of the operations on tuples of updating and replication we also need to define an operation of contraction.

▶ **Definition 25** (Contraction)**.** Contracting $k \geq 2$ positions starting at position i in $\vec{m} = (m_1, \ldots, m_t)$ *is denoted by* $\vec{m}[i, k\downarrow]$ *for* $i + k \leq t$ *and defined by*

$$\vec{m}[i, k\downarrow] = (m_1, \ldots, m_i, m_{i+k}, \ldots, m_t).$$

Given an object E, possibly containing placeholders, let $E[i, k\downarrow]$ denote the result of applying operation $[i, k\downarrow]$ to all placeholders in E. Then, $E[i, k][i, k\downarrow] = E$.

Proof of Theorem 10

Theorem 10 shows correctness of $\hookrightarrow$, where $\vec{k}|_i$ denotes the projection of the ith coordinate of a tuple $\vec{k}$.

Theorem 10. $\mathsf{Nhabs}(\tau) = \{\, M \mid (M : \mathsf{n}(\tau)) \hookrightarrow^* \epsilon \,\}$.

Proof.
($\subseteq$) Consider a derivation tree of $\vdash M : \tau$ and an occurrence of a formula Ψ of the form $\Gamma \vdash N : \rho$ in derivation tree Π, where $\Gamma = \{x_1 : \sigma_1, \ldots, x_s : \sigma_s\}$. Let $n_i^- = \mathsf{n}(\mathsf{nsp}(x_i)) \in \mathsf{N}(\tau)$ for $i \in [1..s]$, and $m^+ = \mathsf{n}(\mathsf{pst}(N))$, according to Definition 24. Then, $\mathsf{t}(n_i) = \sigma_i$ for $i \in [1..s]$ and $\mathsf{t}(m) = \rho$. We show by induction that $(N[\Gamma] : m) \hookrightarrow^* \epsilon$, using $N[\Gamma]$ as an abbreviation for $N[n_1/x_1, \cdots, n_s/x_s]$.

Suppose that Ψ was obtained by rule (var), i.e. $N = x_i$, $\sigma_i = \sigma_i^1 \cap \cdots \cap \sigma_i^l$ and $\rho = \sigma_i^j$ for some $j \in [1..l]$. For $n_i = \mathsf{n}(\sigma_i)$ and $n_i^j = \mathsf{n}(\sigma_i^j)$, we have $m^+ \equiv_{\mathsf{occT}} n_i^j$ and $n_i^j \preceq_\cap n_i$, and consequently $m := n_i \in \mathsf{pre}(\tau)$. Thus, $(N[\Gamma] : m) = (n_i : m) \hookrightarrow \epsilon$.

Next, consider a formula Ψ of the form $\Gamma \vdash x_i N_1 \cdots N_k : \rho$ derived by (I$\to$), with $k > 0$. The derivation of Ψ results from successively applying rule (I$\to$) to $\Gamma \vdash x_i : \delta_1 \to \rho_1$ and $\Gamma \vdash N_1 : \delta_1$, to $\Gamma \vdash x N_1 : \delta_2 \to \rho_2$ and $\Gamma \vdash N_2 : \delta_2$, ..., to $\Gamma \vdash x N_1 \cdots N_{k-1} : \delta_k \to \rho_k$ and $\Gamma \vdash N_k : \delta_k$. Then, $\delta_1 \to \rho_1 \preceq_\cap \sigma_i$, $\delta_j \to \rho_j \preceq_\cap \rho_{j-1}$ for $j \in [2..k]$, and $\rho \preceq_\cap \rho_k$. Consider the identifiers of the subtypes assigned to terms in these formulas according to Definition 24. Let $c_j = \mathsf{n}(\mathsf{nst}(x_i N_1 \dots N_j))$ for $j \in [0..k]$, and $q_j = \mathsf{n}(\mathsf{pst}(N_j))$ for $j \in [1..k]$. Furthermore, let p_j be the identifier of the occurrence of ρ_j in $\mathsf{t}(c_{j-1}) = \delta_j \to \rho_j$ for $j \in [1..k]$. Then, $m^+ \equiv_{\mathsf{occ}\Gamma} c_k \preceq_\cap p_k \rightarrowtail^{q_k} c_{k-1} \preceq_\cap \dots \preceq_\cap p_2 \rightarrowtail^{q_2} c_1 \preceq_\cap p_1 \rightarrowtail^{q_1} c_0 \preceq_\cap n_i$. We conclude that $m := n_i\ q_1 \cdots q_k \in \mathsf{pre}(\tau)$. Thus, $(N[\Gamma] : m) = (n_i(N_1[\Gamma]) \cdots (N_k[\Gamma]) : m) \hookrightarrow (N_1[\Gamma] : q_1), \dots, (N_k[\Gamma] : q_k)$
$\hookrightarrow^* \epsilon$.

Let Ψ be $\Gamma \vdash \lambda x.N' : \sigma \to \delta$ obtained by rule (I$\to$) from $\Gamma' \vdash N' : \delta$, where $\Gamma' = \Gamma \cup \{x : \sigma\}$. For Ψ we have $m^+ = \mathsf{n}(\mathsf{pst}(\lambda x.N'))$. Consider for $\Gamma' \vdash N' : \delta$ the identifiers $k^- = \mathsf{n}(\mathsf{nsp}(x))$ and $n^+ = \mathsf{n}(\mathsf{pst}(N))$. Then, $n \rightarrowtail^k m$, and consequently $m := \lambda k.n \in \mathsf{pre}(\tau)$. Thus, $((\lambda x.N')[\Gamma] : m) \hookrightarrow (N'[\Gamma][k/x] : n) = (N'[\Gamma'], n) \hookrightarrow^* \epsilon$.

Finally, consider $\Gamma \vdash N : \tau_1 \cap \dots \cap \tau_k$ derived from $\Gamma \vdash N : \tau_1, \dots, \Gamma \vdash N : \tau_k$ by rule by (I$\cap$), for some $k > 1$. Let $m^+ = \mathsf{n}(\mathsf{pst}(N))$ and $n_i = \mathsf{n}(\mathsf{pst}(N))$ in formulas $\Gamma \vdash N : \tau_i$, for $i \in [1..k]$. Then, $m := (n_1, \dots, n_k) \in \mathsf{pre}(\tau)$. Thus, $(N[\Gamma] : m) \hookrightarrow (N[\Gamma][1,k] : (n_1, \dots, n_k))$. By induction $(N[\Gamma] : n_i) \hookrightarrow^* \epsilon$, for $i \in [1..k]$. These rewriting sequences are, but for applications of rule 1 which creates replications in corresponding places, determined by the structure of $N[\Gamma]$ and can be combined to a rewriting sequence from $(N[\Gamma][1,k] : (n_1, \dots, n_k))$ to ϵ.

($\supseteq$) For the other inclusion consider a term M, such that $(M : \mathsf{n}(\tau)) \hookrightarrow^* \epsilon$. Let $(E : \vec{m})$ be any pair appearing in the corresponding rewriting sequence, where E is an extended term and $\vec{m} = (m_1, \dots, m_k)$. Furthermore, consider $\vec{P} = \{\vec{p_1}, \dots, \vec{p_l}\}$, the set of placeholders that occur in E, and let us interpret each tuple in $\vec{P}$ as the name of a term variable. We will show, by induction on the length of $(E, \vec{m}) \hookrightarrow^* \epsilon$, that for all $i \in [1..k]$ we have $\Gamma_i \vdash E : \mathsf{t}(\vec{m}|_i)$, where the projection $|_i$ is defined by $(s_1, \dots, s_k)|_i = s_i$ and $\Gamma_i = \{\vec{p_1} : \mathsf{t}(\vec{p_1}|_i), \dots, \vec{p_l} : \mathsf{t}(\vec{p_l}|_i)\}$. In particular, it follows that $\vdash M : \tau$.

Suppose that $(E : \vec{m}) \hookrightarrow (E[j,t] : \vec{m'})$, with $\vec{m'} =$
$\vec{m}[(s_1, \dots, s_t)/j] = (m_1, \dots, m_{j-1}, s_1, \dots, s_t, m_{j+1}, \dots, m_k)$, because $m_j := (s_1, \dots, s_t) \in \mathsf{pre}(\tau)$, for some $j \in [1..k]$. For this last pair the set of placeholders is now $\vec{P'} =$
$= \{\vec{p_1}[j,t], \dots, \vec{p_l}[j,t]\}$. If $j \neq i$, let i' be the new position of m_i in $\vec{m'}$, which is i if $i < j$, and equal to $i + t - 1$ if $i > j$. Then, $\Gamma_{i'} = \{\vec{p_1}[j,t] : \mathsf{t}(\vec{p_1}[j,t]|_{i'}), \dots, \vec{p_l}[j,t] : \mathsf{t}(\vec{p_l}[j,t]|_{i'})\} = \{\vec{p_1}[j,t] : \mathsf{t}(\vec{p_1}|_i), \dots, \vec{p_l}[j,t] : \mathsf{t}(\vec{p_l}|_i)\}$. By the induction hypothesis, $\Gamma_{i'} \vdash E[j,t] : \mathsf{t}(\vec{m'}|_{i'})$. This, means that $\Gamma_i \vdash E : \mathsf{t}(\vec{m}|_i)$, because $\vec{m}|_i = \vec{m'}|_{i'}$ and the only change in both formulas is the name of variables (from $\vec{p_h}$ to $\vec{p_h}[j,t]$). If $i = j$, then we have $\Gamma_i \vdash E : \mathsf{t}(\vec{m}|_i)$ if $\Gamma'_i \vdash E[i,t] : \mathsf{t}(s_r)$ for all $r \in [1..t]$, where $\Gamma'_i = \{\vec{p_1}[i,t] : \mathsf{t}(\vec{p_1}|_i), \dots, \vec{p_l}[i,t] : \mathsf{t}(\vec{p_l}|_i)\}$. It follows from the induction hypothesis (note that the position of s_r in $\vec{p}[i,t]$ is $i + r - 1$) that for $\Gamma^r_i = \{\vec{p_1}[i,t] : \mathsf{t}(\vec{p_1}[i,t]|_{i+r-1}), \dots, \vec{p_l}[i,t] : \mathsf{t}(\vec{p_l}[i,t]|_{i+r-1})\}$ we have $\Gamma^r_i \vdash E[i,t] : \mathsf{t}(s_r)$ for all $r \in [1..t]$. But, $\Gamma'_i = \Gamma^r_i$ and the result holds.

Now, suppose that $(\lambda x.E : \vec{m}) \hookrightarrow (N[\vec{v}/x] : \vec{n})$ because $m_j := \lambda v_j.n_j \in \mathsf{pre}(\tau)$, for all $j \in [1..k]$, where $\vec{v} = (v_1, \dots, v_k)$ and $\vec{n} = (n_1, \dots, n_k)$. In particular, $m_i := \lambda v_i.n_i \in \mathsf{pre}(\tau)$ and $\mathsf{t}(m_i) = \mathsf{t}(v_i) \to \mathsf{t}(n_i)$. By the induction hypothesis $\Gamma_i \cup \{\vec{v} : \mathsf{t}(v_i)\} \vdash E[\vec{v}/x] : \mathsf{t}(n_i)$. Thus, $\Gamma_i \vdash \lambda \vec{v}.E[\vec{v}/x] : \mathsf{t}(\vec{m}|_i)$, but $\lambda \vec{v}.E[\vec{v}/x] \equiv_\alpha \lambda x.E$.

Finally, consider $(\vec{v}\ E_1 \cdots E_s : \vec{m}) \hookrightarrow (E_1 : \vec{n_1}), \dots, (E_s : \vec{n_s})$ with $s \geq 0$, because $\vec{v} = (v_1, \dots, v_k)$, and $m_j := v_j\ n^j_1 \cdots n^j_s \in \mathsf{pre}(\tau)$, for all $j \in [1..k]$. In particular, we have $\vec{m}|_i = m_i := v_i\ n^i_1 \cdots n^i_s \in \mathsf{pre}(\tau)$. It follows from Definition 5 that there is a sequence of identifiers such that

$$m^+_i \equiv_{\mathsf{occ}\Gamma} c_s \preceq_\cap q_s \rightarrowtail c_{s-1} \preceq_\cap \dots \preceq_\cap q_1 \rightarrowtail c_0 \preceq_\cap v^-_i\,,$$

and $\mathsf{lab}(q_t, c_{t-1}) = n_t^i$ for $t \in [1..s]$. Then, $\mathsf{t}(m_i) = \mathsf{t}(c_s) \preceq_\cap \mathsf{t}(q_s)$, $\mathsf{t}(c_{s-1}) = (\mathsf{t}(n_s^i) \to \mathsf{t}(q_s)), \ldots, \mathsf{t}(c_0) = (\mathsf{t}(n_1^i) \to \mathsf{t}(q_1)) \preceq_\cap \mathsf{t}(v_i)$. We have $\vec{v}\!:\!\mathsf{t}(v_i) \in \Gamma_i$. Thus, $\Gamma_i \vdash \vec{v}\!:\!\mathsf{t}(n_1^i) \to \mathsf{t}(q_1)$ by (var). By the induction hypothesis, $\Gamma_{E_1} \vdash E_1 : \mathsf{t}(n_1^i)$, where Γ_{E_1} is the restriction of Γ_i to the free variables (placeholders) in E_1. Thus we also have $\Gamma_i \vdash E_1 : \mathsf{t}(n_1^i)$ and by rule (E$\to$) and $(\mathsf{t}(n_2^i) \to \mathsf{t}(q_2)) \preceq_\cap \mathsf{t}(q_1)$ it follows that $\Gamma_i \vdash \vec{v}\ E_1 : \mathsf{t}(n_2^i) \to \mathsf{t}(q_2)$, etc. Repeating this process we conclude that $\Gamma_i \vdash \vec{v}\ E_1 \cdots E_s : \mathsf{t}(c_s) = \mathsf{t}(m_i) = \mathsf{t}(\vec{m}|_i)$. ◀

Proof of Theorem 13

▶ **Definition 26.** *For a particular rewriting sequence of* $(\emptyset : \mathsf{n}(\tau)) \rightsquigarrow^* \epsilon$*, where in each step the combination of rewriting rules applied is given, we define a function* pair *that computes for each* $(\vec{V} : \vec{m})$ *in that rewriting sequence a tuple* $(\vec{\Gamma}, M) = \mathsf{pair}(\vec{V} : \vec{m})$*, where* M *is an extended term and* $\vec{\Gamma}$ *a set of placeholders with the arity of* $\vec{m}$*. The function* pair *is recursively defined as follows.*

1. *Let* $\vec{n} = (n_1, \ldots, n_k)$ *and suppose that* $(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V}[i,k] : \vec{m}[\vec{n}/i])$ *because* $m_i := \vec{n} \in \mathsf{pre}(\tau)$*, for some* $i \in [1..t]$ *and* $k \geq 2$*. If* $\mathsf{pair}((\vec{V}[i,k] : \vec{m}[\vec{n}/i]) = (\vec{\Gamma}, M)$*, then* $\mathsf{pair}(\vec{V} : \vec{m}) = (\vec{\Gamma}[i, k\downarrow], M[i, k\downarrow])$.
2. *If* $(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V} \cup \{\vec{k}\} : \vec{n})$ *because* $m_i := \lambda k_i.n_i \in \mathsf{pre}(\tau)$*, for all* $i \in [1..t]$*,* $\vec{k} = (k_1, \ldots, k_t)$ *and* $\vec{n} = (n_1, \ldots, n_t)$*, then* $\mathsf{pair}(\vec{V} : \vec{m}) = (\vec{\Gamma} \setminus \{\vec{k}\}, \lambda\vec{k}.N)$*, where* $(\vec{\Gamma}, N) = \mathsf{pair}(\vec{V} \cup \{\vec{k}\} : \vec{n})$.
3. *If* $(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V} : \vec{n_1}), \ldots, (\vec{V} : \vec{n_s})$*, because* $m_i := k_i\ n_1^i \cdots n_s^i \in \mathsf{pre}(\tau)$*, for all* $i \in [1..t]$*,* $\vec{k} = (k_1, \ldots, k_t) \in \vec{V}$*, and* $\vec{n_j} = (n_j^1, \ldots, n_j^t)$*, then* $\mathsf{pair}(\vec{V} : \vec{m}) = (\{\vec{k}\} \cup \vec{\Gamma_1} \cup \cdots \cup \vec{\Gamma_s}\ , \vec{k}\ N_1 \cdots N_s)$*, where* $(\vec{\Gamma_j}, N_j) = \mathsf{pair}(\vec{V} : \vec{n_j})$*, for* $1 \leq j \leq s$ *(*$s \geq 0$*).*

The correctness of function pair is stated in the following lemma.

▶ **Lemma 27.** *Suppose that* $(\vec{V} : \vec{m}) \rightsquigarrow^* \epsilon$ *for* $\vec{m} = (m_1, \ldots, m_t)$*,* $(t \geq 1)$*, and for some particular rewriting sequence* $(\vec{\Gamma}, M) = \mathsf{pair}(\vec{V} : \vec{m})$*. Let* $\vec{\Gamma}|_i$ *denote the context* $\{\ \vec{p}|_i : \mathsf{t}(\vec{p}|_i) \mid \vec{p} \in \vec{\Gamma}\ \}$*. Furthermore, let* $M|_i$ *be the result obtained by replacing every placeholder* $\vec{v}$ *in* M *by* $\vec{v}|_i$*. Then,* $\vec{\Gamma}|_j \vdash M|_j : \mathsf{t}(\vec{m}|_j)$*, for all* $j \in [1..t]$.

Proof. By induction on the length of the rewriting sequence. Let $\vec{n} = (n_1, \ldots, n_k)$ and suppose that $(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V}[i,k] : \vec{m}[\vec{n}/i])$ because $m_i := \vec{n} \in \mathsf{pre}(\tau)$, for some $i \in [1..t]$ and $k \geq 2$. Furthermore, let $\mathsf{pair}(\vec{V} : \vec{m}) = (\vec{\Gamma}[i, k\downarrow], M[i, k\downarrow])$, for $\mathsf{pair}(\vec{V}[i,k] : \vec{m}[\vec{n}/i]) = (\vec{\Gamma}, M)$. First, consider $j \neq i$ $(j \in [1..t])$ and let j' be the position of m_j in $\vec{m}[\vec{n}/i] = (m_1, \ldots, m_{i-1}, n_1, \ldots, n_k, m_{i+1}, \ldots, m_t)$. By induction, we have $\vec{\Gamma}|_{j'} \vdash M|_{j'} : \mathsf{t}(\vec{m}[\vec{n}/i]|_{j'})$. But $(\vec{\Gamma}|_{j'}, M|_{j'}) = (\vec{\Gamma}[i, k\downarrow]|_j, M[i, k\downarrow]|_j)$ and $\mathsf{t}(\vec{m}[\vec{n}/i]|_{j'}) = \mathsf{t}(\vec{m}|_j)$. For $j = i$ note that $\mathsf{t}(m_i) = \mathsf{t}(n_1) \cap \cdots \cap \mathsf{t}(n_k)$. It follows from the induction hypothesis that $\vec{\Gamma}|_{i+r-1} \vdash M|_{i+r-1} : \mathsf{t}(\vec{m}[\vec{n}/i]|_{i+r-1})$ for all $r \in [1..k]$. But, $(\vec{\Gamma}|_{i+r-1}, M|_{i+r-1}) = (\vec{\Gamma}[i, k\downarrow]|_i, M[i, k\downarrow]|_i)$ and $\mathsf{t}(\vec{m}[\vec{n}/i]|_{i+r-1}) = \mathsf{t}(n_r)$. Thus, by rule ($\cap$I) we have $\vec{\Gamma}[i, k\downarrow]|_i \vdash M[i, k\downarrow] : \mathsf{t}(m_i)$.

Now suppose that $(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V} \cup \{\vec{k}\} : \vec{n})$, where $\vec{k} = (k_1, \ldots, k_t)$ and $\vec{n} = (n_1, \ldots, n_t)$ and such that $m_i := \lambda k_i.n_i \in \mathsf{pre}(\tau)$, for all $i \in [1..t]$. For $j \in [1..t]$ we have $\mathsf{t}(\vec{m}|_j) = \mathsf{t}(\vec{k}|_j) \to \mathsf{t}(\vec{n}|_j)$. Let $\mathsf{pair}(\vec{V} \cup \{\vec{k}\} : \vec{n}) = (\vec{\Gamma}, N)$ and consider $\mathsf{pair}(\vec{V} : \vec{m}) = (\vec{\Gamma} \setminus \{\vec{k}\}, \lambda\vec{k}.N)$. By the induction hypothesis $\vec{\Gamma}|_j \vdash N|_j : \mathsf{t}(\vec{n}|_j)$. Since $\vec{\Gamma}|_j \cup \{\vec{k}|_j : \mathsf{t}(\vec{k}|_j)\}$ is consistent by definition, it follows that $\vec{\Gamma}|_j \setminus \{\vec{k}|_j : \mathsf{t}(\vec{k}|_j)\} \vdash \lambda\vec{k}|_j.(N|_j) : \mathsf{t}(\vec{k}|_j) \to \mathsf{t}(\vec{n}|_j)$. But $\vec{\Gamma}|_j \setminus \{\vec{k}|_j : \mathsf{t}(\vec{k}|_j)\} = (\vec{\Gamma} \setminus \{\vec{k}\})|_j$ and $\lambda\vec{k}|_j.(N|_j) = (\lambda\vec{k}.N)|_j$.

Finally suppose that $(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V} : \vec{n_1}), \ldots, (\vec{V} : \vec{n_s})$, because $m_i := k_i\ n_1^i \cdots n_s^i \in \mathsf{pre}(\tau)$, for all $i \in [1..t]$, $\vec{k} = (k_1, \ldots, k_t) \in \vec{V}$, and $\vec{n_j} = (n_j^1, \ldots, n_j^t)$, (Â§1 $\leq j \leq s$). Let $(\vec{\Gamma_h}, N_h) = \mathsf{pair}(\vec{V} : \vec{n_h})$, for $1 \leq h \leq s$ $(s \geq 0)$. Consider $\mathsf{pair}(\vec{V} : \vec{m}) = (\vec{\Gamma}, \vec{k}\ N_1 \cdots N_s)$, where $\vec{\Gamma} =$

$\{\vec{k}\}\cup\vec{\Gamma_1}\cup\cdots\cup\vec{\Gamma_s}$. By definition $\vec{\Gamma}|_j$ is consistent. It follows from the induction hypothesis that $\vec{\Gamma_h}|_j \vdash N_h|_j : \mathsf{t}(\vec{n_h}|_j)$. On the other hand, $\vec{\Gamma}|_j \vdash \vec{k}|_j : \mathsf{t}(\vec{k}|_j)$. Now, it remains to apply the same argument as in the proof of Theorem 10 to conclude that $\vec{\Gamma}|_j \vdash (\vec{k}\ N_1 \cdots N_s)|_j : \mathsf{t}(\vec{m}|_j)$. ◀

Theorem 13. $\mathsf{Nhabs}(\tau) \neq \emptyset$ if and only if $(\emptyset : \mathsf{n}(\tau)) \rightsquigarrow^* \epsilon$.

Proof. The 'if' part follows from Lemma 27. For the 'only if' part consider a term P such that $\vdash P : \tau$. By Theorem 10 there exists a $\hookrightarrow$-rewriting sequence, such that $(P : \mathsf{n}(\tau)) \hookrightarrow^* \epsilon$. We show, by induction on its length, that for each pair $(M : \vec{m})$ in that sequence we have $(\mathsf{FV}(M) : \vec{m}) \rightsquigarrow^* \epsilon$, where M is an extended term with placeholders figuring as names for term variables. In this part of the proof we also use the fact that $(\vec{V}, \vec{p}) \rightsquigarrow^* \epsilon$ implies $(\vec{V'}, \vec{p}) \rightsquigarrow^* \epsilon$, whenever $\vec{V} \subseteq \vec{V'}$. If $(M : \vec{m}) \hookrightarrow (M[i,k] : \vec{m}[\vec{n}/i])$, then $(\mathsf{FV}(M) : \vec{m}) \rightsquigarrow (\mathsf{FV}(M)[i,k] : \vec{m}[\vec{n}/i]) = (\mathsf{FV}(M[i,k]) : \vec{m}[\vec{n}/i])$ and the result follows from the induction hypothesis. If $(\lambda x.N : \vec{m}) \hookrightarrow (N[\vec{k}/x] : \vec{n})$, then $(\mathsf{FV}(\lambda x.N) : \vec{m}) \rightsquigarrow (\mathsf{FV}(\lambda x.N) \cup \{\vec{k}\} : \vec{n})$. But $\mathsf{FV}(N[\vec{k}/x]) \subseteq \mathsf{FV}(\lambda x.N) \cup \{\vec{k}\}$ and the result follows from the induction hypothesis. Finally, if $(\vec{k}\ N_1 \cdots N_s : \vec{m}) \hookrightarrow (N_1 : \vec{n_1}), \ldots, (N_s : \vec{n_s})$, then $(\mathsf{FV}(\vec{k}\ N_1 \cdots N_s) : \vec{m})$
$\rightsquigarrow (\mathsf{FV}(\vec{k}\ N_1 \cdots N_s) : \vec{n_1}), \ldots, (\mathsf{FV}(\vec{k}\ N_1 \cdots N_s) : \vec{n_s})$. Again $\mathsf{FV}(N_j) \subseteq \mathsf{FV}(\vec{k}\ N_1 \cdots N_s)$ and the result holds by induction. ◀

Proof of Proposition 16

Proposition 16. Let τ be a strict intersection type. Then, there exists M such that M can be typed with τ at bounded multiset dimension $\leq p$ (denoted by $\Vdash_p M : \tau$) if and only if $\tau \in \mathsf{INH}_p$.

Proof. We consider the decision procedure $\mathcal{J}$ given in [16], Section 6.1. The procedure transforms multisets of constraints of the form $\mathcal{C} = \langle \Gamma_1 \vdash ? : \gamma_1, \ldots, \Gamma_n \vdash ? : \gamma_n \rangle$, where all Γ_i's have the same domain and each γ_i is strict and not an intersection. Such a multiset is referred to as a configuration of the decision procedure. Consider a configuration $\mathcal{C}$ as above with $\mathsf{dom}(\Gamma_i) = \{x_1, \ldots, x_k\}$. Regarding some particular order in the multiset (for instance $1, \ldots, n$) we associate the pair $\mathcal{C}^\nabla = (\vec{V} : \vec{m})$, where $\vec{m} = (\mathsf{n}(\gamma_1), \ldots, \mathsf{n}(\gamma_n))$ and $\vec{V} = \{\ (\mathsf{n}(\Gamma_1(x_i)), \ldots, \mathsf{n}(\Gamma_n(x_i)))\ \mid\ x_i \in [1..k]\ \}$. In the other direction, let $(\vec{V} : \vec{m})^\triangle = (\{\vec{v}_1, \ldots, \vec{v}_k\} : (m_1, \ldots, m_n))^\triangle = \langle \Gamma_1 \vdash ? : \mathsf{t}(m_1), \ldots, \Gamma_n \vdash ? : \mathsf{t}(m_n) \rangle$, where $\Gamma_i(x_j) = \mathsf{t}(\vec{v}_j|_i)$, for $i \in [1..n]$ and $j \in [1..k]$. We have that $\mathsf{dim}(\mathcal{C}^\nabla)$ equals the number of constraints in multiset $\mathcal{C}$, and the same is true for a pair $(\vec{V}, \vec{m})$ and configuration $(\vec{V}, \vec{m})^\triangle$.

Transformation step 1 of procedure $\mathcal{J}$ corresponds to an application of rule 2 by algorithm $\mathcal{I}$. Since in strict types intersections are not allowed on the right side of $\to$, rule 1 never applies right after rule 2. On the other hand, transformation step 2 in $\mathcal{J}$ corresponds to an application of rule 3, followed subsequently by all possible applications of rule 1 (which is incompatible with the remaining two rules) by algorithm $\mathcal{I}$.

Now, using these conversions from configurations[4] to pairs and back, it remains to show by induction on the height of trees that for every accepting computation tree $T^{\mathcal{J}}_{\langle \vdash ? : \tau_1, \ldots, \vdash ? : \tau_n \rangle}$ determined by a run of $\mathcal{J}$ (cf. [16]) there is an accepting computation tree Π for $\tau = \tau_1 \cap \cdots \cap \tau_n$ by $\mathcal{I}$, and vice-versa. The height of Π is typically $O(|\tau|) \cdot h$, where h is the height of $T^{\mathcal{J}}_{\langle \vdash ? : \tau \rangle}$, due to the fact that each step 2 of $\mathcal{J}$ corresponds to an application of rule 3 followed by all possible ($\leq |\tau|$) applications of rule 1 by $\mathcal{I}$. ◀

[4] For conversion $\cdot^\nabla$ one has to consider in each step a convenient order in $\mathcal{C}$.