The Complexity of Inhabitation with Explicit Intersection

Jakob Rehof[1] and Paweł Urzyczyn*[2]

[1] Technical University of Dortmund
Department of Computer Science
jakob.rehof@cs.tu-dortmund.de
[2] University of Warsaw
Institute of Informatics
urzy@mimuw.edu.pl

Abstract. It is shown that the inhabitation problem for intersection types without the intersection introduction rule is EXPSPACE-complete and that the further restriction without subtyping is PSPACE-complete.

1 Introduction

We consider the complexity of the inhabitation problem (is there a pure λ-term that can be assigned a given type?) for the λ-calculus with intersection types [4] under the restriction that the intersection introduction rule ($\cap$I) is eliminated. We refer to the restricted system as $\lambda(-\cap\mathrm{I})$. Whereas the inhabitation problem for the full intersection type system is undecidable [16], it was shown by Kurata and Takahashi [7] that the inhabitation problem for $\lambda(-\cap\mathrm{I})$ is decidable. However, the question of complexity of the inhabitation problem for $\lambda(-\cap\mathrm{I})$ was left open. We prove here that the problem is EXPSPACE-complete. In addition, we show that the further restriction without subtyping is PSPACE-complete.

The system $\lambda(-\cap\mathrm{I})$ is a natural restriction of the full intersection type system [4], because it is equivalent to the *explicitly* typed version ("Church style fragment") of that system [7, Lemma 3.1]), in which all abstractions are of the form $\lambda x:\tau.M$. One can also say that in system $\lambda(-\cap\mathrm{I})$ intersections are explicit because, in the absence of intersection introduction, they cannot be introduced by the typing rules.

The full intersection type system arises from the Curry-style simply-typed λ-calculus [2] by addition of the rules

$$\frac{\Gamma\vdash M:\tau_1 \quad \Gamma\vdash M:\tau_2}{\Gamma\vdash M:\tau_1\cap\tau_2}(\cap\mathrm{I}) \quad \frac{\Gamma\vdash M:\tau_1\cap\tau_2}{\Gamma\vdash M:\tau_i}(\cap\mathrm{E}) \quad \frac{\Gamma\vdash M:\tau \quad \tau\le\tau'}{\Gamma\vdash M:\tau'}(\le)$$

where the subtyping rule ($\le$) refers to a certain quasi-order on types. Usually, the intersection elimination rule ($\cap$E) is contained in the subtyping theory, by the axiom $\tau_1\cap\tau_2\le\tau_i$, and consequently rule ($\cap$E) is left out in systems with

* Partly supported by MNiSW grant N N206 355836.

subtyping thus presented (we follow this convention in the present paper). However, in the standard core system without subtyping, the rule ($\cap$E) is present. In the present paper, we consider the system $\lambda(-\cap\mathrm{I})$, where ($\cap$I) is eliminated (and ($\cap$E) is contained in rule ($\leq$)) as well as the system $\lambda(-\cap\mathrm{I})$ without subtyping, in which ($\cap$E) is present and both of the rules ($\cap$I) and ($\leq$) are absent.

Intersection type systems belong to a class of propositional logics with enormous expressive power. Intersection types capture several semantic properties of λ-terms, and their type reconstruction problem has long been known to be undecidable, since they characterize exactly the set of strongly normalizing terms [11]. The inhabitation problem for the full system is undecidable [16] and is closely related to the λ-definability problem [13].

Our result adds to a recently growing body of knowledge concerning the fine structure of the intersection type inhabitation (provability) problem obtained by considering various restrictions of the system. Recall that the borderline between decidability and undecidability was clarified in [8, 17] by rank restrictions (as defined in [9]), with inhabitation in rank 2 types being shown Expspace-complete and undecidable from rank 3 and up. The system without intersection elimination ($\cap$E) was shown to be decidable in [18] (the upper bound is non-elementary and the exact complexity is open). In finite combinatory logic with intersection types [12] (combinatory logic with intersection types [5] restricted to monomorphic types) inhabitation is Exptime-complete with or without subtyping. Finite combinatory logic with intersection types can be presented as the restriction of system $\lambda(-\cap\mathrm{I})$, in which the function type introduction rule ($\to$ I) is eliminated. Seen in this perspective, the present result allows us to compare the relative complexity of the explicitly typed rule ($\to$ I) in terms of inhabitation complexity (Exptime vs. Expspace). Following [12], inhabitation can be seen as a foundation for type-based function composition synthesis. Under this perspective, our result determines the complexity of synthesis in the "Church-style" fragment of the intersection typed λ-calculus. Finally, our present result shows that, in contrast to some other restrictions (e.g., finite combinatory logic), the presence of subtyping makes a big difference (Expspace vs. Pspace). Intuitively, in system $\lambda(-\cap\mathrm{I})$, subtyping can express a certain part of the logic contained in intersection introduction. For example, the judgment

$$f : (p \to q) \cap (r \to s), \quad x : p \cap r \quad \vdash \quad fx : q \cap s$$

is derivable in the full system without subtyping (because $fx : q$ and $fx : s$ can be derived separately), and also in system $\lambda(-\cap\mathrm{I})$ with subtyping (because $(p \to q) \cap (r \to s) \leq p \cap r \to q \cap s$), but not in system $\lambda(-\cap\mathrm{I})$ without subtyping.[3]

The situation is summarized in the following overview, where we list the known results for the various restrictions mentioned above. In addition to the rank restrictions we consider the cases where one or more of the rules ($\cap$I), ($\cap$E), ($\leq$), ($\to$I) are eliminated (indicated by a "$-$"). In the cases where rule ($\leq$) is

[3] However, subtyping and ($\cap$I) are in general incomparable: for instance the judgment $\vdash \mathbf{I} : (p \to p) \cap (q \to q)$ requires ($\cap$I), while $x : p \to q \cap r \vdash x : p \to q$ requires ($\leq$).

not eliminated, the presence or absence of rule ($\cap$E) is immaterial.

Rank $k = 2$	:	EXPSPACE-complete
Rank $k > 2$	:	Undecidable
$-(\cap\text{I})$	:	EXPSPACE-complete
$-(\cap\text{I}, \leq)$	:	PSPACE-complete
$-(\cap\text{E}, \leq)$	:	Decidable
$-(\to\text{I})$	:	EXPTIME-complete
$-(\to\text{I}, \leq)$	:	EXPTIME-complete

Let us note here that *alternation* [3] plays a fundamental role in problems of type inhabitation. Intuitively, we may need to consider subgoals of the form $\Gamma \vdash ? : \tau$ where the inhabitant (indicated by "?") is an application of the form $(xY_1 \dots Y_n)$. This is equivalent to asking whether there *exists* a variable x in Γ with a type of the form $\tau_1 \to \cdots \to \tau_n \to \tau$ such that *all* of the types τ_1 through τ_n are inhabited. Hence, one can cast the PSPACE-algorithm for inhabitation in simple types [14] as an alternating polynomial time procedure (see also [15]). In intersection type systems, inhabitation goals may involve questions of the form $\Gamma \vdash ? : \tau_1 \cap \tau_2$ leading to the parallel (universal) questions of inhabitation in τ_1 *and* τ_2 by *some* single term X. In the present paper, our exponential space upper bound is achieved by an alternating exponential time procedure, and our lower bound relies on the computational model of *bus machines*, introduced in [17], which generically simulates alternating exponential time Turing machines.

Comparing the EXPSPACE-completeness of inhabitation with rank 2 types [17] to our present result, it is remarkable that the same model (bus machines) is used as a basis of reduction in the hardness proofs in both cases. However, the reduction must be adjusted, as a result of fundamental differences in the systems under consideration. In the former case, types are restricted (rank 2) and the typing rules are unrestricted, whereas here we consider unrestricted types with restricted rules (see Section 3).

2 Preliminaries

Types: Type expressions, ranged over by τ, σ etc., are defined by

$$\tau \ ::= \ a \mid \tau \to \tau \mid \tau \cap \tau$$

where $a, b, c, \dots$ range over *atoms* comprising of type constants, including the constant ω, and type variables. As usual, types are taken modulo commutativity ($\tau \cap \sigma = \sigma \cap \tau$), associativity ($(\tau \cap \sigma) \cap \rho = \tau \cap (\sigma \cap \rho)$), and idempotency ($\tau \cap \tau = \tau$). A type *environment* Γ is a finite set of type assumptions of the form $x : \tau$, and we let $dm(\Gamma)$ and $rn(\Gamma)$ denote the domain and range of Γ.

A type $\tau \cap \sigma$ is said to have τ and σ as *components*. For an intersection of several components we sometimes write $\bigcap_{i=1}^{n} \tau_i$ or $\bigcap_{i \in I} \tau_i$ or $\bigcap\{\tau_i \mid i \in I\}$, where the empty intersection is identified with ω.

If $\tau = \tau_1 \to \cdots \to \tau_n \to \sigma$, then we write $\sigma = tgt_n(\tau)$ and $\tau_i = arg_i(\tau)$, for $i \leq n$. A type of the form $\tau_1 \to \cdots \to \tau_n \to p$, where $p \neq \omega$ is an atom, is called a *path of length n*. A type τ is *organized* if it is a (possibly empty) intersection of paths (those are called *paths in* τ). Note that premises in an organized type do not have to be organized, i.e., organized is not necessarily normalized [6].

Subtyping: Subtyping $\leq$ is the least quasi-order (reflexive and transitive relation), satisfying the following conditions:

$$\sigma \leq \omega, \quad \omega \leq \omega \to \omega, \quad \sigma \cap \tau \leq \sigma, \quad \sigma \cap \tau \leq \tau, \quad \sigma \leq \sigma \cap \sigma;$$

$$(\sigma \to \tau) \cap (\sigma \to \rho) \leq \sigma \to \tau \cap \rho;$$

$$\text{If } \sigma \leq \sigma' \text{ and } \tau \leq \tau' \text{ then } \sigma \cap \tau \leq \sigma' \cap \tau' \text{ and } \sigma' \to \tau \leq \sigma \to \tau'.$$

We identify σ and τ when $\sigma \leq \tau$ and $\tau \leq \sigma$. Note that $\tau \to \omega = \omega$, for all τ. The distributivity properties below follow from the axioms of subtyping:

$$(\sigma \to \tau) \cap (\sigma \to \rho) = \sigma \to (\tau \cap \rho)$$

$$(\sigma \to \tau) \cap (\sigma' \to \tau') \leq (\sigma \cap \sigma') \to (\tau \cap \tau')$$

The following property, probably first stated in [1], is often called *beta-soundness*. Note that the converse is trivially true.

Lemma 1. *Let a_j, for $j \in J$, be atoms.*

1. *If $\bigcap_{i\in I}(\sigma_i \to \tau_i) \cap \bigcap_{j\in J} a_j \leq \alpha$ then $\alpha = a_j$, for some $j \in J$.*
2. *If $\bigcap_{i\in I}(\sigma_i \to \tau_i) \cap \bigcap_{j\in J} a_j \leq \sigma \to \tau$, where $\sigma \to \tau \neq \omega$, then the set $\{i \in I \mid \sigma \leq \sigma_i\}$ is nonempty and $\bigcap\{\tau_i \mid \sigma \leq \sigma_i\} \leq \tau$.*

Lemma 2. *Every type τ has an equivalent organized type $\overline{\tau}$, computable in polynomial time.*

Proof: Define $\overline{a} = a$ if a is an atom, and $\overline{\tau \cap \sigma} = \overline{\tau} \cap \overline{\sigma}$. If $\overline{\sigma} = \bigcap_{i\in I} \sigma_i$ then take $\overline{\tau \to \sigma} = \bigcap_{i\in I}(\tau \to \sigma_i)$. □

In what follows types are assumed to be organized.

Lemma 3. *Let $\bigcap_{i\in I} \tau_i \leq \beta_1 \to \cdots \to \beta_n \to p$, where τ_i are paths. Then there is an $i \in I$ such that $\tau_i = \alpha_1 \to \cdots \to \alpha_n \to p$ and $\beta_j \leq \alpha_j$, for all $j \leq n$.*

Proof: Induction with respect to n, using the beta soundness (Lemma 1). □

Following [7], we define the least upper bound $\sigma_1 \oplus \cdots \oplus \sigma_n$ of organized types $\sigma_1, \ldots, \sigma_n$ as the intersection of all paths of the form

$$(\alpha_1^1 \cap \cdots \cap \alpha_n^1) \to \cdots \to (\alpha_1^k \cap \cdots \cap \alpha_n^k) \to p,$$

where $p \neq \omega$ is an atom and, for all $i \leq n$, type $\alpha_i^1 \to \cdots \to \alpha_i^k \to p$ is a path in σ_i. It should be obvious that $\sigma_i \leq \sigma_1 \oplus \cdots \oplus \sigma_n$, for all $i \leq n$. We show that it is actually the l.u.b.:

Lemma 4. *If $\sigma_i \leq \tau$, for all $i \leq n$, then $\sigma_1 \oplus \cdots \oplus \sigma_n \leq \tau$.*

Proof: We prove the claim for $\tau = \tau^1 \to \cdots \to \tau^k \to p$ being a path. (The general case then follows easily.) By Lemma 3 there are paths $\alpha_i^1 \to \cdots \to \alpha_i^k \to p$ in σ_i such that $\tau^j \leq \alpha_i^j$, for all i, j. Then in $\sigma_1 \oplus \cdots \oplus \sigma_n$ we have a path $\pi = (\alpha_1^1 \cap \cdots \cap \alpha_n^1) \to \cdots \to (\alpha_1^k \cap \cdots \cap \alpha_n^k) \to p$. Since $\tau^j \leq \alpha_1^j \cap \cdots \cap \alpha_n^j$, we obtain $\pi \leq \tau$, whence $\sigma_1 \oplus \cdots \oplus \sigma_n \leq \tau$. □

Type assignment

We consider the standard intersection type assignment system [4] for pure λ-terms without the rule for intersection introduction. The system is shown in Figure 1. As already noted, the system is equivalent to the explicitly typed version of the standard system (see [7]). We have the following generation lemma:

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}(\text{var}) \qquad \frac{\Gamma \vdash M : \tau \to \tau' \quad \Gamma \vdash N : \tau}{\Gamma \vdash (M\ N) : \tau'}(\to\text{E})$$

$$\frac{\Gamma, x : \tau \vdash M : \tau'}{\Gamma \vdash \lambda x\, M : \tau \to \tau'}(\to\text{I}) \qquad \frac{\Gamma \vdash M : \tau \quad \tau \leq \tau'}{\Gamma \vdash M : \tau'}(\leq)$$

Fig. 1: Type system $\lambda(-\cap\text{I})$.

Lemma 5.

1. *If $\Gamma \vdash MN : \tau$ then $\Gamma \vdash M : \sigma \to \tau$, and $\Gamma \vdash N : \sigma$, for some σ.*
2. *If $\Gamma \vdash xN_1 \ldots N_k : \tau$ then $\Gamma(x) \leq \sigma_1 \to \cdots \to \sigma_k \to \tau$, where $\Gamma \vdash N_i : \sigma_i$, for all $i \leq k$. In particular, $\Gamma \vdash x : \tau$ implies $\Gamma(x) \leq \tau$.*
3. *If $\Gamma \vdash \lambda x\, M : \tau$ then there are σ, ρ with $\Gamma, x : \sigma \vdash M : \rho$ and $\sigma \to \rho \leq \tau$.*

Proof: Routine. Type derivation ends with a logical rule followed by subsumption. In part 2 use induction. □

Subject reduction now follows easily:

Lemma 6. *If $\Gamma \vdash M : \sigma$ and $M \twoheadrightarrow_\beta M'$ then $\Gamma \vdash M : \sigma$.*

Proof: Standard induction using the substitution property: If $\Gamma, x : \tau \vdash M : \sigma$ and $\Gamma \vdash N : \tau$ then $\Gamma \vdash M[N/x] : \sigma$. □

System $\lambda(-\cap\text{I})$ has the strong normalization property, as a subsystem of the ordinary intersection type-assignment. It therefore follows from Lemma 6 that an inhabited type must have a normal inhabitant.

Lemma 7. *If $\Gamma \vdash M : \sigma$ then there is a normal form M' with $\Gamma \vdash M' : \sigma$.*

The crucial Lemmas 9 and 11 below characterize types of normal forms. To prove them we need the following simple property:

Lemma 8. *If $\tau \leq \sigma$ and $\Gamma, x:\sigma \vdash M : \rho$ then also $\Gamma, x:\tau \vdash M : \rho$.*

Proof: Routine induction. □

Lemma 9. *The following are equivalent conditions:*

1. $\Gamma \vdash \lambda x\, M : \bigcap_{i\in I}(\tau_i \to \sigma_i)$;
2. $\Gamma,\ x : \bigoplus_{i\in I} \tau_i \ \vdash\ M : \bigcap_{i\in I} \sigma_i$.

Proof: (1 ⇒ 2) By the generation lemma, we have $\Gamma, x:\tau \vdash M : \sigma$, with $\tau \to \sigma \leq \bigcap_{i\in I}(\tau_i \to \sigma_i)$. That is, $\tau_i \leq \tau$ and $\sigma \leq \sigma_i$, for every i. It follows that $\bigoplus_{i\in I} \tau_i \leq \tau$ (by Lemma 4) and $\sigma \leq \bigcap_{i\in I} \sigma_i$, and we use Lemma 8.

(2 ⇒ 1) If (2) then $\Gamma \vdash \lambda x\, M : \bigoplus_{i\in I} \tau_i \to \bigcap_{i\in I} \sigma_i$ and we apply subsumption using the inequality $\bigoplus_{i\in I} \tau_i \to \bigcap_{i\in I} \sigma_i \ \leq\ \bigcap_{i\in I}(\tau_i \to \sigma_i)$. □

Remark 10. Observe that Lemma 9 has the following "paradoxical" consequence: An abstraction of type $\bigcap_{i\in I}(\tau_i \to \sigma_i)$ must have type $\bigcap_{i,j\in I}(\tau_i \to \sigma_j)$. It is also not enough to know that $\Gamma, x:\tau_i \vdash M : \sigma_i$ holds for all $i \in I$ to conclude that $\Gamma \vdash \lambda x\, M : \bigcap_{i\in I}(\tau_i \to \sigma_i)$. The latter remains true even if all σ_i are the same: the conjunction of $x:\tau \vdash M : \sigma$ and $x:\rho \vdash M : \sigma$ does not imply $x : \tau\oplus\rho \vdash M : \sigma$. Indeed, we have $x : (p \to p) \to q \vdash x\mathbf{I} : q$ and $x : (r \to r) \to q \vdash x\mathbf{I} : q$. But, on the other hand, $((p \to p) \to q) \oplus ((r \to r) \to q) = (p \to p) \cap (r \to r) \to q$ and $x : (p \to p) \cap (r \to r) \to q \nvdash x\mathbf{I} : q$, because $\nvdash \mathbf{I} : (p \to p) \cap (r \to r)$.

Lemma 11 (Path Lemma). *The following are equivalent conditions:*

1. *$\Gamma \vdash xN_1 \dots N_k : \tau$;*
2. *There exists a set P of paths in $\Gamma(x)$ such that*
   (a) *$\bigcap_{\pi\in P} tgt_k(\pi) \ \leq\ \tau$;*
   (b) *$\Gamma \vdash N_i : \bigcap_{\pi\in P} arg_i(\pi)$, for all $i \leq k$.*

Proof: (1 ⇒ 2) Similar to the proof of Lemma 10 in [12]. By Lemma 5 we have $\Gamma(x) \leq \sigma_1 \to \cdots \to \sigma_k \to \tau$, where $\Gamma \vdash N_i : \sigma_i$, for all $i \leq k$. Now, assume that $\Gamma(x)$ is organized and let ρ be the organized form of $\sigma_1 \to \cdots \to \sigma_k \to \tau$. We apply lemma 3 to every path in ρ. This yields a set P of paths in $\Gamma(x)$ such that, for every path μ in ρ, there is $\pi \in P$ with $\sigma_i \leq arg_i(\pi)$ and also $tgt_k(\pi) \leq tgt_k(\mu)$. Since all paths in (organized) τ are of the form $tgt_k(\mu)$, we obtain $\bigcap_{\pi\in P} tgt_k(\pi) \leq \tau$ (part 2a). Part 2b follows by subsumption.

(2 ⇒ 1) Easy (as in [12]). □

3 EXPSPACE-completeness of inhabitation

We study the inhabitation problem for system $\lambda(-\cap\mathrm{I})$, formulated as follows:

Given an environment Γ and a type τ,
does there exist a λ-term M such that $\Gamma \vdash M : \tau$?

Our main result, Theorem 19 states that the problem is EXPSPACE-complete. We prove the exponential space upper bound in Section 3.1 and the lower bound in Section 3.2.

The upper bound is achieved by constructing an alternating exponential time algorithm. We exploit the fact, also noted in [7], that least upper bounds of types with respect to $\leq$ are definable. Our algorithm is quite different from the algorithm of [7], which relies on a fixpoint construction that reduces inhabitation to the emptiness problem for a class of context-free grammars. No specific upper bound was given in [7].

The lower bound is achieved by reduction from a slight variation of the *bus machine* model introduced in [17] which generically simulates alternating exponential time Turing machines. The reduction exploits distributivity properties of the subtyping relation together with the definability of least upper bounds.

3.1 EXPSPACE upper bound

Algorithm

Lemmas 9 and 11 together define an alternating algorithm to check type inhabitation in system $\lambda(-\cap\mathrm{I})$. The algorithm is formalized in Figure 2, which specifies an alternating Turing machine accepting on input Γ and τ if and only if there exists a normal inhabitant of τ in the environment Γ. (Note that, by Lemma 7, we only need to consider inhabitants in normal form.)

Recall ([3], see also, e.g., [10, section 16.2]) that the state set Q of an alternating Turing machine is partitioned into two subsets, $Q = Q_\exists \cup Q_\forall$. States in $Q_\exists$ are referred to as existential states, and states in $Q_\forall$ are referred to as universal states. A configuration whose state is in $Q_\forall$ is accepting if and only if all its successor configurations are accepting, and a configuration whose state is in $Q_\exists$ is accepting if and only if at least one of its successor configurations is accepting.

In the specification in Figure 2 we use shorthand notation for instruction sequences starting from existential states (CHOOSE ...) and instruction sequences starting from universal states (FORALL$(i = 1 \ldots k)\, S_i$). For example, a command of the form CHOOSE $x \in S$ branches from an existential state (associated with the CHOOSE-command) to successor states in which x gets assigned distinct elements of S. A command of the form FORALL$(i = 1 \ldots k)\, S_i$ branches from a universal state (associated with the FORALL-command) to successor states from which each instruction sequence S_i is executed.

For an organized type σ, we let $\|\sigma\|$ denote the maximal length of a path in σ. The symbol $\mathbb{P}_k(\sigma)$ denotes the set of all paths of length k or more in organized type σ.

```
      Input :  Γ, τ

1     G := {(Γ, τ)};

2     loop :
3     τ := organize(τ);
4     CHOOSE case ∈ {1, 2};
5     IF (case = 1) THEN
6        IF(τ = ⋂_{i∈I}(τ_i → σ_i)) THEN
7           IF(⨁_{i∈I} τ_i ∉ rn(Γ)) THEN
8              Γ := Γ ∪ {x : ⨁_{i∈I} τ_i} where x is fresh;
9           τ := ⋂_{i∈I} σ_i;
10          IF((Γ, τ) ∈ G) THEN REJECT
11          ELSE
12             G := G ∪ {(Γ, τ)};
13             GOTO loop;
14       ELSE REJECT;

15    IF (case = 2) THEN
16       CHOOSE (x : σ) ∈ Γ;
17       CHOOSE k ∈ {0, . . . , ‖σ‖};
18       CHOOSE P ⊆ ℙ_k(σ);
19       IF(⋂_{π∈P} tgt_k(π) ≤ τ) THEN
20          IF(k = 0) THEN ACCEPT
21          ELSE
22                FORALL(i = 1 . . . k)
23                      τ := ⋂_{π∈P} arg_i(π);
24                      IF((Γ, τ) ∈ G) THEN REJECT
25                      ELSE
26                         G := G ∪ {(Γ, τ)};
27                         GOTO loop;
28       ELSE REJECT;
```

Fig. 2: Alternating EXPTIME-Turing machine

Proposition 12. *Inhabitation in system* $\lambda(-\cap\mathrm{I})$ *is in* EXPSPACE.

Proof: We process problems of the form $\Gamma \vdash \tau$, where τ is organized and every $\Gamma(x)$ is a sum ($\bigoplus$) of organized types. There are two cases, nondeterministically chosen. One is justified by Lemma 9, the other by Lemma 11.

Case 1: For $\tau = \bigcap_{i\in I}(\tau_i \to \sigma_i)$, ask if $\Gamma,\ x : \bigoplus_{i\in I} \tau_i \ \vdash\ \bigcap_{i\in I} \sigma_i$.

Case 2: We guess x and k and a set P of paths in $\Gamma(x)$. That suffices to verify condition (2b) of Lemma 11.

To handle condition (2a) assume for a moment that τ is a path. By Lemma 3, the inequality (2a) holds if and only if we have $tgt_k(\pi) \leq \tau$ for a single $\pi \in P$. If τ is an intersection of paths, we need one $\pi \in P$ for each.

At all stages, we ask questions of the form $\Gamma \vdash \tau$, where τ is an intersection of (organized form of) disjoint subterms of the original type. It takes linear space to write down each such τ. Types in Γ are sums of such disjoint subterms. The environment can only grow and one can optimize by assuming that no type is repeated (Figure 2 line 7). There are at most exponentially many such intersections and sums.

The variable G is an auxiliary device to prevent repeating goals. As inhabitation goals (Γ, τ) are generated (starting with the input goal), they are accumulated in G along branches of the computation tree to ensure that a goal (Γ, τ) is only considered in a branch if it has not already been considered in the branch.

It follows that the depth of alternating paths of computation are exponentially bounded and hence the (alternating) time used by the procedure is at most exponential. The proposition now follows from the well-known identity AEXPTIME = EXPSPACE [3, Corollary 3.6]. □

3.2 EXPSPACE lower bound

The lower bound is obtained by encoding the halting problem for bus machines into the inhabitation problem. A bus machine is an alternating computing device operating on a finite word (bus) of a fixed length. At every step the whole contents of the bus is updated according to one of the instructions of the machine. In addition new instructions may be created each time to be used in the future. A precise definition is as follows.

A *simple switch* over a finite alphabet $\mathcal{A}$ is a pair of elements of $\mathcal{A}$, written $a \leftarrow b$. A *labeled switch* is a quadruple, written $a \leftarrow b(c \leftarrow d)$, where the simple switch $c \leftarrow d$ is the *label*. Finally, a *universal switch* is a triple, written $a \leftarrow b \times c$.

Formally, a *bus machine* is a tuple $\mathcal{M} = \langle\, \mathcal{A}, m, w_0, w_1, \mathcal{I} \,\rangle$, where $\mathcal{A}$ is a finite alphabet, $m > 0$ is the *bus length* of $\mathcal{M}$ (the length of the words processed), w_0 and w_1 are words of length m over $\mathcal{A}$, called the *initial* and *final word*, respectively, and $\mathcal{I}$ is the set of *global instructions*.

Every global instruction is an m-tuple $\mathbb{I} = \langle\, I_1, \ldots, I_m \,\rangle$ of sets of switches. Switches in I_i are meant to act on the i-th symbol of the bus. It is required that all switches in a given instruction $\mathbb{I}$ are of the same kind: either all are simple, or all are labeled, or all are universal. Therefore we classify instructions as simple, labeled, and universal. A *local instruction* is an m-tuple of simple switches and is considered a special case of a simple instruction (singletons at all coordinates).

A *configuration* of $\mathcal{M}$ is a pair $\langle\, w, \mathcal{J} \,\rangle$, where w is a word over $\mathcal{A}$ of length m, and $\mathcal{J}$ is a set of local instructions. The *initial* configuration is of course $\langle\, w_0, \varnothing \,\rangle$, and any configuration of the form $\langle\, w_1, \mathcal{J} \,\rangle$ is called *final*.

Suppose that $\mathbb{I} = \langle\, I_1, \ldots, I_m \,\rangle$, and let $w = a_1 \ldots a_m$ and $w' = b_1 \ldots b_m$, $w'' = c_1 \ldots c_m$.

- If $\mathbb{I}$ is a simple instruction, and for every $i \leq m$ the simple switch $a_i \leftarrow b_i$ belongs to I_i, then $\langle\, w, \mathcal{J} \,\rangle \Rightarrow^{\mathbb{I}}_{\mathcal{M}} \langle\, w', \mathcal{J} \,\rangle$;
- If for every $i \leq m$ there is $a_i \leftarrow b_i(c_i \leftarrow d_i)$ in I_i, then $\langle\, w, \mathcal{J} \,\rangle \Rightarrow^{\mathbb{I}}_{\mathcal{M}} \langle\, w', \mathcal{J}' \,\rangle$, where $\mathcal{J}' = \mathcal{J} \cup \{\langle\, c_1 \leftarrow d_1, \ldots, c_m \leftarrow d_m \,\rangle\}$;

– If $\mathbb{I}$ is universal and $a_i \leftarrow b_i \times c_i$ is in I_i, for $i \leq m$, then $\langle\, w, \mathcal{J} \,\rangle \Rightarrow^{\mathbb{I}}_{M} \langle\, w', \mathcal{J} \,\rangle$, and also $\langle\, w, \mathcal{J} \,\rangle \Rightarrow^{\mathbb{I}}_{M} \langle\, w'', \mathcal{J} \,\rangle$.

A configuration $\langle\, w, \mathcal{J} \,\rangle$ is *accepting* iff it is either a final configuration, or

– There exists a non-universal instruction $\mathbb{I}$, such that $\langle\, w, \mathcal{J} \,\rangle \Rightarrow^{\mathbb{I}}_{\mathcal{M}} \langle\, w', \mathcal{J}' \,\rangle$ and $\langle\, w', \mathcal{J}' \,\rangle$ is accepting, or
– There is a universal instruction $\mathbb{I}$ such that we have $\langle\, w, \mathcal{J} \,\rangle \Rightarrow^{\mathbb{I}}_{\mathcal{M}} \langle\, w', \mathcal{J} \,\rangle$ and $\langle\, w, \mathcal{J} \,\rangle \Rightarrow^{\mathbb{I}}_{\mathcal{M}} \langle\, w'', \mathcal{J} \,\rangle$, where both $\langle\, w', \mathcal{J} \,\rangle$ and $\langle\, w'', \mathcal{J} \,\rangle$ are accepting.

The machine $\mathcal{M}$ *halts* iff the initial configuration is accepting.

Example 13. This example is repeated after [17], and is based on the ideas of [8]. Let $\mathcal{A} = \{0, 1\}$, $I = \{0 \leftarrow 0, 1 \leftarrow 1\}$, $I^+ = \{0 \leftarrow 1\}$, and $I^- = \{1 \leftarrow 0\}$. Take $\mathcal{M} = \langle\, \mathcal{A}, 4, 0000, 1111, \mathcal{I} \,\rangle$, where $\mathcal{I}$ consists of the following tuples (note that all global instructions are simple).

$$(I, I, I, I^+),\ (I, I, I^+, I^-),\ (I, I^+, I^-, I^-),\ (I^+, I^-, I^-, I^-).$$

The machine $\mathcal{M}$ makes $2^4 - 1$ steps and halts after it has seen all binary strings of length 4.

Example 14. This example demonstrates how local instructions work. We modify Example 13 so that $\mathcal{A} = \{0, 1, 2, 3\}$ and

$$I = \{0 \leftarrow 0(2{\leftarrow}2), 1 \leftarrow 1(3{\leftarrow}3)\}, \quad I^+ = \{0 \leftarrow 1(2{\leftarrow}3)\}, \quad I^- = \{1 \leftarrow 0(3{\leftarrow}2)\}.$$

In addition, let $I^* = \{1 \leftarrow 2\}$. Our new machine is $\mathcal{M} = \langle\, \mathcal{A}, 4, 0000, 3333, \mathcal{I} \,\rangle$, where $\mathcal{I}$ consists of the following tuples

$$(I, I, I, I^+),\ (I, I, I^+, I^-),\ (I, I^+, I^-, I^-),\ (I^+, I^-, I^-, I^-),\ (I^*, I^*, I^*, I^*).$$

For the first $2^4 - 1$ steps the machine behaves as in Example 13, using global instructions only. This time however, every application of a global instruction creates a new unique local instruction. After arriving at 1111 the machine rewrites the bus to 2222 and then executes one by one all the local instructions, finally reaching the final 3333. Observe that the number of local instructions is exponential and so is the (implicit) space needed to store them.

The following result was shown in [17].

Proposition 15. *The halting problem for bus machines is* Expspace*-complete.*

The proof of Proposition 15 makes an essential use of both alternation and the ability of bus machines to create local instructions. Without the latter (when all global instructions are either simple or universal) the problem turns out to be complete in Exptime, and that is implicit in [8], because such bus machines are of the same power as alternating linear bounded automata.
Without alternation, as usual, the problems classify one step lower in the hierarchy (as Exptime- and Pspace-complete, respectively).

We need a slightly improved version of Proposition 15. Call a bus machine $\mathcal{M} = \langle\, \mathcal{A}, m, w_0, w_1, \mathcal{I} \,\rangle$ *separated*, when the alphabet $\mathcal{A}$ is a union of m disjoint alphabets $\mathcal{A}_i$ and all links in the i-th component of every instruction are built exclusively from symbols in $\mathcal{A}_i$. We have

Corollary 16. *The halting problem for separated bus machines is* EXPSPACE*-complete.*

Proof: Simply replace $\mathcal{A}$ by a disjoint union of m copies of $\mathcal{A}$ and rewrite all switches and the initial and final words accordingly. The new machine halts if and only if the original one halts, because different components of the bus do not interact at all. □

Exponential space hardness

We reduce the halting problem for separated bus machines to the inhabitation problem. The principle of the encoding is similar to that in Section 3.3 of [17]. However, a major difference is that [17] deals with restricted types but unrestricted rules. In particular, rule (∩I) makes it possible to simulate the m coordinates of the bus by using an intersection. To inhabit this intersection one looks for a single term solving the m problems represented by the components of the intersection.

Now, in system $\lambda(-\cap\mathrm{I})$, we have restricted rules. Without rule (∩I) we cannot create a parallel problem. Instead we represent a bus as an intersection of atoms. By making the alphabets disjoint we ensure that an intersection of m atoms represents only one word of length m, because we know that the i-th symbol belongs to the i-th alphabet. Rather than simultaneously switching from a_i to b_i, at every $i = 1, \dots, m$, we switch from $a_1 \cap \dots \cap a_m$ to $b_1 \cap \dots \cap b_m$. To make sure this works correctly, we need a new technique for coding rules, especially labeled rules. For this, a major instrument is Lemma 17 below.

So let $\mathcal{M} = \langle\, \mathcal{A}, m, w_0, w_1, \mathcal{I} \,\rangle$ be separated, and assume that the final word is $f_1 \dots f_m$, and the initial word is $s_1 \dots s_m$. Simple and universal switches of $\mathcal{M}$ are encoded by types of the form $a \to b$ and $a \to b \to c$. A labeled switch $a \leftarrow b(c \leftarrow d)$, with $a, b, c, d \in \mathcal{A}_i$, is encoded by

$$(((d \to c) \cap \bigcap\nolimits_{o \in \mathcal{A} - \mathcal{A}_i} (\omega \to o)) \to b) \to a.$$

The following property is useful.

Lemma 17. *Let $d_i, c_i \in \mathcal{A}_i$. Then*

$$\bigoplus\nolimits_{i=1}^{m} ((d_i \to c_i) \cap \bigcap\nolimits_{o \in \mathcal{A} - \mathcal{A}_i} (\omega \to o)) = \bigcap\nolimits_{i=1}^{m} (d_i \to c_i).$$

Proof: According to the definition of $\bigoplus$, each path in the left-hand side must be of the form $d_i \cap \omega \cap \dots \cap \omega \to c_i$, i.e., must be equal to $d_i \to c_i$. □

A local instruction $J = \langle\, c_1 \leftarrow b_1, \dots, c_m \leftarrow b_m \,\rangle$ is encoded by the type $\alpha_J = \bigcap_{i=1}^{m} (b_i \to c_i)$. A global instruction $\mathbb{I} = \langle\, I_1, \dots, I_m \,\rangle$ is encoded as the intersection of types representing all switches in the union of $I_1, \dots, I_n$. For example, a simple instruction $\mathbb{I} = \langle\, I_1, \dots, I_m \,\rangle$ with $I_j = \{a_k^j \leftarrow b_k^j \mid k = 1, \dots, q_j\}$ for $j = 1, \dots, m$, is represented by the type $\tau^{\mathbb{I}}$ defined by

$$\tau^{\mathbb{I}} = \bigcap \{b_k^j \to a_k^j \mid j = 1 \dots m, k = 1 \dots q_j\}.$$

Assuming $\mathcal{I} = \{\mathbb{I}_1, \ldots, \mathbb{I}_n\}$, we take $\Gamma = \{x_0 : f_1 \cap \cdots \cap f_m,\ x_1 : \tau^1, \ldots, x_n : \tau^n\}$, where each τ^j represents $\mathbb{I}_j$. In addition, let $\Gamma^{\mathcal{J}} = \Gamma \cup \{y_J : \alpha_J \mid J \in \mathcal{J}\}$, for any set $\mathcal{J}$ of local instructions.

A configuration $\langle\, e_1 \ldots e_m, \mathcal{J} \,\rangle$ of the machine is now represented by the inhabitation problem $\Gamma^{\mathcal{J}} \vdash e_1 \cap \cdots \cap e_m$. Here is what we need to prove.

Lemma 18. *The configuration $\langle\, e_1 \ldots e_m, \mathcal{J} \,\rangle$ is accepting if and only if there is a term M satisfying $\Gamma^{\mathcal{J}} \vdash M : e_1 \cap \cdots \cap e_m$.*

Proof: The proof of "only if" goes by induction with respect to the definition of acceptance.

Consider the case $\langle\, e_1 \ldots e_m, \mathcal{J} \,\rangle \Rightarrow^{\mathbb{I}_j}_{\mathcal{M}} \langle\, b_1 \ldots b_m, \mathcal{J} \,\rangle$, where $\mathbb{I}_j = \langle\, I_1, \ldots, I_m \,\rangle$ is simple and where $e_i \leftarrow b_i$ is in the i-th component of $\mathbb{I}_j$, for $i = 1, \ldots, m$, and $\langle\, b_1 \ldots b_m, \mathcal{J} \,\rangle$ is accepting. By induction hypothesis, there exists M such that $\Gamma^{\mathcal{J}} \vdash M : b_1 \cap \cdots \cap b_m$. By construction, we have $(x_j : \tau^j) \in \Gamma^{\mathcal{J}}$ with

$$\tau^j \leq \textstyle\bigcap_{i=1}^m (b_i \to e_i) \leq \bigcap_{i=1}^m b_i \to \bigcap_{i=1}^m e_i$$

where the last inequality follows by distributivity properties of $\leq$. It follows that $\Gamma^{\mathcal{J}} \vdash x_j M : e_1 \cap \cdots \cap e_m$. The similar case of a universal instruction is left out.

Consider the case of a labeled rule $\mathbb{I}_j$ corresponding to the variable x_j. We have a switch $e_i \leftarrow b_i(c_i \leftarrow d_i)$ at every component, and an accepting configuration $\langle\, b_1 \ldots b_m, \mathcal{J} \cup \{J\} \,\rangle$, where $J = \langle\, c_1 \leftarrow d_1, \ldots, c_m \leftarrow d_m \,\rangle$. By the induction hypothesis there is a term N such that

$$\Gamma^{\mathcal{J}}, y_J : \alpha_J \vdash N : b_1 \cap \cdots \cap b_m.$$

By Lemma 17, we have $\alpha_J = \bigoplus_i ((d_i \to c_i) \cap \bigcap_{o \in \mathcal{A} - \mathcal{A}_i} (\omega \to o))$, whence, by Lemma 9,

$$\Gamma^{\mathcal{J}} \vdash \lambda y_J\, N : \textstyle\bigcap_i [((d_i \to c_i) \cap \bigcap_{o \in \mathcal{A} - \mathcal{A}_i} (\omega \to o)) \to b_i]$$

Since $\Gamma(x_j)$ contains all paths $(((d_i \to c_i) \cap \bigcap_{o \in \mathcal{A} - \mathcal{A}_i} (\omega \to o)) \to b_i) \to e_i$, we may conclude from Lemma 11 that $\Gamma^{\mathcal{J}} \vdash x_j(\lambda y_J\, N) : e_1 \cap \cdots \cap e_m$.

The "if" part is shown by induction with respect to M, where M is assumed to be in normal form, cf. Lemma 7. Consider the case of $M = x_j M'$, where $(x_j : \tau^j) \in \Gamma^{\mathcal{J}}$ represents a simple instruction of the form $\mathbb{I}_j = \langle\, I_1, \ldots, I_m \,\rangle$. Since $\Gamma^{\mathcal{J}} \vdash x_j M' : e_1 \cap \cdots \cap e_m$, it follows from Lemma 5 that we must have $\tau^j \leq \zeta \to \bigcap_{i=1}^m e_i$, and $\Gamma^{\mathcal{J}} \vdash M' : \zeta$, for some ζ.

By Lemma 1, we have $\zeta \leq \bigcap\{b_t \mid t \in T\}$, and $\bigcap\{a_t \mid t \in T\} \leq \bigcap_{i=1}^m e_i$, where $\bigcap_{t \in T}(b_t \to a_t)$ is a component of τ^j. Since e_i and a_t are atoms, we actually have $\{e_i \mid 1 \leq i \leq m\} \subseteq \{a_t \mid t \in T\}$. Assume for simplicity that $\{1, \ldots, m\} \subseteq T$, and $e_i = a_i$, for all i. So $\zeta \leq \bigcap\{b_i \mid i = 1 \ldots m\}$ and thus $\Gamma^{\mathcal{J}} \vdash M' : b_1 \cap \cdots \cap b_m$. By induction hypothesis, $\langle\, b_1 \ldots b_m, \mathcal{J} \,\rangle$ is accepting. Moreover, by construction of τ^j together with $\tau^j \leq \zeta \to \bigcap_{i=1}^m e_i$, it must be that $e_i \leftarrow b_i$ is in the i-th component of $\mathbb{I}_j$, for $i = 1 \ldots m$. It follows that $\langle\, e_1 \ldots e_m, \mathcal{J} \,\rangle \Rightarrow^{\mathbb{I}_j}_{\mathcal{M}} \langle\, b_1 \ldots b_m, \mathcal{J} \,\rangle$, and hence $\langle\, e_1 \ldots e_m, \mathcal{J} \,\rangle$ is accepting.

The case of a universal instruction is similar. Again the most complex case is when $M = x(\lambda y\, N)$. If $\Gamma^{\mathcal{J}} \vdash M : e_1 \cap \cdots \cap e_m$ then, for every i, there must be a component $(((a_i \to b_i) \cap \bigcap_{o \in \mathcal{A} - \mathcal{A}_i} (\omega \to o)) \to c_i) \to e_i$ in $\Gamma(x)$ such that

$$\Gamma^{\mathcal{J}} \vdash \lambda y\, N : \bigcap_i (((a_i \to b_i) \cap \bigcap_{o \in \mathcal{A} - \mathcal{A}_i} (\omega \to o)) \to c_i).$$

Using again Lemmas 11 and 17, we see that $\Gamma^{\mathcal{J}}, y : \alpha_J \vdash \bigcap_i c_i$, for an appropriate J, and here we apply the induction hypothesis. □

Theorem 19. *The inhabitation problem for* $\lambda(-\cap\mathrm{I})$ *is* Expspace*-complete.*

Proof: The problem is in Expspace by Proposition 12, hardness follows from Corollary 16. Indeed, by Lemma 18, the machine $\mathcal{M}$ halts if and only if type $\gamma_1 \to \cdots \to \gamma_k \to s_1 \cap \cdots \cap s_m$ is inhabited, where γ_i are all types from the environment Γ. (Note that the initial $\mathcal{J}$ is empty.) □

4 Pspace-completeness without subtyping

Systems of intersection types are often defined without subtyping, but with the intersection elimination rule ($\cap$E) (which is derivable in presence of the subsumption rule). Such a restriction of $\lambda(-\cap\mathrm{I})$ is given in Figure 3.

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}(\mathrm{var}) \qquad \frac{\Gamma \vdash M : \tau \to \tau' \quad \Gamma \vdash N : \tau}{\Gamma \vdash (M\ N) : \tau'}(\to\mathrm{E})$$

$$\frac{\Gamma, x : \tau \vdash M : \tau'}{\Gamma \vdash \lambda x\, M : \tau \to \tau'}(\to\mathrm{I}) \qquad \frac{\Gamma \vdash M : \tau_1 \cap \tau_2}{\Gamma \vdash M : \tau_i}(\cap\mathrm{E})$$

Fig. 3: Type system $\lambda(-\cap\mathrm{I})$ without subtyping.

Define a relation $\sqsubseteq$ as the least quasi-order such that:

- $\tau \cap \sigma \sqsubseteq \tau$;
- If $\sigma \sqsubseteq \sigma'$ then $\tau \to \sigma \sqsubseteq \tau \to \sigma'$.

Lemma 20. *If* $\Gamma \vdash xN_1 \dots N_k : \sigma$ *then* $\Gamma(x) \sqsubseteq \tau_1 \to \cdots \to \tau_k \to \sigma$*, where* $\Gamma \vdash N_i : \tau_i$*, for all* i*.*

Proof: Induction. □

A derivation is *long normal* when it only uses the following rules:

$$\frac{\Gamma(x : \tau) \vdash N : \sigma}{\Gamma \vdash \lambda x\, N : \tau \to \sigma} \qquad \frac{\Gamma \vdash N_i : \tau_i \quad (i = 1, \dots, k)}{\Gamma \vdash xN_1 \dots N_k : \alpha}\ (\Gamma(x) \sqsubseteq \tau_1 \to \cdots \to \tau_k \to \alpha)$$

where α is not an arrow type. The next lemma is almost literally the same as proving that an inhabited simple type has a long normal inhabitant.

Lemma 21. *If $\Gamma \vdash M : \tau$ then there is a long normal derivation of $\Gamma \vdash M' : \tau$, for some M'.*

Proof: Without loss of generality, we can take M normal. We use induction with respect to the size of M. If $\tau = \sigma \to \rho$ and $M = \lambda z\, N$, apply induction to N. If $M = zN_1 \dots N_k$, write τ as $\tau = \rho_1 \to \cdots \to \rho_m \to \alpha$, with α not an arrow type. Use Lemma 20, and apply induction to $\Gamma \vdash N_i : \tau_i$ to obtain terms N_i' such that $\Gamma \vdash N_i' : \tau_i$ have long normal derivations. Finally define M' as the term $\lambda y_1 \dots y_m.\, zN_1' \dots N_k' y_1 \dots y_m$. □

Corollary 22. *Inhabitation in system $\lambda(-\cap\mathrm{I})$ without subtyping is* PSPACE*-complete.*

Proof: By Lemma 21 an inhabited type has an inhabitant with a long normal derivation. Therefore, a PSPACE inhabitation algorithm can be obtained as a slight modification of the ordinary Wajsberg/Ben-Yelles procedure for simple types, as in e.g. [15]. The only difference is that, when looking for a proof of a non-arrow type α, one chooses a "path" of the form $\tau_1 \to \cdots \to \tau_k \to \alpha$ where $\Gamma(x) \sqsubseteq \tau_1 \to \cdots \to \tau_k \to \alpha$ (rather than $\Gamma(x) = \tau_1 \to \cdots \to \tau_k \to \alpha$), for some x. A path is selected nondeterministically by inspecting the tree of $\Gamma(x)$ from the root towards the leaves (may stop at an internal $\cap$-node). The case $\Gamma \vdash \tau \to \sigma$ is reduced as usual to $\Gamma \vdash \sigma$ or $\Gamma, x : \tau \vdash \sigma$, with fresh x, depending on whether $\tau \in rn(\Gamma)$ or not.

The PSPACE-hardness follows easily from PSPACE-hardness of inhabitation in simple types, since $\lambda(-\cap\mathrm{I})$ without subtyping is easily seen to be a conservative extension of the simple typed system $\lambda_\to$ (if Γ contains only simple types, then $\Gamma \vdash M : \tau$ in $\lambda(-\cap\mathrm{I})$ is equivalent to $\Gamma \vdash M : \tau$ in $\lambda_\to$). □

5 Conclusion

We have considered the question of complexity of inhabitation for a natural fragment of the intersection typed λ-calculus, in which intersections are *explicit*, in the sense of "Church-style" type assumptions or, equivalently, in the sense that intersections cannot be introduced. Decidability was shown by Kurata and Takahashi [7], but the question of complexity was left open. In the present paper we have settled the question: the problem is EXPSPACE-complete. We have emphasized the importance of subtyping by pointing out that the problem is PSPACE-complete in the absence of subtyping. Since decidable inhabitation can be seen as a basis for automatic program synthesis problems [12], our result determines the complexity of synthesis for the "Church-style fragment" of the intersection typed λ-calculus. More generally, our result adds to the emerging systematic understanding of the fine structure of intersection type systems.

References

1. H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983.
2. Henk Barendregt, S. Abramsky, D. M. Gabbay, T. S. E. Maibaum, and H. P. Barendregt. Lambda calculi with types. In *Handbook of Logic in Computer Science*, pages 117–309. Oxford University Press, 1992.
3. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.
4. M. Coppo and M. Dezani-Ciancaglini. An extension of basic functionality theory for lambda-calculus. *Notre Dame Journal of Formal Logic*, 21:685–693, 1980.
5. M. Dezani-Ciancaglini and R. Hindley. Intersection types for combinatory logic. *Theoretical Computer Science*, 100(2):303–324, 1992.
6. J. R. Hindley. The simple semantics for Coppo-Dezani-Sallé types. In M. Dezani-Ciancaglini and U. Montanari, editors, *International Symposium on Programming*, volume 137 of *LNCS*, pages 212–226. Springer, 1982.
7. T. Kurata and M. Takahashi. Decidable properties of intersection type systems. In M. Dezani-Ciancaglini and G.D. Plotkin, editors, *TLCA*, volume 902 of *LNCS*, pages 297–311. Springer, 1995.
8. Dariusz Kuśmierek. The inhabitation problem for rank two intersection types. In S. Ronchi Della Rocca, editor, *TLCA*, volume 4583 of *LNCS*, pages 240–254. Springer, 2007.
9. D. Leivant. Polymorphic type inference. In *Proc. 10th ACM Symp. on Principles of Programming Languages*, pages 88–98. ACM, January 1983.
10. Ch. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
11. 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.
12. J. Rehof and P. Urzyczyn. Finite combinatory logic with intersection types. In C.-H. Luke Ong, editor, *TLCA*, volume 6690 of *Lecture Notes in Computer Science*, pages 169–183. Springer, 2011.
13. S. Salvati. Recognizability in the simply typed lambda-calculus. In H. Ono, M. Kanazawa, and R. J. G. B. de Queiroz, editors, *WoLLIC*, volume 5514 of *LNCS*, pages 48–60. Springer, 2009.
14. R. Statman. Intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science*, 9:67–72, 1979.
15. P. Urzyczyn. Inhabitation in typed lambda-calculi (a syntactic approach). In Philippe de Groote, editor, *TLCA*, volume 1210 of *LNCS*, pages 373–389. Springer, 1997.
16. P. Urzyczyn. The emptiness problem for intersection types. *Journal of Symbolic Logic*, 64(3):1195–1215, 1999.
17. P. Urzyczyn. Inhabitation of low-rank intersection types. In P.-L. Curien, editor, *TLCA*, volume 5608 of *LNCS*, pages 356–370. Springer, 2009.
18. P. Urzyczyn. The logic of persistent intersection. *Fundamenta Informaticae*, 103:303–322, 2010.