Intersection Type Matching with Subtyping

Boris Düdder, Moritz Martens, and Jakob Rehof

Technical University of Dortmund, Faculty of Computer Science
{boris.duedder,moritz.martens,jakob.rehof}@cs.tu-dortmund.de

Abstract. Type matching problems occur in a number of contexts, including library search, component composition, and inhabitation. We consider the intersection type matching problem under the standard notion of subtyping for intersection types: Given intersection types τ and σ, where σ is a constant type, does there exist a type substitution S such that $S(\tau)$ is a subtype of σ? We show that the matching problem is NP-complete. NP-hardness holds already for the restriction to atomic substitutions. The main contribution is an NP-algorithm which is engineered for efficiency by minimizing nondeterminism and running in Ptime on deterministic input problems. Our algorithm is based on a nondeterministic polynomial time normalization procedure for subtype constraint systems with intersection types. We have applied intersection type matching in optimizations of an inhabitation algorithm.

1 Introduction

By *intersection type matching* we understand the following decision problem:

Given intersection types τ and σ, where σ does not contain any type variables, is there a type substitution S with $S(\tau) \leq_{\mathbf{A}} \sigma$?

Here the relation $\leq_{\mathbf{A}}$ is the standard theory of subtyping for intersection types as defined in [1].

Generally, type matching may refer to a range of decision problems spanning from various forms of type equivalence [2] to problems involving substitution and subtyping [3]. The *subtype matching* problem considered in this paper is a special case of the subtype *satisfiability* problem, to decide for given τ and σ whether there exists a substitution S such that $S(\tau) \leq S(\sigma)$, where $\leq$ is a subtyping relation (a partial order on types). *Equational* matching problems of the form $\exists S.\ S(\tau) \sim \sigma$, where $\sim$ is some equivalence relation, is a special case of *unification* modulo an equational theory (see, e.g., [4]). Typical applications of type matching include component retrieval and composition [3,2], where τ might for example be the parametric type of a component F and σ the expected type in a usage context C, and where the matching condition $S(\tau) \leq \sigma$ ensures that $C[F]$ is well typed.

Our study of intersection type matching with subtyping is motivated from two perspectives. First, from a systematic standpoint, there is a general need to develop the algorithmic theory of intersection type subtyping. Such a theory is

M. Hasegawa (Ed.): TLCA 2013, LNCS 7941, pp. 125–139, 2013.

sorely missing in the literature (see Sect. 2), and an algorithmic study of matching is one necessary step towards such a theory. Second, from the standpoint of applications, we have applied the results reported in this paper to the problem of inhabitation in systems of combinatory logic with intersection types [5,6] in the context of combinatory logic synthesis (see [7] for an introduction to this research programme). With polymorphic combinators [6] a central step in solving the inhabitation problem $\Gamma \vdash ? : \sigma$ (is there a term with type σ in type environment Γ?) is to decide the condition $S(\tau) \leq \sigma$, where σ is an inhabitation goal and τ is the type of a combinator from Γ. A solution to the matching problem leads to a very substantial optimization of the inhabitation algorithm, since it allows us to filter out uninteresting choices of τ early in the process based on the matching condition. We refer the reader to [8] for details of this application.

We show that the problem of intersection type matching is NP-complete, even when substitutions are restricted to be atomic (mapping type variables to either variables or constants). We present an NP-algorithm which is engineered for efficiency by localizing nondeterminism as much as possible. The core of the algorithm consists in a nondeterministic constraint set normalization procedure. Matching substitutions can be efficiently constructed from normalized constraint systems provided they are consistent. The constraint normalization procedure performs a fine-grained analysis of subtyping constraints generated from the input problem. In the absence of nondeterminism in the input problem, the algorithm operates in Ptime (note that this is obviously far superior to approaches where solutions are simply guessed).

Due to space limitations some proofs have been shortened or left out. They can be found in [8].

2 Related Work

Surprisingly, the algorithmic properties of the standard intersection subtyping relation $\leq_{\mathbf{A}}$, clearly of systematic importance in type theory, do not appear to have been very much investigated in the literature. To the best of our knowledge, there are no tight results for any of the problems, unification, satisfiability, or, indeed, matching, for intersection types under the standard subtyping relation of [1]. This situation is not satisfactory, and with the present paper we take a step towards remedying it. Only recently, a Ptime-procedure for deciding the relation $\leq_{\mathbf{A}}$ itself was given in [5] and is used as a subroutine here (decidability of $\leq_{\mathbf{A}}$ follows from the results of [9], but with an exponential time algorithm).

Subtype satisfiability has been studied in various subtyping theories without intersection. It is particularly useful to compare with results in simple types (an overview can be found in [10]). Satisfiability there is Pspace-complete when arbitrary partial orders of base types are allowed [11,12], but it is in Ptime over lattices of base types [11]. The Ptime result uses the property that consistent constraint systems are satisfiable, because a satisfying substitution can be constructed from such systems using either one of the lattice operations. A similar property is used here (Lem. 3) to solve the intersection type matching problem,

but it is based on a much more intricate constraint normalization procedure. As can indeed be concluded from our NP-completeness result, the presence of intersection changes the problem fundamentally. Let us remark that our technique for solving the matching problem does not transfer to the satisfiability problem with intersection (the reason will become clear in our technical development), and satisfiability could very well lie higher in the complexity hierarchy.

There are various results concerning the complexity of matching and unification in different classes of equational theories (cf. [4] for a general survey), which exhibit similar properties as the type operators $\cap$ and $\to$ (associativity (A), commutativity (C), and idempotence (I), respectively distributivity (D)) and which are NP-complete. For example, AC- and ACI-matching as well as AC- and ACI-unification are NP-complete [13,14,15]. The techniques used to prove membership in NP are completely different from the approach we follow, though. For example, it is easy to show that the size of a substitution solving an AC-matching problem is bounded by the size of the terms involved [13]. Therefore, simply guessing the right substitution yields an NP-algorithm. As discussed in the beginning of Sect. 5 this is not so simple in the case of intersection type matching. Moreover, this approach foregoes any detailed analysis of the sources of nondeterminism and leads to pragmatically suboptimal algorithms. The algorithms deciding AC- and ACI-unification are relatively intricate [15]. They reduce unification to unification in certain commutative semigroups which is known to be NP-complete. This reduction does not appear to be possible in our setting, however.[1] There are semi-decidability results concerning an altogether different notion of unification for intersection types [16,17] where unification is considered along with other operations that can be used to characterize principal typings with intersection types.

3 Preliminaries

Type expressions, ranged over by τ, σ, etc., are defined by $\tau ::= a \mid \tau \to \tau \mid \tau \cap \tau$ where $a, b, p, q \ldots$ range over *atoms* comprising of *type constants*, drawn from a finite set $\mathbb{A}$ including the constant ω, and *type variables*, drawn from a disjoint denumerable set $\mathbb{V}$ ranged over by α, β, etc. We let $\mathbb{T}$ denote the set of all types.

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 ($\tau \cap \tau = \tau$). As a matter of notational convention, function types associate to the right, and $\cap$ binds stronger than $\to$.

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.

The standard [1] intersection type *subtyping* relation $\leq_{\mathbf{A}}$ is the least preorder (reflexive and transitive relation) on $\mathbb{T}$ generated by the following set $\mathbf{A}$

[1] We also note that the results on unification mentioned above do not directly encompass the complexity of intersection type unification modulo the ACID equational theory induced by $\leq_{\mathbf{A}}$.

of axioms:

$$\sigma \leq_{\mathbf{A}} \omega, \quad \omega \leq_{\mathbf{A}} \omega \to \omega, \quad \sigma \cap \tau \leq_{\mathbf{A}} \sigma, \quad \sigma \cap \tau \leq_{\mathbf{A}} \tau, \quad \sigma \leq_{\mathbf{A}} \sigma \cap \sigma;$$
$$(\sigma \to \tau) \cap (\sigma \to \rho) \leq_{\mathbf{A}} \sigma \to \tau \cap \rho;$$
$$\text{If } \sigma \leq_{\mathbf{A}} \sigma' \text{ and } \tau \leq_{\mathbf{A}} \tau' \text{ then } \sigma \cap \tau \leq_{\mathbf{A}} \sigma' \cap \tau' \text{ and } \sigma' \to \tau \leq_{\mathbf{A}} \sigma \to \tau'.$$

We identify σ and τ when $\sigma \leq_{\mathbf{A}} \tau$ and $\tau \leq_{\mathbf{A}} \sigma$. The distributivity properties $(\sigma \to \tau) \cap (\sigma \to \rho) = \sigma \to \tau \cap \rho$ and $(\sigma \to \tau) \cap (\sigma' \to \tau') \leq_{\mathbf{A}} \sigma \cap \sigma' \to \tau \cap \tau'$ follow from the axioms of subtyping. Note also that $\tau_1 \to \cdots \to \tau_m \to \omega = \omega$. We say that a type τ is *reduced with respect to* ω if it has no subterm of the form $\rho \cap \omega$ or $\tau_1 \to \cdots \to \tau_m \to \omega$ with $m \geq 1$. It is easy to reduce a type with respect to ω, by applying the equations $\rho \cap \omega = \rho$ and $\tau_1 \to \cdots \to \tau_m \to \omega = \omega$ left to right.

A type of the form $\tau_1 \to \cdots \to \tau_m \to a$, where $a \neq \omega$ is an atom, is called a *path of length* m. A type τ is *organized* if it is a (possibly empty) intersection of paths (those are called *paths in* τ). Every type τ is equal to an organized type $\overline{\tau}$, computable in polynomial time, with $\overline{a} = a$, $\overline{\tau \cap \sigma} = \overline{\tau} \cap \overline{\sigma}$, and $\overline{\tau \to \sigma} = \bigcap_{i \in I}(\tau \to \sigma_i)$ where $\overline{\sigma} = \bigcap_{i \in I} \sigma_i$. Note that premises in an organized type do not have to be organized, i.e., organized types are not necessarily *normalized* as defined in [9] (in contrast to organized types, the normalized form of a type may be exponentially large in the size of the type).

A *substitution* is a function $S : \mathbb{V} \to \mathbb{T}$, such that S is the identity everywhere but on a finite subset of $\mathbb{V}$. Whenever we consider a substitution S and a type variable α such that $S(\alpha)$ is not explicitly defined we assume $S(\alpha) = \alpha$. A substitution S is tacitly lifted to a function on types, $S : \mathbb{T} \to \mathbb{T}$, by homomorphic extension.

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

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

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

We will need three specializations of this lemma to organized types:

Lemma 2.

1. *Let $\tau = \bigcap_{i \in I} \tau_{i,1} \to \ldots \to \tau_{i,m_i} \to p_i$ be an organized type and let $\sigma = \sigma_1 \to \ldots \to \sigma_m \to p$ be a path. We have $\tau \leq_{\mathbf{A}} \sigma$ if and only if there is an $i \in I$ with $m_i = m$, $\sigma_j \leq_{\mathbf{A}} \tau_{i,j}$ for all $j \leq m$, and $p_i = p$.*
2. *Let $\tau = \bigcap_{i \in I} \tau_{i,1} \to \ldots \to \tau_{i,m_i} \to p_i$ be an organized type and let $\sigma = \sigma_1 \to \ldots \to \sigma_m \to \rho$ be a type with $\rho \neq \omega$. We have $\tau \leq_{\mathbf{A}} \sigma$ if and only if there is a nonempty subset $I' \subseteq I$ such that for all $i \in I'$ and all $1 \leq j \leq m$ we have $\overline{\sigma_j} \leq_{\mathbf{A}} \overline{\tau_{i,j}}$ and such that $\bigcap_{i \in I'} \tau_{i,m+1} \to \ldots \to \tau_{i,m_i} \to p_i \leq_{\mathbf{A}} \rho$.*
3. *Let $\tau = \tau_1 \to \ldots \to \tau_m \to \rho$ be a type and let $\sigma = \sigma_1 \to \ldots \to \sigma_n \to p$ be a path with $m \leq n$. We have $\tau \leq_{\mathbf{A}} \sigma$ if and only if for all $1 \leq j \leq m$ we have $\sigma_j \leq_{\mathbf{A}} \tau_j$ and $\rho \leq_{\mathbf{A}} \sigma_{m+1} \to \ldots \to \sigma_n \to p$.*

Using Lem. 1, the lemmas are proved by induction over m. We conclude by formally defining intersection type matching:

Definition 1. *Let $\tau, \sigma \in \mathbb{T}$ be types and let $\tau \leq \sigma$ be a formal type constraint.*

Let $C = \{\tau_1 \leq \sigma_1, \ldots, \tau_n \leq \sigma_n\}$ be a set of type constraints such that for every i it is the case that σ_i or τ_i does not contain any type variables. We say that C is matchable *if there is a substitution $S : \mathbb{V} \to \mathbb{T}$ such that for all i we have $S(\tau_i) \leq_{\mathbf{A}} S(\sigma_i)$. We say that S* matches *C.*

CMATCH *denotes the decision problem whether a given set of constraints C is matchable.* cMATCH *denotes the decision problem whether a given constraint $\tau \leq \sigma$ where σ does not contain any type variables is matchable.*

We sometimes denote CMATCH and cMATCH as matching problems. Note that in $S(\tau) \leq_{\mathbf{A}} S(\sigma)$ at least one of the two types does not contain variables, i.e., we have $S(\sigma) = \sigma$ or $S(\tau) = \tau$. If it is known that σ does not contain any variables we write $S(\tau) \leq_{\mathbf{A}} \sigma$ (and analogously for τ). Note that we use $\leq$ to denote a formal constraint whose matchability is supposed to be checked whereas $\tau \leq_{\mathbf{A}} \sigma$ states that τ is a subtype of σ.

4 NP-Hardness of Intersection Type Matching

We show that intersection type matching is NP-hard by defining a reduction $\mathcal{R}$ from 3SAT to CMATCH such that any formula F in 3CNF is satisfiable if and only if $\mathcal{R}(F)$ is matchable. Let $F = c_1 \wedge \ldots \wedge c_m$ where for each i we have $c_i = L_i^1 \vee L_i^2 \vee L_i^3$ and each L_i^j is either a propositional variable x or a negation $\neg x$ of such a variable. For all propositional variables x occurring in F we define two fresh type variables called α_x and $\alpha_{\neg x}$. Furthermore, we assume the two type constants 1 and 0. For a given formula F, let $\mathcal{R}(F)$ denote the set containing the following constraints:

1. For all x in F: $\big((1 \to 1) \to 1\big) \cap \big((0 \to 0) \to 0\big) \leq (\alpha_x \to \alpha_x) \to \alpha_x$
2. For all x in F: $(0 \to 1) \cap (1 \to 1) \leq \alpha_x \to 1$
3. For all x in F: $\big((1 \to 1) \to 1\big) \cap \big((0 \to 0) \to 0\big) \leq (\alpha_{\neg x} \to \alpha_{\neg x}) \to \alpha_{\neg x}$
4. For all x in F: $(0 \to 1) \cap (1 \to 1) \leq \alpha_{\neg x} \to 1$
5. For all x in F: $(1 \to 0) \cap (0 \to 1) \leq \alpha_x \to \alpha_{\neg x}$
6. For all c_i: $\alpha_{L_i^1} \cap \alpha_{L_i^2} \cap \alpha_{L_i^3} \leq 1$

It is clear that $\mathcal{R}(F)$ can be constructed in polynomial time.

Theorem 1. *A formula F in* 3CNF *is satisfiable if and only if $\mathcal{R}(F)$ is matchable.*

Proof. For the "only if"-direction let v be a valuation that satisfies F. We define a substitution S_v as follows:

- $S_v(\alpha_x) = v(x)$
- $S_v(\alpha_{\neg x}) = \neg v(x)$

By way of slight notational abuse the right hand sides of these defining equations represent the truth values $v(x)$ and $\neg v(x)$ as types. We claim that S_v matches $\mathcal{R}(F)$. For the first five constraints this is obvious. Consider a clause c_i in F and the corresponding constraint in the sixth group of constraints: because $v(F) = 1$ there is a literal L_i^j with $v(L_i^j) = 1$. Thus, $S_v(\alpha_{L_i^j}) = 1$ and the constraint corresponding to c_i is matched.

For the "if"-direction, from a substitution S matching $\mathcal{R}(F)$ we construct a satisfying valuation v_S for F. We define $v_S(x) = S(\alpha_x)$, and show that v_S is well-defined and satisfies F. Consider a type variable α_x. Using Lem. 1 it is not difficult to show that S can only match the first constraint if $S(\alpha_x) \in \{0, 1, \omega\}$. The second constraint, however, will not be matched if $S(\alpha_x) = \omega$. It is matched by the instantiations $S(\alpha_x) = 0$ and $S(\alpha_x) = 1$, though. Thus, the first two constraints make sure that $S(\alpha_x) \in \{0, 1\}$. The same argument, using the third and fourth constraint, shows $S(\alpha_{\neg x}) \in \{0, 1\}$. These two observations can be used together with the fact that the fifth constraint is matched for x to show that $S(\alpha_x) = 1$ if and only if $S(\alpha_{\neg x}) = 0$ and vice versa. We conclude that v_S is well-defined. In order to show that it satisfies F we need to show that for every clause c_i there is a literal L_i^j with $v_S(L_i^j) = 1$. Because S matches $\mathcal{R}(F)$ we have $S(\alpha_{L_i^1}) \cap S(\alpha_{L_i^2}) \cap S(\alpha_{L_i^3}) \leq_{\mathbf{A}} 1$. Lemma 1 states that the type on the left-hand side must have a component which is equal to 1. We already know that each of the three variables is instantiated either to 0 or 1. Thus, at least one of them must be instantiated to 1. Therefore, $v_S(c_i) = 1$ and v_S satisfies F.

We immediately get the following corollary:

Corollary 1. CMATCH *is* NP*-hard.*

Exploiting co- and contravariance, a set C of constraints can be transformed into a single constraint c such that C is matchable if and only if c is matchable. This yields a reduction from CMATCH to cMATCH, hence the following corollary:

Corollary 2. cMATCH *is* NP*-hard.*

Remark 1. Notice that the lower bound holds even when restricting the matching problem to atomic substitutions (mapping variables to atoms), since S_v as constructed above only uses such substitutions. Furthermore, note that the source of nondeterminism in the matching problem stems from the fact that in the first case of Lem. 1 one has to choose $j \in J$ whereas in the second case a nonempty subset of I has to be chosen. Both cases are reflected in the constraints defined for the reduction $\mathcal{R}$. A nondeterministic choice as in the first case of the lemma has to be made in the constraints of the sixth kind whereas the second case arises in the other constraints resulting in the choice whether a type variable should be substituted by 0 or 1.

5 NP-Membership of Intersection Type Matching

We show that CMATCH and cMATCH are in NP. Interestingly, the NP upper bound is quite challenging. In a first approach, one could try to polynomially

bound the size of substitutions and then guess one nondeterministically. However, proving such a bound turns out to be complicated. Moreover, the nondeterminism exhibited by such an algorithm would completely ignore the sources of nondeterminism that were identified in Rem. 1 and would be pragmatically very suboptimal. Instead, we exploit the special structure of paths and organized types and attempt to minimize nondeterminism. Recall that every type can be organized in polynomial time. Thus, we may assume that all types occurring in a set of constraints are organized. Using the cases of Lem. 2, we successively decompose the set of constraints until we arrive at a set of constraints, that are basic in a certain sense. This process exhibits nondeterministic choices *only* in exactly the same manner as in the lemma with one exception: the case where $\tau \leq \sigma$ is matched by a substitution S with $S(\sigma) = \omega$ is not treated by the lemma and must be dealt with by a separate nondeterministic choice. For a set of basic constraints it is then possible to define a notion of consistency which is similar to the notion of ground consistency defined in [11]. Then, we can use intersections to construct a matching substitution for a consistent set of basic constraints. Since it can be shown that a set of basic constraints resulting from a successful run of the algorithm is matchable if and only if the original set of constraints was matchable, we arrive at an efficient nondeterministic polynomial-time algorithm deciding intersection type matching.

5.1 Algorithm

Our algorithm is shown in the figure Alg. 1 below. It will be seen that Alg. 1 heavily exploits the rather regular structure of organized types. We take a short digression to discuss an alternative strategy, that normalizes [9] types in a set of constraints to be matched and uses directed acyclic graphs (DAGs) to represent these types. Even though normalization may lead to an exponentially large syntax tree this is not per se a problem with regard to the polynomial bound, because a sharing representation of the syntax trees by DAGs only needs polynomial space. We do not, however, follow this approach for two reasons. First, it can be seen that such an approach would need exactly the same decomposition strategy for the sets of constraints that arise in Alg. 1. Furthermore, a DAG-approach requires a normalization of the types in advance. In general, this will cause applications of the distributivity law to types that are eliminated from the set of constraints according to some nondeterministic choices. On the other hand, our algorithm only organizes types in an argument position if necessary, i.e., if a constraint containing such a type is added to the set of constraints. In this case the distributivity law is *only* applied to the target position of paths that are components of the top-level intersection of the type in question. This reflects the structure of Lem. 1 which reduces the test whether a type is a subtype of an arrow-type to a subtype-test for certain types which are arguments, respectively targets, of top-level arrow-types. Thus, there is no reason

to apply the distributivity law to types which are on some lower levels of the syntax tree.[2]

Definition 2. *We call $\tau \leq \sigma$ a* basic *constraint if either τ is a type variable and σ does not contain any type variables or σ is a type variable and τ does not contain any type variables.*

Definition 3. *Let C be a set of basic constraints.*

Let α be a variable occurring in C. Let $\tau_i \leq \alpha$ for $1 \leq i \leq n$ be the constraints in C where α occurs on the right hand side of $\leq$ and let $\alpha \leq \sigma_j$ for $1 \leq j \leq m$ be the constraints in C where α occurs on the left hand side of $\leq$. We say that C is consistent with respect to α *if for all i and j we have $\tau_i \leq_{\mathbf{A}} \sigma_j$.*

We say that C is consistent *if it is consistent with respect to all variables occurring in C.*

Lemma 3. *Let C be a set of basic constraints. The set C can be matched if and only if it is consistent.*

Proof. The implication from left to right is easy and left out. For the direction from right to left, let C be a set of basic constraints and let $\alpha \leq \sigma_j$ for $1 \leq j \leq m$ be the constraints in C where α occurs on the left hand side of $\leq$. We define the substitution S_C by $S_C(\alpha) = \bigcap_{j=1}^{m} \sigma_j$, for every variable α occurring in C. Recall that empty intersections equal ω (thus, S_C is well-defined even if $m = 0$). It can easily be verified that S_C matches C, if C is consistent.

The definition $S_C(\alpha) = \bigcap_{j=1}^{m} \sigma_j$ corresponds to the technique used by Tiuryn [11] for satisfiability in simple types over lattices of type constants, but generalized to arbitrary constant types σ_j (and not just type constants). It is important here that we can treat variables independently since the basic constraints in C do not contain any variables on one side of $\leq$ (hence the types $\bigcap_{j=1}^{m} \sigma_j$ contain no variables). The proof technique would not work for the satisfiability problem with intersection types, where the types on *both* sides of $\leq$ may contain variables.

Alg. 1 below is the nondeterministic procedure mentioned above that decomposes the constraints in the set C until we arrive at a set of basic constraints. According to the lemma above, it suffices to check whether this set of constraints is consistent. A few issues that are not made explicit in the algorithm should be addressed:

[2] Even with respect to a decision procedure for $\leq_{\mathbf{A}}$ the compact representation of normalized types by DAGs does not help: In [9] subtyping for normalized types $\tau = \bigcap_{i \in I} \tau_i$ and $\sigma = \bigcap_{j \in J} \sigma_j$ is characterized by $\tau \leq_{\mathbf{A}} \sigma \Leftrightarrow \forall j \exists i\ \tau_i \leq_{\mathbf{A}} \sigma_j$. Even if the normalized types were represented using DAGs the characterization requires for all j the identification of an index i with $\tau_i \leq_{\mathbf{A}} \sigma_j$. Checking this subtype relation, however, again requires a similar choice of certain components in the types. Repeating this argument, it can be seen that in order to check the characterization above one has to choose certain paths in the DAGs. The number of paths in a DAG however is still exponential if the original normalized type is of exponential size. Since a PTIME-procedure deciding $\leq_{\mathbf{A}}$ is known [5], clearly this approach using DAGs does not lead to an improvement.

1. Memoization is used to make sure that no constraint is added to C more than once.
2. Failing choices always return **false**.
3. The reduction with respect to ω in line 6 means, in particular, that, unless they are syntactically identical to ω, neither τ nor σ contain any types of the form $\rho_1 \to \ldots \to \rho_m \to \omega$ as top-level components.
4. Line 14 subsumes the case $\sigma = \omega$ if $I = \emptyset$. Then c is simply removed from C, and no new constraints are added.
5. We assume that the cases in the **switch**-block are mutually exclusive and checked in the given order. Thus, we know, for example, that for the two cases in lines 19 and 30 σ is *not* an intersection and therefore a path. Thus, we may fix the notation in line 17. Note, though, that the index set I for τ may be empty.
6. In line 21 the choice between the two cases is made nondeterministically. The first option covers the case where $I_2 \cup I_3 = \emptyset$, i.e., all paths in τ are shorter than σ.[3] The only possibility to match such a constraint is to make sure that $S(\sigma) = \omega$, which is only possible if $S(a) = \omega$. Thus, a must be a variable. Clearly, a cannot be a constant different from ω, and it cannot be ω either, because then σ would have been reduced to ω in line 6 and the case in line 8 would have been applicable.
7. The following example shows that even if $I_2 \cup I_3 \neq \emptyset$ it must be possible to choose the first option in line 21: the constraint set $\{a' \leq \beta, b \to a \leq \beta \to \alpha\}$ is matchable according to the substitution $\{\alpha \mapsto \omega, \beta \mapsto a'\}$. If the algorithm were not allowed to choose the option in line 22 it would have to choose the option in the following line which would result in the constraint set $\{a' \leq \beta, a \leq \alpha, \beta \leq b\}$. This set is clearly not matchable and the algorithm would incorrectly return **false**.
8. We assume that the nondeterministic choice in line 21 is made deterministically, choosing the first option whenever $I_2 \cup I_3 = \emptyset$. In this case choosing the second option would always result in **false**.

Nondeterminism results *only* from line 21 and lines 23, 27, and 31. We emphasize that the nondeterminstic choice in line 21 differs from the other cases in that it is the only occurrence of nondeterminism which is not structural in the sense that it can be traced to Lem. 2. Indeed, the first option of this choice covers the case that the type on the right hand side of $\leq_{\mathbf{A}}$ is of the form $\sigma_1 \to \ldots \to \sigma_m \to \omega$ which is excluded in the lemma. This situation can arise if a substitution maps a variable in the target of σ to ω. The other three occurrences of nondeterminism can directly be traced to Lem. 2: the choice of $I' \subseteq I$ in line 23 corresponds to Lem. 2.2 whereas the choice of *one* index in the other two lines can be traced back to Lem. 2.1.

The following example illustrates why it is indeed necessary to choose an index *set* in line 23 as opposed to a single index as in the other two cases: the constraint set $C = \{(a \to b) \cap (a \to p) \leq a \to \alpha, \alpha \leq b \cap p\}$ is matchable with the substitution $\{\alpha \mapsto b \cap p\}$. If, in line 23, the algorithm were only allowed to choose a *single* index from I' this would result in the addition of the new constraints $a \leq a$ and *either*

[3] In particular, $\tau = \omega$ is allowed if furthermore $I_1 = \emptyset$, as well.

Algorithm 1. $\texttt{Match}(C)$

Input: $C = \{\tau_1 \leq \sigma_1, \ldots, \tau_n \leq \sigma_n\}$ such that for all i at most one of σ_i and τ_i contains variables. Furthermore, all types have to be organized.
Output: **true** if C can be matched otherwise **false**

while $\exists$ nonbasic constraint in C **do**
  choose a nonbasic constraint $c = (\tau \leq \sigma) \in C$
  reduce τ and σ with respect to ω
  switch
    case: c does not contain any variables
      if $\tau \leq_{\mathbf{A}} \sigma$ **then**
        $C := C \backslash \{c\}$
      else
        return false
      end if
    case: $\sigma = \bigcap_{i \in I} \sigma_i$
      $C := C \backslash \{c\} \cup \{\tau \leq \sigma_i | i \in I\}$

    write $\tau = \bigcap_{i \in I} \tau_{i,1} \to \ldots \to \tau_{i,m_i} \to p_i$, $\sigma = \sigma_1 \to \ldots \to \sigma_m \to a$
    write $I_1 = \{i \in I | m_i < m\}$, $I_2 = \{i \in I | m_i = m\}$, $I_3 = \{i \in I | m_i > m\}$
    case: σ contains variables, τ does not contain any variables
      if $a \in \mathbb{V}$ **then**
        choose case 1 or 2:
          1. $C := C \backslash \{c\} \cup \{\omega \leq a\}$
          2. **choose** $\emptyset \neq I' \subseteq I_2 \cup I_3$
            $C := C \backslash \{c\} \cup \{\overline{\sigma_j} \leq \overline{\tau_{i,j}} | i \in I', 1 \leq j \leq m\} \cup$
            $\{\bigcap_{i \in I'} \tau_{i,m+1} \to \ldots \to \tau_{i,m_i} \to p_i \leq a\}$
      else
        choose $i_0 \in I_2$
        $C := C \backslash \{c\} \cup \{\overline{\sigma_j} \leq \overline{\tau_{i_0,j}} | 1 \leq j \leq m\} \cup \{p_{i_0} \leq a\}$
      endif
    case: τ contains variables, σ does not contain any variables
      choose $i_0 \in I_1 \cup I_2$
      $C := C \backslash \{c\} \cup \{\overline{\sigma_j} \leq \overline{\tau_{i_0,j}} | 1 \leq j \leq m_{i_0}\} \cup \{p_{i_0} \leq \sigma_{m_{i_0}+1} \to \ldots \to \sigma_m \to a\}$
  end switch
end while
if C is consistent **then**
  return true
else
  return false
end if

$b \leq \alpha$ *or* $p \leq \alpha$. Thus, the resulting set of constraints would be $\{a \leq a, b \leq \alpha, \alpha \leq b \cap p\}$, for example. This set of constraints is not matchable, and it is easy to see that the algorithm would incorrectly return **false** even though the initial set of constraints was matchable. The reason is that, if σ is a path whose target is a variable, a substitution may cause σ to not be a path any more.

As mentioned above, nondeterminism is localized in the algorithm, leading to efficiency. Thus, if for some input C none of the cases above ever arises or, in fact, the choices are deterministic because $I_2 \cup I_3 = \emptyset$, respectively, the index sets to choose from in the other cases are always singleton sets, then the algorithm becomes a PTIME-procedure.

In general, a most general matching substitution is not unique: the constraint set $C = \{\alpha \cap \beta \leq a\}$ is matched by the two substitutions $S_1 = \{\alpha \mapsto a, \beta \mapsto \beta\}$ and $S_2 = \{\alpha \mapsto \alpha, \beta \mapsto a\}$, for example. It is clear that neither of the two substitutions can be obtained from instantiating the other. Furthermore, there is no substitution more general than S_1 and S_2 that still matches C (this substitution would be the identity which clearly does not match C). Note that S_1 and S_2 can directly be traced to the nondeterminstic choice of either α or β in line 31. Thus, only $\alpha \leq a$ or $\beta \leq a$ replaces the original constraint in C. The least upper bound construction from the proof of Lem. 3 then results in S_1 respectively S_2.

Lemma 4. *Algorithm 1 operates in nondeterministic polynomial time.*

Proof. (Sketch) It is clear that the algorithm terminates because in every iteration of the **while**-loop the height of the types involved in a constraint is reduced. Thus, if the algorithm does not return **false**, C only consists of basic constraints after a finite number of iterations.

The algorithm terminates in nondeterministic polynomial time, because (as can be shown by a detailed case analysis) every type occurring in a nonbasic constraint considered during one execution of the **while**-loop is a subterm of a type occurring in an initial constraint (we call such types *initial* types) or a subterm of an organized argument of an initial type. The number of subterms of an initial type is linear. The number of arguments of an initial type is also linear. Organizing each of these linearly many arguments causes only a polynomial blowup. Therefore, it is clear that the number of subterms of an organized argument of an initial type is polynomial. Let k denote the number of subterms of the initial types plus the number of subterms of organized arguments of the initial types. The total number of nonbasic constraints that the algorithm considers is therefore bounded by k^2 which, in turn, is polynomial in the size of C. Furthermore, it can also be seen that every constraint that is considered is of polynomial size. The consistency check and the check in line 9 require checking the truth of constraints $\tau_i \leq \sigma_j$ not containing any type variables, which means deciding the relation $\leq_{\mathbf{A}}$. As is shown in [5] this can be done in PTIME.

The statement of the previous lemma might come as a surprise since the execution of the **while**-loop requires a repeated organization of the arguments of the occurring types. It can be asked why this repeated organization does not result in a normalization [9] of the types involved. As mentioned before, this could cause

an exponential blowup in the size of the type. The reason why this problem does not occur is the fact that this organization is interleaved with decomposition steps. We illustrate this by the following small example. We inductively define two families of types:

$$\tau_0 = a_0 \cap b_0 \qquad \sigma_0 = \alpha_0$$
$$\tau_l = \tau_{l-1} \to (a_l \cap b_l) \qquad \sigma_l = \sigma_{l-1} \to \alpha_l$$

The size of τ_n in normalized form is exponential in n. However, if the algorithm processes the constraint $\tau_n \leq \sigma_n$ only a polynomial number of new constraints (of polynomial size) are constructed: First, the types have to be organized. We obtain $(\tau_{n-1} \to a_n) \cap (\tau_{n-1} \to b_n) \leq \sigma_n$. In the first iteration of the **while**-loop the case in line 20 applies and the nondeterministic choice in line 21 may be resolved in such a way that a subset of components of the toplevel intersection of $(\tau_{n-1} \to a_n) \cap (\tau_{n-1} \to b_n)$ has to be chosen. In order to maximize the size of C we choose both components which forces the construction of the constraints $a_n \cap b_n \leq \alpha_n$, $\overline{\sigma_{n-1}} \leq \overline{\tau_{n-1}}$, and $\overline{\sigma_{n-1}} \leq \overline{\tau_{n-1}}$. The last two constraints are the same, however, and therefore the memoization of the algorithm makes sure that this constraint is only treated once. In the next step the case in line 14 applies (note that $\overline{\tau_{n-1}}$ is a top-level intersection). This leads to the construction of the constraints $\overline{\sigma_{n-1}} \leq \tau_{n-2} \to a_{n-1}$ and $\overline{\sigma_{n-1}} \leq \tau_{n-2} \to b_{n-1}$. For both constraints the same rule applies and causes a change of C according to line 32. This leads to the construction of the basic constraints $\alpha_{n-1} \leq a_{n-1}$ and $\alpha_{n-1} \leq b_{n-1}$ as well as to the construction of $\overline{\sigma_{n-2}} \leq \overline{\tau_{n-2}}$ and $\overline{\sigma_{n-2}} \leq \overline{\tau_{n-2}}$. Then, the same argument as above can be repeated.

We conclude that the doubling of the arguments of the τ_l that occurs in the normalization (and which eventually causes the exponential blowup if repeated) does not occur in the algorithm, because the types involved are decomposed such that the arguments and targets are treated separately. This implies that the arguments cannot be distinguished any more such that the new constraints coincide and are only added once.

The proof of Lem. 4 relies on the fact that every new nonbasic constraint added to C only contains types that are essentially subterms of an initial type. A new intersection which possibly does *not* occur as a subterm in any of the initial types has to be constructed in line 25, though. Since, in principle, this new intersection represents a subset of an index set, it is not clear that there cannot be an exponential number of such basic constraints. However, this construction of new intersections only happens as a consequence to the nonbasic constraint that is treated there. As noted above there can be at most a polynomial number of nonbasic constraints and therefore new intersections can also only be introduced a polynomial number of times.

5.2 Correctness

We consider correctness of the algorithm, i.e., we have to show that it can return **true** if and only if the original set of constraints can be matched. For soundness we first state the following lemma:

Lemma 5. *Let C be a set of constraints and let C' be a set of constraints that results from C by application of one of the cases of Alg. 1.*

Every substitution that matches C' also matches C.

The proof consists of a case analysis. To give the reader an idea we give a proof sketch which discusses one of the cases in more detail.

Proof. For all cases C' results from C by removing the constraint c and possibly by further adding some new constraints. Assuming we have a substitution S that matches C', it suffices to show that S satisfies c in order to show that it also satisfies C.

The argument is not difficult for the cases occurring because of lines 10, 15, and 22. The interesting cases occur because of lines 24, 28, and 32. We will discuss the case in line 24 in detail. The other cases are argued similarly. We then have $\sigma = \sigma_1 \to \ldots \to \sigma_m \to a$, with $a = \alpha$ a type variable, and $\tau = \bigcap_{i \in I} \tau_i$ with $\tau_i = \tau_{i,1} \to \ldots \to \tau_{i,m_i} \to p_i$. In this case C' is constructed by adding $\overline{\sigma_j} \leq \overline{\tau_{i,j}}$ for all $i \in I'$ and all $1 \leq j \leq m$ and $\bigcap_{i \in I'} \tau_{i,m+1} \to \ldots \to \tau_{i,m_i} \to p_i \leq \alpha$. Since S matches C' we know $S(\overline{\sigma_j}) \leq_{\mathbf{A}} \overline{\tau_{i,j}}$ and $\bigcap_{i \in I'} \tau_{i,m+1} \to \ldots \to \tau_{i,m_i} \to p_i \leq_{\mathbf{A}} S(\alpha)$. We want to show $\tau \leq_{\mathbf{A}} S(\sigma)$. We write $S(\sigma) = \sigma'_1 \to \ldots \to \sigma'_m \to \rho$ where $S(\sigma_j) = \sigma'_j$ and $S(\alpha) = \rho$. Using $S(\overline{\sigma_j}) \leq_{\mathbf{A}} \overline{\tau_{i,j}}$, it can be shown[4] that $\overline{\sigma'_j} \leq_{\mathbf{A}} \overline{\tau_{i,j}}$. We may apply the "if"-part of Lem. 2.2 to conclude $\tau \leq_{\mathbf{A}} S(\sigma)$.

Corollary 3. *Algorithm 1 is sound.*

Proof. Assume that the algorithm returns **true**. This is only possible in line 36 if the algorithm leaves the **while**-loop with a consistent set C of basic constraints. By the "if"-direction of Lem. 3, C is matchable. Using Lem. 5, an inductive argument shows that *all* sets of constraints considered in the algorithm during execution of the **while**-loop are matchable. This is in particular true for the initial set of constraints.

For completeness we need the following lemma:

Lemma 6. *Let C be matchable and $c \in C$.*

There exists a set of constraints C' such that C' results from C by application of one of the cases of Alg. 1 to c and C' is matchable.

Again we give a sketch of the case analysis necessary for the proof:

Proof. We show that no matter which case applies to c, a choice can be made that results in a matchable set C'. In particular, it must be argued that there is a choice that does not result in **false**.

As above, note that for all cases C' results from C by removing c and by possibly adding some new constraints. Let S be a substitution that matches C. In order to show that a choice is possible such that S also matches C' it suffices to show that S matches the newly introduced constraints. The argument is not

[4] At this point a technical lemma showing that for a type σ we have $S(\overline{\sigma}) = S(\sigma)$ has to be proven.

difficult for the cases in lines 8 and 14 (and besides there is no nondeterministic choice involved, here). For the cases in lines 19 and 30 subcase-analyses are necessary. The subcases different from the one in line 22 (explained below) are similar to the following:

We consider the case in line 30. We have $\tau = \bigcap_{i\in I} \tau_i$ where $\tau_i = \tau_{i,1} \to \ldots \to \tau_{i,m_i} \to p_i$ and $\sigma = \sigma_1 \to \ldots \to \sigma_m \to a$, and we know $S(\tau) \leq_{\mathbf{A}} \sigma$. We organize $S(\tau)$ and we write $\overline{S(\tau)} = \bigcap_{h\in H} \rho_{h,1} \to \ldots \to \rho_{h,n_h} \to b_h$. Using Lem. 2.1, we conclude that there exists $h_0 \in H$ with $b_{h_0} = a$, $n_{h_0} = m$, and $\sigma_l \leq_{\mathbf{A}} \rho_{h_0,l}$ for all $1 \leq l \leq m$. Note that with $\pi_{h_0} = \rho_{h_0,1} \to \ldots \to \rho_{h_0,m} \to a$ this implies $\pi_{h_0} \leq_{\mathbf{A}} \sigma$. For π_{h_0} there must be an index $j_0 \in I$ such that π_{h_0} occurs as a component in $\overline{S(\tau_{j_0})}$. It is clear that $m_{j_0} \leq m$. Otherwise all paths in this type would be of length greater than m (neither a substitution nor an organization may reduce the length of a path). Thus, j_0 as above is contained in $I_1 \cup I_2$ (cf. line 18), and we may choose $i_0 = j_0$ in line 31.

We have to show that S matches $\overline{\sigma_l} \leq \overline{\tau_{j_0,l}}$ for all $1 \leq l \leq m_{j_0}$ and $p_{j_0} \leq \sigma_{m_{j_0}+1} \to \ldots \to \sigma_m \to a$. Because π_{h_0} is a component in $\overline{S(\tau_{j_0})}$, it is clear that $\pi_{h_0} \leq_{\mathbf{A}} \sigma$ implies $\overline{S(\tau_{j_0})} \leq_{\mathbf{A}} \sigma$ and, thus, also $S(\tau_{j_0}) \leq_{\mathbf{A}} \sigma$. Applying the "only if"-part of Lem. 2.3 to $S(\tau_{j_0,1}) \to \ldots \to S(\tau_{j_0,m_{j_0}}) \to S(p_{j_0}) = S(\tau_{j_0}) \leq_{\mathbf{A}} \sigma = \sigma_1 \to \ldots \to \sigma_m \to a$, we obtain $\sigma_l \leq_{\mathbf{A}} S(\tau_{j_0,l})$ for all $1 \leq l \leq m_{j_0}$ and $S(p_{j_0}) \leq_{\mathbf{A}} \sigma_{m_{j_0}+1} \to \ldots \to \sigma_m \to a$. It can be shown[5] that from $\sigma_l \leq_{\mathbf{A}} S(\tau_{j_0,l})$ we get $\overline{\sigma_l} \leq_{\mathbf{A}} S(\overline{\tau_{j_0,l}})$. Altogether this shows that S matches the newly introduced constraints.

Concerning line 22, if $\tau \leq_{\mathbf{A}} S(\sigma)$ because a is a variable with $S(a) = \omega$ the nondeterminstic choice (line 21) is resolved such that the first option is chosen. Then, it is clear that the newly added constraint $\omega \leq a$ is matched by S.

Corollary 4. *Algorithm 1 is complete.*

Proof. We assume that the initial set C of constraints is matchable. We have to show that there is an execution sequence of the algorithm that results in **true**. Using Lem. 6 in an inductive argument it can be shown that for every iteration of the **while**-loop it is possible to make the nondeterministic choice in such a way that the iteration results in a matchable set of constraints. Thus, there is an execution sequence of the **while**-loop that results in a matchable set of basic constraints. Lemma 3 shows that this set is consistent and therefore the algorithm returns **true**.

Corollaries 2, 3, and 4 and Lem. 4 immediately prove the following theorem:

Theorem 2. cMATCH *and* CMATCH *are* NP*-complete.*

6 Conclusion and Future Work

We have proven the intersection type matching problem to be NP-complete and have provided an algorithm which is engineered for efficiency by reducing

[5] Again using the lemma, showing that $S(\overline{\sigma}) = S(\sigma)$.

nondeterminism as much as possible. Future work includes further experiments to study optimizations of the inhabitation algorithm in [6] based on matching, and studying the satisfiability and unification problems with intersection types.

References

1. Barendregt, H., Coppo, M., Dezani-Ciancaglini, M.: A Filter Lambda Model and the Completeness of Type Assignment. Journal of Symbolic Logic 48(4), 931–940 (1983)
2. Jha, S., Palsberg, J., Zhao, T.: Efficient Type Matching. In: Nielsen, M., Engberg, U. (eds.) FOSSACS 2002. LNCS, vol. 2303, pp. 187–204. Springer, Heidelberg (2002)
3. Zaremski, A., Wing, J.: Signature Matching: A Tool for Using Software Libraries. ACM Trans. Softw. Eng. Methodol. 4(2), 146–170 (1995)
4. Baader, F., Snyder, W.: Unification Theory. In: Handbook of Automated Reasoning, ch. 8, pp. 439–526. Elsevier (2001)
5. Rehof, J., Urzyczyn, P.: Finite Combinatory Logic with Intersection Types. In: Ong, L. (ed.) TLCA 2011. LNCS, vol. 6690, pp. 169–183. Springer, Heidelberg (2011)
6. Düdder, B., Martens, M., Rehof, J., Urzyczyn, P.: Bounded Combinatory Logic. In: Proceedings of CSL 2012. LIPIcs, vol. 16, pp. 243–258. Schloss Dagstuhl (2012)
7. Rehof, J.: Towards Combinatory Logic Synthesis. In: 1st International Workshop on Behavioural Types, BEAT 2013, January 22 (2013), http://beat13.cs.aau.dk/pdf/BEAT13-proceedings.pdf
8. Düdder, B., Martens, M., Rehof, J.: Intersection Type Matching and Bounded Combinatory Logic (Extended Version). Technical Report 841, Faculty of Computer Science, TU Dortmund (2012), http://ls14-www.cs.tu-dortmund.de/index.php/Jakob_Rehof_Publications#Technical_Reports
9. Hindley, J.R.: The Simple Semantics for Coppo-Dezani-Sallé Types. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) Programming 1982. LNCS, vol. 137, pp. 212–226. Springer, Heidelberg (1982)
10. Rehof, J.: The Complexity of Simple Subtyping Systems. PhD thesis, DIKU, Department of Computer Science (1998)
11. Tiuryn, J.: Subtype Inequalities. In: Proceedings of LICS 1992, pp. 308–315. IEEE Computer Society (1992)
12. Frey, A.: Satisfying Subtype Inequalities in Polynomial Space. Theor. Comput. Sci. 277(1-2), 105–117 (2002)
13. Benanav, D., Kapur, D., Narendran, P.: Complexity of Matching Problems. J. Symb. Comput. 3(1/2), 203–216 (1987)
14. Kapur, D., Narendran, P.: NP-Completeness of the Set Unification and Matching Problems. In: Siekmann, J.H. (ed.) CADE 1986. LNCS, vol. 230, pp. 489–495. Springer, Heidelberg (1986)
15. Kapur, D., Narendran, P.: Complexity of Unification Problems with Associative-Commutative Operators. J. Autom. Reasoning 9(2), 261–288 (1992)
16. Ronchi Della Rocca, S.: Principal Type Scheme and Unification for Intersection Type Discipline. Theor. Comput. Sci. 59, 181–209 (1988)
17. Kfoury, A.J., Wells, J.B.: Principality and Type Inference for Intersection Types Using Expansion Variables. Theor. Comput. Sci. 311(1-3), 1–70 (2004)