The Complexity of Subtype Entailment for Simple Types

Fritz Henglein and Jakob Rehof
DIKU, Department of Computer Science
Universitetsparken 1, DK-2100 Copenhagen Ø, Denmark
Electronic mail: {henglein,rehof}@diku.dk
Phone: +45 35321408. Fax: +45 35321405

Abstract

A subtyping $\tau \leq \tau'$ is *entailed* by a set of subtyping constraints C, written $C \models \tau \leq \tau'$, if every valuation (mapping of type variables to ground types) that satisfies C also satisfies $\tau \leq \tau'$.

We study the complexity of subtype entailment for *simple* types over *lattices* of base types. We show that:

- deciding $C \models \tau \leq \tau'$ is coNP-complete.
- deciding $C \models \alpha \leq \beta$ for consistent, atomic C and α, β atomic can be done in linear time.

The structural lower (coNP-hardness) and upper (membership in coNP) bounds as well as the optimal algorithm for atomic entailment are new. The coNP-hardness result indicates that entailment is strictly harder than satisfiability, which is known to be in PTIME for lattices of base types. The proof of coNP-completeness, on the other hand, gives an improved algorithm for deciding entailment and puts a precise complexity-theoretic marker on the intuitive "exponential explosion" in the algorithm.

Central to our results is a novel characterization of $C \models \alpha \leq \beta$ for atomic, consistent C. This is the basis for correctness of the linear-time algorithm, and it shows how to axiomatize completely $C \models \alpha \leq \beta$ by extending the usual proof rules for subtype inference.

1 Introduction

1.1 Subtype entailment: Relevance and related work

Subtyping is a fundamental concept in the theory of typed programming languages. Its basic principles are typically studied in extensions of the simply typed lambda calculus. Following this line of research, the present paper studies a basic problem in the standard system of structural subtyping defined by Mitchell [12, 13] and subsequently studied extensively by many others (see references below.)

In structural subtyping a poset P of base types is lifted to an order (subtype) relation on structured types. This is done via a subtype logic $\vdash_P$ which defines derivable judgements of the form $C \vdash_P \tau \leq \tau'$, where C is a finite set of subtype constraints between type expressions and τ, τ' are possibly structured types. The simply typed lambda calculus is then extended with a rule of *subsumption*, which allows any term having type τ under the subtype assumptions C to have any supertype τ' of τ, i.e., any type τ' such that $C \vdash_P \tau \leq \tau'$. The presence of subtype assumptions C in typings of lambda terms is necessitated by the desire to have *principal typings*, see [7, 13].

However, in contrast to, *e.g.*, the simply typed lambda calculus and ML, subtyping systems typically suffer from the problem that the size of principal typings may get intractably large for natural programs of even moderate size, which makes typing information difficult to read and may seriously slow down type inference in polymorphic frameworks. The problem is now widely recognized and has generated a substantial amount of work that aims at *simplifying* typings generated by subtype inference algorithms (see, *e.g.*, [7, 4, 10, 17, 5, 9, 14, 20, 16, 1, 6].) Recently, many researchers have independently suggested that the problem should be attacked by introducing stronger systems, based on the more powerful model-theoretic notion of *entailment* ($\models_P$) rather than the syntactic proof relation $\vdash_P$. Such works include [14, 20, 6, 1]. The rationale is that a more powerful subsumption rule allows more typings to be considered equivalent in the system, and hence more succinct representations of principal typings can be found. Simplification algorithms exploiting this idea typically have to decide a predicate of the form $C \models_P \alpha \leq \beta$ in order to verify that a simplification step is sound. However, even though several such algorithms have been suggested (see references above), no study has so far addressed the problem of the inherent computational complexity of entailment. Indeed no nontrivial complexity results — neither lower bounds nor interesting upper bounds — have been given for subtype entailment. The present paper aims at filling this gap for simple types over lattices.

Whereas the complexity of subtype entailment is largely unstudied, the complexity of subtype *satisfiability* is rather well-understood by now (see [18, 19, 2, 3]). In particular, Tiuryn [18] has shown that the satisfiability problem for simple types [1] is PSPACE-hard in general but uniformly in PTIME if P is a lattice (see also [2, 3, 15] for various generalizations.) Since the *entailment* predicate $C \models_P \tau \leq \tau'$ (does every valuation satisfying C also satisfy $\tau \leq \tau'$?) is at least as hard to decide as satisfiability in any non-trivial poset, [2] the problem is certainly PSPACE-hard in general, and a naive algorithm will in fact be doubly exponential. [3] For two reasons, however, it is of particular interest to know the complexity of entailment over *lattices* of base types. Firstly, it is not clear that the problem is harder or just as easy as satisfiability; in particular, it is a *prima facie* possibility that it might be in PTIME. [4] Secondly, virtually all recent proposals for model-theoretic subtyping systems use lattices as models. Thus the lower bounds for subtype satisfiability are inapplicable. Hence, we restrict ourselves in this paper to consideration of predicates of the form $C \models_L \alpha \leq \beta$ where L is a lattice.

1.2 Main results

The main results of this paper are as follows:

1. We prove a characterization theorem for entailment with atomic types (type variables and type constants) over lattices, leading to a complete axiomatization of atomic entailment and a linear time algorithm for deciding $C \models_L \alpha \leq \beta$ where C is consistent.

[1]The satisfiability problem is: given a constraint set C of inequalities between simple types, determine whether there exists a valuation assigning elements of P to variables in C which makes all inequalities in C true in the ground ordering?

[2]I.e. if the poset P is not the singleton set, then there are two distinct elements b and b' in P such that $b \not\leq_P b'$ and the problem $C \models_P b \leq b'$ is then equivalent to the satisfiability problem for C

[3]The problem $C \models_P \alpha \leq \beta$ can be reduced to a problem with only *atomic* subtype assumptions (*i.e.*, with no constructed types) modulo an exponential expansion of the size of C; considering all valuations satisfying the resulting assumption set then leads to a doubly exponential procedure.

[4]Note that whereas logical systems with negation sign typically have entailment problems of equal complexity to that of the unsatisfiability problem (by standard reduction of the former to the latter via negation), the situation is different in the rather weak logics of subtyping under study here.

2. We prove that the general entailment problem for structural subtyping for simple types is uniformly coNP-complete over any non-trivial lattice. Proving membership in coNP is non-trivial and exploits the characterization given for the atomic case. The result shows in which precise sense the intuitive "exponential" explosion incurred by a naive reduction of the general case to the atomic case is inherent. The resulting coNP-algorithm contains ideas that appear to be useful for developing incomplete but efficient algorithms for deciding entailment in subtype systems of practical interest. The coNP-hardness result shows that entailment is strictly harder than satisfiability for subtype inequalities.

3. The entailment problem is coNP-complete in a system with only a single binary *covariant* type constructor. It is also coNP-complete for simple types with additional type constructors, including the contravariant function type constructor. This shows that the presence or absence of contravariance — in contrast to some other type inference problems — has *no impact* on the complexity of the problem. Moreover, our methods are robust with respect to other systems, including *nonstructural* type systems. For example, they can be used to show that the entailment problem is coNP-complete for simple types with a bottom type (less than any type) and a top type (larger than any type). It remains to be seen, however, whether they can shed insight into the problem of subtype entailment with regular recursive types.

For simplicity the present summary gives the development only for a structural system containing a single binary covariant constructor (pairing). The result for other systems as mentioned above can be obtained by extending the methods used here, and details are left out in this summary but may be included in the final paper.

The paper is structured as follows. In Section 2 we review basic results on subtyping. These are primarily used to establish the central characterization of entailment for atomic types (Theorem 3.2); this is done in Section 3. In Section 4 we generalize the characterization to entailment for simple types. This is used in our proof that entailment is in coNP in Section 4.2 and in the coNP-hardness proof of entailment in Section 4.3.

2 Preliminaries

Fix a lattice $(L, \leq_L)$ of base types. We assume a denumerable set $\mathcal{V}$ of type variables. The set of *type expressions over* L, denoted by $T_L(\mathcal{V})$, is ranged over by τ, defined by

$$\begin{array}{rcl} \tau & ::= & A \mid \tau * \tau' \\ A & ::= & \alpha \mid b \end{array}$$

where α ranges over $\mathcal{V}$ and b ranges over the constants in L. Constants from L and variables are referred to collectively as *atoms* and are ranged over by A. If $\tau \in T_L(\mathcal{V})$ and no variable occurs in τ, then τ is called a *ground type*. The set of ground types over L is denoted T_L. A *constraint set* C is a finite set of formal inequalities of the form $\tau \leq \tau'$ with $\tau, \tau' \in T_L(\mathcal{V})$. C is called *atomic* if all inequalities in C have the form $A \leq A'$. We let $\mathrm{Var}(C)$ and $\mathrm{Cnst}(C)$ denote, respectively, the set of type variables appearing in C and the set of constants appearing in C.

The subtype logic $\vdash_L$ is used in the definition of entailment below, and it is given in Figure 1. The subtype logic defines provable judgements of the form $C \vdash_L \tau \leq \tau'$, meaning that the inequality $\tau \leq \tau'$ is a provable consequence of the subtype assumptions in constraint set C.

The following definition explains the concepts we shall mainly be interested in.

$$[const] \quad C \vdash_L b \leq b' \text{ , provided } b \leq_L b'$$

$$[ref] \quad C \vdash_L \tau \leq \tau$$

$$[hyp] \quad C \cup \{\tau \leq \tau'\} \vdash_L \tau \leq \tau'$$

$$[trans] \quad \frac{C \vdash_L \tau \leq \tau' \quad C \vdash_L \tau' \leq \tau''}{C \vdash_L \tau \leq \tau''}$$

$$[arrow] \quad \frac{C \vdash_L \tau_1 \leq \tau_1' \quad C \vdash_L \tau_2 \leq \tau_2'}{C \vdash_L \tau_1 * \tau_2 \leq \tau_1' * \tau_2'}$$

Figure 1: Subtype logic

Definition 2.1 (Valuation, satisfaction, entailment, consistence)
A ground substitution $\rho : \mathcal{V} \to T_L$ is called a *valuation*. We say that ρ *satisfies* an inequality $\tau \leq \tau'$, written $\rho \models_L \tau \leq \tau'$, if and only if $\vdash_L \rho(\tau) \leq \rho(\tau')$. If C is a finite set of inequalities, then we say that ρ *satisfies* C, written $\rho \models_L C$, if and only if $\rho \models_L \tau \leq \tau'$ for all $\tau \leq \tau' \in C$. We say that C *entails* $\tau \leq \tau'$, written $C \models_L \tau \leq \tau'$, if and only if we have

$$\forall \rho : \mathcal{V} \to T_L.\ \rho \models_L C \Rightarrow \rho \models_L \tau \leq \tau'$$

We say that C is *cosistent*, if and only if we have

$$\forall b, b' \in L.\ C \vdash_L b \leq b' \Rightarrow b \leq_L b'$$

i.e., C is consistent just in case C has no provable consequences contradicting the ordering of L. □

In the remainder of this section we give some basic lemmas most of which are standard. A substitution S is a function mapping type variables in $\mathcal{V}$ to types in $T_L(\mathcal{V})$, and it is lifted homomorphically to types in the standard way, and it is extended to constraint sets by setting $S(C) = \{S(\tau) \leq S(\tau') \mid \tau \leq \tau' \in C\}$.

Lemma 2.2 *(Substitution Lemma)*

1. *If* $C \vdash_L \tau \leq \tau'$, *then* $S(C) \vdash_L S(\tau) \leq S(\tau')$.
2. *If* $C \models_L \tau \leq \tau'$, *then* $S(C) \models_L S(\tau) \leq S(\tau')$

PROOF As for 1, see [13]. As for 2, assume $C \models_L \tau \leq \tau'$ and suppose that $\rho \models_L S(C)$, i.e., $\rho \circ S \models C$, hence (by assumption) $\rho \circ S \models_L \tau \leq \tau'$, i.e., $\rho \models_L S(\tau) \leq S(\tau')$. □

Two types are *matching* if they have the same shape; in more detail, τ and τ' match if they are both atoms (possibly distinct) or else $\tau = \tau_1 * \tau_2$, $\tau' = \tau_1' * \tau_2'$ with τ_1 matching τ_1' and τ_2 matching τ_2'. A set C of inequalities is matching if it holds for all $\tau \leq \tau' \in C$ that τ matches τ'. A *matching substitution* for C is a substitution S such that $S(C)$ is matching. If there is a matching substitution S for C then there is a *most general* matching substitution (see [13] for details), denoted M_C, with the property that, whenever R is a matching substitution for C, then there exists a substitution V such that $R = V \circ M_C$.

Lemma 2.3 *(Match Lemma)*

1. *If C is atomic, and $C \vdash_L \tau \leq \tau'$, then τ and τ' match.*

2. *Assume C has a matching substitution. Then $C \models_L \tau \leq \tau'$ if and only if $M_C(C) \models_L M_C(\tau) \leq M_C(\tau')$.*

PROOF As for 1, see [13]. As for 2, the implication ($\Rightarrow$) follows from the Substitution Lemma. To see the implication ($\Leftarrow$), assume $M_C(C) \models_L M_C(\tau) \leq M_C(\tau')$ and suppose that $\rho \models_L C$. Then it follows from the Match Lemma that $\rho(C)$ is matching, so there exists a substitution V such that $\rho = V \circ M_C$, and so we have by $\rho \models_L C$ that $V \models_L M_C(C)$, and hence (by the assumption) we have $V \models_L M_C(\tau) \leq M_C(\tau')$, i.e., $V \circ M \models_L \tau \leq \tau'$, i.e., $\rho \models_L \tau \leq \tau'$. $\square$

Note that, if C is satisfiable, then C has a matching substitution (if ρ satisfies C, then $\emptyset \vdash_L \rho(C)$ and hence, by the first part of the Match Lemma, $\rho(C)$ must be matching.)

Lemma 2.4 *(Decomposition Lemma)*
If C is atomic, then $C \vdash_L \tau_1 * \tau_2 \leq \tau_1' * \tau_2'$ if and only if $C \vdash_L \{\tau_1 \leq \tau_1', \tau_2 \leq \tau_2'\}$.

PROOF See [13]. $\square$

For atomic constraint set C over L, define the sets

$$\begin{array}{rcl} \uparrow_C(\alpha) & = & \{b \in L \cap \mathrm{Cnst}(C) \mid C \vdash_L \alpha \leq b\} \\ \downarrow_C(\alpha) & = & \{b \in L \cap \mathrm{Cnst}(C) \mid C \vdash_L b \leq \alpha\} \end{array}$$

Let the operations $\vee$ and $\wedge$ denote the least upper bound and the greatest lower bound in L.

Lemma 2.5 *(Satisfiability)*
Let C be atomic. If C is consistent, then

1. *The valuation $\rho_\uparrow = \{\alpha \mapsto \bigwedge \uparrow_C(\alpha)\}_{\alpha \in Var(C)}$ is a solution to C*

2. *The valuation $\rho_\downarrow = \{\alpha \mapsto \bigvee \downarrow_C(\alpha)\}_{\alpha \in Var(C)}$ is a solution to C*

PROOF See [11, 18]. $\square$

3 Atomic entailment

In this section, we consider the predicate $C \models_L \alpha \leq \beta$ with C atomic, i.e., every inequality in C has the form $A \leq A'$. We aim at a complete axiomatization of entailment with atomic constraint sets. Note the entailment problem $C \models_L \tau \leq \tau'$ can be reduced, in linear time, to the problem $C' \models_L \alpha \leq \beta$, where α and β are fresh variables and with C' the atomic decomposition (by Lemma 2.4) of the set $C \cup \{\alpha = \tau, \beta = \tau'\}$. [5] For the purposes of the following development it is therefore sufficient to consider just entailments of the form $C \models_L \alpha \leq \beta$ with C atomic.

We begin with a technical lemma. Given atomic constraint set C, we can regard $C \cup L$ as a digraph: there is an edge from A to A' for every inequality $A \leq A'$ in C, and L defines a digraph by stipulating that there is an edge from b to b' whenever $b \leq_L b'$.

[5]Here $\tau_1 = \tau_2$ is a shorthand for the two inequalities $\tau_1 \leq \tau_2$ and $\tau_2 \leq \tau_2$

Lemma 3.1 *Assume C atomic, let $b \in L$, and let α and β be two distinct variables. Assume that*

(i) $b \not\in \downarrow_C(\beta)$ *and*

(ii) $C \not\vdash_L \alpha \leq \beta$

Then $\downarrow_C(\beta) = \downarrow_{C[b/\alpha]}(\beta)$

PROOF (Sketch) By Lemma 2.2 we have (since $\alpha \neq \beta$) that $C \vdash_L b' \leq \beta$ implies $C[b/\alpha] \vdash_L b' \leq \beta$, for any $b' \in L$, which shows that $\downarrow_C(\beta) \subseteq \downarrow_{C[b/\alpha]}(\beta)$.

To prove the inclusion $\downarrow_{C[b/\alpha]}(\beta) \subseteq \downarrow_C(\beta)$, we use that, whenever $b' \in \downarrow_{C[b/\alpha]}(\beta)$, there must exist a path $P_{b'}$ in $C[b/\alpha] \cup L$ (regarded as a digraph) witnessing this fact. The inclusion then follows from the property

(∗) For any $b' \in L$ and any path $P_{b'}$ in $C[b/\alpha] \cup L$ witnessing $b' \in \downarrow_{C[b/\alpha]}(\beta)$ there is a path P' in $C \cup L$ witnessing $b' \in \downarrow_C(\beta)$.

The property (∗) is proven by induction on the length of a path $P_{b'}$ in $C[b/\alpha] \cup L$ of shortest length witnessing $b' \in \downarrow_{C[b/\alpha]}(\beta)$. The induction proof involves a case analysis over the form of the path $P_{b'}$ which is tedious but not difficult, and we leave the details for the reader. □

If G is a graph, we let G^* denote the transitive closure of G. We can now prove the main result of this section:

Theorem 3.2 *Let C be atomic, consistent, α and β distinct variables in C. Then $C \models_L \alpha \leq \beta$ if and only if one of the following conditions holds:*

(i) either $\alpha \leq \beta \in (C \cup L)^*$

(ii) or $\bigwedge \uparrow_C(\alpha) \leq_L \bigvee \downarrow_C(\beta)$

PROOF (⇒). Assume $C \models_L \alpha \leq \beta$ and $\alpha \leq \beta \not\in (C \cup L)^*$. We proceed by contradiction, assuming

$$\bigwedge \uparrow_C(\alpha) \not\leq_L \bigvee \downarrow_C(\beta) \tag{1}$$

Since $C \models_L \alpha \leq \beta$, we have by substitutivity of $\models_L$ (Lemma 2.2) and $\alpha \neq \beta$ that

$$C[\bigwedge \uparrow_C(\alpha)/\alpha] \models_L \bigwedge \uparrow_C(\alpha) \leq \beta \tag{2}$$

Let $C^\dagger = C[\bigwedge \uparrow_C(\alpha)/\alpha]$. By the Satisifiability Lemma (Lemma 2.5) we know, since C is consistent, that the map

$$\{\alpha \mapsto \bigwedge \uparrow_C(\alpha)\}$$

can be extended to a satisfying valuation for C. It follows that $C^\dagger$ is consistent. Then, by the Satisfiability Lemma again, we get that the valuation

$$\{\gamma \mapsto \bigvee \downarrow_{C^\dagger}(\gamma)\}_{\gamma \in \mathrm{Var}(C^\dagger)}$$

satisfies $C^\dagger$. By (2) we then have

$$\bigwedge \uparrow_C(\alpha) \leq_L \bigvee \downarrow_{C^\dagger}(\beta) \tag{3}$$

If $C \models_L \bigwedge \uparrow_C(\alpha) \leq \beta$, then the Satisfiability Lemma entails $\bigwedge \uparrow_C(\alpha) \leq_L \bigvee \downarrow_C(\beta)$ via the satisfying map $\{\gamma \mapsto \bigvee \downarrow_C(\gamma)\}_{\gamma \in \mathrm{Var}(C)}$, contradicting (1). We must therefore conclude that $\bigwedge \uparrow_C(\alpha) \leq \beta \notin (C \cup L)^*$. Hence, $\bigwedge \uparrow_C(\alpha) \notin \downarrow_C(\beta)$. This together with the assumption $\alpha \leq \beta \notin (C \cup L)^*$ allows us to apply Lemma 3.1, which shows that $\downarrow_C(\beta) = \downarrow_{C^\dagger}(\beta)$, and so by (3) we have

$$\bigwedge \uparrow_C(\alpha) \leq_L \bigvee \downarrow_C(\beta) \tag{4}$$

But (4) contradicts (1), thereby proving the implication ($\Rightarrow$).

($\Leftarrow$). If (i) is the case, the result follows immediately, and if (ii) is the case, the result follows because we evidently have

$$C \models_L \alpha \leq \bigwedge \uparrow_C(\alpha) \text{ and } C \models_L \bigvee \downarrow_C(\beta) \leq \beta$$

□

Theorem 3.2 shows that we get a sound and complete axiomatization of the relation $\models_L$ on *atomic* constraint sets by adding the following rules to $\vdash_L$:

$$\frac{C \vdash_L A \leq b_1 \quad C \vdash_L A \leq b_2}{C \vdash_L A \leq b_1 \wedge_L b_2} \qquad \frac{C \vdash_L b_1 \leq A \quad C \vdash_L b_2 \leq A}{C \vdash_L b_1 \vee_L b_2 \leq A}$$

$$\frac{C \vdash_L b_1 \leq b_2 \quad b_1 \not\leq_L b_2}{C \vdash_L \tau \leq \tau'}$$

The theorem also shows that the predicate $C \models_L \alpha \leq \beta$, where C is atomic, can be decided very efficiently:

Corollary 3.3 *Assuming $\vee_L, \wedge_L$ and $\leq_L$ are constant-time operations, given C of size n (number of symbols in C) it can be decided in linear time whether $C \models \alpha \leq \beta$.*

PROOF Note that C, viewed as a directed graph, has $O(n)$ vertices and $O(n)$ edges. Criterion (i) can be decided in time $O(n)$ by computing the set of of nodes reachable from α in C. As for criterion (ii), both $\bigwedge \uparrow_C(\alpha)$ and $\bigvee \downarrow_C(\beta)$ can be computed in time $O(n)$, again by computing the set of nodes reachable from α in C and by computing the set of nodes reachable from β along the reverse edges in C. □

4 Entailment for simple types

In this section we consider the general entailment problem $C \models_L \alpha \leq \beta$ where C is a set of arbitrary inequalities between simple types. We first prove membership in coNP, and then we prove hardness.

We assume w.l.o.g. that C is *standardized* in such a way that every inequality in C has one of the forms $A \leq A_1 * A_2$, $A_1 * A_2 \leq A$ or $A \leq A'$. Every constraint set can be transformed into this form, modulo a blow-up in the size of the set which is at most linear, by exchanging inequalities of the form $\tau_1 * \tau_2 \leq \tau$ with the inequalities $\alpha = \tau_1, \beta = \tau_2, \gamma = \tau, \alpha * \beta \leq \gamma$ (α, β, γ fresh) and similarly for inequalities of the form $\tau \leq \tau_1 * \tau_2$.

4.1 Graph representation

To prove that the problem $C \models_L \alpha \leq \beta$ is in coNP, it is sufficient to provide an NP-algorithm to solve the problem $C \not\models_L \alpha \leq \beta$. This will be done by generalizing Theorem 3.2 to arbitrary constraint sets (in Section 4.2 below), and it is the main goal of the present subsection to prepare for this.

In order to generalize Theorem 3.2, we shall represent the set C using a data structure called the *graph* of C, denoted $\mathcal{G}_C$. This is a labelled graph whose nodes are elements of L, $\mathcal{V}$ and *occurrences* of $*$ in C and whose edges are denoted $\leadsto_\ell$ with label ℓ being drawn from the set $\mathcal{L} = \{F^+, F^-, S^+, S^-\}$ or $\ell = \Lambda$. We consider $\mathcal{L}$ as an alphabet, and we let $\mathcal{L}^*$ denote the set of finite strings over $\mathcal{L}$, where Λ denotes the empty string.

If $A \leq A_1 * A_2$ is an inequality in C, the notation $A \leadsto_\ell A_1 * A_2$ denotes an edge labelled ℓ from A to the particular occurrence of $*$ in that inequality (similarly for the notation $A_1 * A_2 \leadsto_\ell A$.) The relations $\leadsto_\ell$ for $\ell \in \{F^+, S^+, F^-, S^-\}$ represent the terms occurring in C. For example, the relation $A_1 \leadsto_{F^-} A_1 * A_2$ signifies that, in the syntax tree for $A_1 * A_2$, there is an edge leading from A_1 to $*$, and $A_1 * A_2 \leadsto_{F^+} A_1$ signifies that, in the syntax tree for $A_1 * A_2$, there is an edge leading from $*$ to A_1 (in the syntax tree, these are not different edges, but here we wish to distinguish between the directions.) The special relation $\leadsto_\Lambda$ captures the inequalities in C; this relation is sometimes referred to as the *flow* in C.

The graph $\mathcal{G}_C$ is defined to be the least graph satisfying the following conditions:

1. Whenever $A \leq \tau \in C$, we have $A \leadsto_\Lambda \tau$ in $\mathcal{G}_C$

2. Whenever $\tau \leq A \in C$, we have $\tau \leadsto_\Lambda A$ in $\mathcal{G}_C$

3. For every occurrence of $A_1 * A_2 \in C$ we have $A_1 \leadsto_{F^-} A_1 * A_2$, $A_2 \leadsto_{S^-} A_1 * A_2$, $A_1 * A_2 \leadsto_{F^+} A_1$ and $A_1 * A_2 \leadsto_{S^+} A_2$ in $\mathcal{G}_C$.

4. The relation $\leadsto_\Lambda$ is transitively closed in $\mathcal{G}_C$

5. $\mathcal{G}_C$ is closed under *induced flow*, i.e., whenever $A_1 * A_2 \leadsto_\Lambda A_3 * A_4$ in $\mathcal{G}_C$, we also have $A_1 \leadsto_\Lambda A_3$ and $A_2 \leadsto_\Lambda A_4$ in $\mathcal{G}_C$.

We can assume that $\mathcal{G}_C$ is acyclic, since if C is satisfiable in simple types, only cycles among atomic types can occur, and such cycles can be eliminated (see, e.g., [7].) $\mathcal{G}_C$ can therefore be regarded as a *dag*-representation of C. Note that $\mathcal{G}_C$ has maximal sharing of atoms, since, apart from the distinct occurrences of $*$, the nodes of $\mathcal{G}_C$ are defined to be the *set* $L \cup \mathcal{V}$. Hence, every variable and every constant occurring in C occurs exactly once in $\mathcal{G}_C$.

Example 4.1 Consider the constraint set

$$C = \{\alpha * \beta \leq \gamma, \gamma \leq \beta * \delta\}$$

Then $\mathcal{G}_C$ is as shown below, where dotted arrows indicate the relations $\leadsto_\ell$ for $\ell \in \mathcal{L}$, and full arrows labelled with Λ are flow-arrows; the labels from $\mathcal{L}$ have been left out for readability.

$$\begin{tikzcd}
& * \arrow[r, "\Lambda"] \arrow[dl, dotted, bend right] \arrow[dr, dotted, bend left] & \gamma \arrow[r, "\Lambda"] & * \arrow[dl, dotted, bend right] \arrow[dr, dotted, bend left] & \\
\alpha \arrow[ur, dotted, bend left] \arrow[rr, "\Lambda"] & & \beta \arrow[ul, dotted, bend left] \arrow[ur, dotted, bend left] \arrow[rr, "\Lambda"] & & \delta \arrow[ul, dotted, bend right]
\end{tikzcd}$$

□

We now introduce some notation and concepts for talking about various forms of paths in $\mathcal{G}_C$. Let π be a string in $\{F, S\}^*$. Then π can be regarded as a function from type expressions to type expressions, as follows:

- $\mathrm{dom}(\Lambda) = T_L(\mathcal{V})$ and $\Lambda(\tau) = \tau$
- $\mathrm{dom}(\pi F) = \{\tau_1 * \tau_2 \mid \tau_1 \in \mathrm{dom}(\pi)\}$ and $\pi F(\tau_1 * \tau_2) = \pi(\tau_1)$
- $\mathrm{dom}(\pi S) = \{\tau_1 * \tau_2 \mid \tau_2 \in \mathrm{dom}(\pi)\}$ and $\pi S(\tau_1 * \tau_2) = \pi(\tau_2)$

We are mainly interested in satisfiable constraint sets, since the entailment problem for unsatisfiable sets is trivial. Recall that, if C is satisfiable, then C has a matching substitution, and so M_C is defined. We say that π is a *valid path* for $\alpha \in \mathrm{Var}(C)$ if $M_C(\alpha) \in \mathrm{dom}(\pi)$, and such a path π is called a *complete path* for α if π (regarded as a string) is not a proper prefix of any valid path for α. The intuition here is that the set of valid paths for α completely describes the common structure of all types in the set $\{\rho(\alpha) \mid \rho \models_L C\}$; hence, if π is valid for α, we know that π (regarded as a function) is a well-defined operation on the image of α under any satisfying valuation. Notice that, if $C \models_L \alpha \leq \beta$ with C satisfiable, then π is valid for α if and only if π is valid for β. Ultimately, we shall be interested in checking efficiently whether a given path is valid for a α in C; we shall do so by inspecting $\mathcal{G}_C$, *without* actually expanding C under the substitution M_C which could be an exponential time process in the worst case.

We now introduce a few more technical notions, which will be needed to generalize Theorem 3.2 in Section 4.2 below. Let σ be a string in $\mathcal{L}^*$ and π a path in $\{F, S\}^*$. Then σ is called a *valid π-sequence* if there is a prefix π_1 of π such that $\sigma = (\pi_1)^+(\pi_1)^-$, where $(\pi_1)^+$ is the result of exchanging F with F^+ and S with S^+ in π_1, and similarly for $(\pi_1)^-$. The idea here is that, if π is a valid path in C, then this fact will be witnessed by π-valid sequences in $\mathcal{G}_C$.

Let σ be a string in $\mathcal{L}^*$; a *σ-transition* from τ to τ' in $\mathcal{G}_C$ is a series of relations holding in $\mathcal{G}_C$ of the form

$$\tau \rightsquigarrow_{\ell_1} \tau_1 \rightsquigarrow_{\ell_2} \ldots \rightsquigarrow_{\ell_{n-1}} \tau_{n-1} \rightsquigarrow_{\ell_n} \tau'$$

such that $\sigma = \ell_1 \ell_2 \ldots \ell_{n-1} \ell_n$. Here equality is the string-equality of the monoid $\mathcal{L}^*$, so, in particular, σ may arise from a series of relations which contains any number of relations of the form $\tau_i \rightsquigarrow_\Lambda \tau_{i+1}$.

For two variables $\alpha, \beta \in \mathrm{Var}(C)$ and a path π valid for α in C, we write $\alpha \preceq_C^\pi \beta$ if there is a valid π-sequence σ for α with a σ-transition leading from α to β in $\mathcal{G}_C$.

Finally, for π a valid path for α in C, define the sets

$$\uparrow_{C,\pi}(\alpha) = \{b \in L \mid \text{there is a } (\pi)^+\text{-transition from } \alpha \text{ to } b \text{ in } \mathcal{G}_C\}$$

$$\downarrow_{C,\pi}(\alpha) = \{b \in L \mid \text{there is a } (\pi)^-\text{-transition from } b \text{ to } \alpha \text{ in } \mathcal{G}_C\}$$

4.2 Membership in coNP

We begin with a technical lemma. The main point of it is to lift Theorem 3.2 to non-atomic constraint sets. The intuition here is that, since (Lemma 2.3) $C \models_L \alpha \leq \beta$ holds if and only if $M_C(C) \models_L M_C(\alpha) \leq M_C(\beta)$, it is sufficient to inspect reachability properties in the decomposition of $M_C(C)$ (by Match Lemma, Decomposition Lemma and Theorem 3.2) when considering entailment with C. However, all such properties in the decomposition of $M_C(C)$ must clearly be witnessed already in $M_C(C)$, and all such witnesses, in turn, must be forced by valid paths and transitions that are already present in $\mathcal{G}_C$.

Lemma 4.2 *Let C be satisfiable. Then $C \models_L \alpha \leq \beta$ if and only if for every complete path π valid for α and β in $\mathcal{G}_C$ one of the following conditions holds:*

(i) either $\alpha \preceq^{\pi}_{C} \beta$

(ii) or $\bigwedge \uparrow_{C,\pi} (\alpha) \leq_L \bigvee \downarrow_{C,\pi} (\beta)$

PROOF The implication ($\Leftarrow$) is easy and left out.

($\Rightarrow$). For $\alpha \in \mathrm{Var}(C)$ we define the *size* of α in C, denoted $|\alpha|_C$, to be the number of occurrences of atoms in $M_C(\alpha)$. Note that, since C is satisfiable, it follows from the Match Lemma that $|\alpha|_C = |\beta|_C$, whenever $C \models_L \alpha \leq \beta$. We prove the result by induction on $|\alpha|_C = |\beta|_C$.

For the base case ($|\alpha|_C = 1$), it follows from the fact that $\mathcal{G}_C$ is closed under induced flow that we have $C \models \alpha \leq \beta$ if and only if $C' \models_L \alpha \leq \beta$ when we take C' to be the *atomic* set

$$C' = \{A \leq A' \mid A \leadsto_\Lambda A' \text{ in } \mathcal{G}_C\}$$

The result now follows from Theorem 3.2.

For the inductive case ($d_C(\alpha) = d_C(\beta) > 1$) let $\sim$ denote the reflexive, symmetric, transitive closure of $\leadsto_\Lambda$ in $\mathcal{G}_C$. Let $\alpha_1, \alpha_2, \beta_1, \beta_2$ be fresh variables and let $C' = C[\alpha_1 * \alpha_2/\alpha, \beta_1 * \beta_2/\beta]$; finally, let $C^\dagger$ be a standardized version of C'. Since $|\alpha|_C = |\beta|_C > 1$, we have (by Match Lemma and Decomposition Lemma) that $C \models_L \alpha \leq \beta$ if and only if $C^\dagger \models_L \alpha_1 \leq \beta_1$ and $C^\dagger \models_L \alpha_2 \leq \beta_2$. Since we clearly have $d_{C^\dagger}(\alpha_i) < d_C(\alpha)$, it follows by induction hypothesis that either (i) or (ii) holds for some path π_1 valid for α_1 and β_1 in $\mathcal{G}_{C^\dagger}$ and some path π_2 valid for α_2 and β_2 in $\mathcal{G}_{C^\dagger}$. Now, for every inequality $\alpha \leq A_1 * A_2$ in C, we have $\alpha \leadsto_\Lambda A_1 * A_2 \leadsto_{F^+} A_1$ and $\alpha \leadsto_\Lambda A_1 * A_2 \leadsto_{S^+} A_2$. It follows by closure of $\mathcal{G}_C$ under induced flow that, for every σ-transition from α_1 to β_1 with σ a valid π-sequence for α_1 in $\mathcal{G}_{C^\dagger}$, there is a σ'-transition from α to β in $\mathcal{G}_C$ with σ' a valid πF-sequence for α. Similary, for any transition from α_2 to β_2, we find in $\mathcal{G}_C$ a valid πS- sequence. By the same reasoning, it can be seen that $\uparrow_{C^\dagger,\pi_i} (\alpha_i) = \uparrow_{C,\pi\ell_i} (\alpha)$ ($i = 1, 2$, $\ell_1 = F^+$, $\ell_2 = S^+$) and $\downarrow_{C^\dagger,\pi_i} (\beta_i) = \downarrow_{C,\pi\ell_i} (\beta)$ ($i = 1, 2$, $\ell_1 = F^-$, $\ell_2 = S^-$.) This entails that, in any case, (i) or (ii) must already hold for some path π valid for α and β in $\mathcal{G}_C$. □

We now show that Lemma 4.2 leads to a coNP-procedure for deciding the predicate $C \models_L \alpha \leq \beta$. The basic insight is that the difficulty of the complementary problem $C \not\models_L \alpha \leq \beta$ is concentrated in finding a valid path serving as a witness that neither one of the properties mentioned in Lemma 4.2 holds. This can be done by non-deterministically guessing such a witness.

Theorem 4.3 *The predicate $C \models_L \alpha \leq \beta$ is in* **coNP**.

PROOF It is sufficient to show that the predicate $C \not\models_L \alpha \leq \beta$ is in **NP**.

Given C, α and β we first check if C is satisfiable; this can be done deterministically in polynomial time, by the result of [18]. If C is not satisfiable, we return *true*, otherwise we proceed as follows, under the assumption that C is satisfiable.

We non-deterministically guess a path π, of length at most linear in the size of C, which will serve as a succinct witness for the property

$$(*) \quad \begin{cases} (i) & \alpha \not\preceq^{\pi}_{C} \beta, \text{ and} \\ (ii) & \bigwedge \uparrow_{C,\pi} (\alpha) \not\leq_L \bigvee \downarrow_{C,\pi} (\beta) \end{cases}$$

By Lemma 4.2, it is sufficient if such a witness can be found and verified non-deterministically in polynomial time.

In order to verify, deterministically in polynomial time, that π is a witness of $(*)$, we first build the graph $\mathcal{G}_C$. This can be done deterministically in polynomial time, since it is reducible to a transitive closure problem. Only paths of length not greater than a maximal complete path for α and β need to be considered, and the size of such a path can be at most linear in the size of C; this follows from the obvious fact that, whenever $\pi = \ell_1 \ldots \ell_n$ is a valid path for γ in C, there must be a distinct inequality in C for each n which forces the depth of γ. It follows from this observation that we can check deterministically in polynomial time whether a given path π is in fact *valid* for α and β: we compute the reflexive, symmetric, transitive closure of $\rightsquigarrow_\Lambda$ (denoted $\sim$) and consider the equivalence class $[\alpha]$ with respect to $\sim$, where we check to see that α is compared to a pair. If ℓ_1 is F, we look for a pair in $[\alpha]$ forcing ℓ_1 to be valid for α and we collect all first components of pairs in $[\alpha]$ (in case $\ell_1 = S$, we collect all second components, of course.) We proceed, for each element in this collection, to collect new equivalence classes with respect to $\sim$ and inspect their union to check that ℓ_2 is forced; this process continues until we have reached ℓ_n. Clearly, the process can be completed in polynomial time, since distinct nodes with label $*$ are inspected at most once, each being in just one equivalence class processed by the above search, by satisfiability of C.

Having checked that π is valid for α and β, we now proceed to check that π is indeed a witness for $(*)$. This is reducible to collecting all the nodes in $\mathcal{G}_C$ reachable by π-valid transitions and all constant nodes reachable by the complete paths $(\pi)^+$ and $(\pi)^-$.

Part *(i)* of $(*)$ is tantamount to checking whether there exists a π-valid transition from α to β. This can be checked in polynomial time by computing the set $\pi^+\{\alpha\}$ and $\pi^-\{\beta\}$. Here $\pi^+\{\alpha\}$ is the set of nodes reachable from α in $\mathcal{G}_C$ by following F^+ and S^+ labeled edges as dictated by π and Λ-labeled edges (flow edges), and $\pi^-\{\beta\}$ is similar to $\pi^+\{\beta\}$ with the difference, however, that Λ-labeled nodes must be traversed in the *reverse* direction. Both sets can be computed in polynomial time. Now, $\alpha \not\leq_C^\pi \beta$ if and only if $\pi^+\{\alpha\} \cap \pi^-\{\beta\} = \emptyset$.

Part *(ii)* is also checked by using the sets $\pi^+\{\alpha\}$ and $\pi^-\{\beta\}$. The condition

$$\bigwedge \uparrow_{C,\pi}(\alpha) \not\leq_L \bigvee \downarrow_{C,\pi}(\beta)$$

is equivalent to checking $\bigwedge(\pi^+\{\alpha\} \cap L) \not\leq_L \bigvee(\pi^-\{\beta\} \cap L)$ since $(\pi^+\{\alpha\} \cap L = \uparrow_{C,\pi}(\alpha)$ and $\pi^-\{\beta\} \cap L = \downarrow_{C,\pi}(\beta)$. □

4.3 coNP-hardness

In this section we show that $C \models_L \alpha \leq \beta$ is coNP-hard. A basic idea in the proof presented below is that, by exploiting Theorem 3.2, we can use valid paths to encode logical truth valuations.

Theorem 4.4 *The predicate $C \models_L \alpha \leq \beta$ is hard for* **coNP** *under log-space reductions.*

PROOF Let NENT be the problem of deciding whether $C \not\models_L \alpha \leq \beta$. Since we assume α and β to be fixed an instance of NENT is given by C.

We reduce SAT (propositional satisfiability, [8]) to NENT. This shows that NENT is NP-hard, which in turn shows that the problem of deciding $C \models_L \alpha \leq \beta$ is coNP-hard.

The basic idea is that a path π defines a truth assignment of an instance of SAT. Given n variables $x_1, \ldots, x_n$ containing all the propositional variables occurring in an instance of SAT, we say that a path $\pi = l_1 \ldots l_n \in \{F, S\}^n$ of length n defines a truth assignment T_π as follows: $T(x_i) = \text{true}$ if $l_i = F$ and $T(x_i) = \text{false}$ if $l_i = S$ (note that F means *true* and not false.) Henceforth we shall think of paths both as paths and as the truth assignments they induce in this fashion.

Let us assume now that we are given an instance I of SAT, a set of clauses $\{C_1, \ldots, C_m\}$ over $x_1, \ldots, x_n$. Each clause is a finite disjunction of *atoms* A. An atom is a propositional variable or its negation. Recall that I is satisfiable, if there exists a truth assignment that makes every clause true.

Given I we construct an instance $C(I)$ of NENT (that is, a constraint set) as follows. For every clause $C_i = A_1 \vee \ldots \vee A_k$ in I we build a set of constraints that *excludes* every path that *falsifies* the clause. By excluding we mean that every such path π satisfies the inequality $\bigwedge \uparrow_{C(I),\pi}(\alpha) \leq_L \bigvee \downarrow_{C(I),\pi}(\beta)$. This is done by making sure that $\uparrow_{C(I),\pi}(\alpha)$ contains $\perp$ (the bottom element in L) and $\downarrow_{C(I),\pi}(\beta)$ contains $\top$ (the top element in L). Furthermore our construction is such that $\alpha \not\leq^{\pi}_{C(I)} \beta$ holds for *all* complete paths π in $C(I)$ and their prefixes. This means that existence or nonexistence of a path π such that

$$\bigwedge \uparrow_{C(I),\pi}(\alpha) \not\leq_L \bigvee \downarrow_{C(I),\pi}(\beta)$$

determines whether $C(I) \not\models_L \alpha \leq \beta$ or $C(I) \models_L \alpha \leq \beta$.

The set of paths that falsify clause C_i can be described by a unique *path pattern* $P_i \in \{F, S, \#\}^n$. A path pattern defines the set of paths that arise by replacing $\#$ arbitrarily by either F or S. For example, $S\#\#FS\#\#\#$ is the path pattern that falsifies clause $x_1 \vee \neg x_4 \vee x_5$ for $n = 8$.

Given a path pattern P we define $\mathcal{C}(P, \gamma)$ as follows:

$$\begin{array}{rcll}
\mathcal{C}(FP', \gamma) & = & \{\delta * \delta' \leq \gamma\} \cup \mathcal{C}(P', \delta) & (\delta, \delta' \text{ new}) \\
\mathcal{C}(SP', \gamma) & = & \{\delta * \delta' \leq \gamma\} \cup \mathcal{C}(P', \delta') & (\delta, \delta' \text{ new}) \\
\mathcal{C}(\#P', \gamma) & = & \{\delta * \delta \leq \gamma\} \cup \mathcal{C}(P', \delta) & (\delta \text{ new}) \\
\mathcal{C}(\Lambda, \gamma) & = & \{\top \leq \gamma\} &
\end{array}$$

Similarly, we define $\mathcal{C}^-(P, \gamma)$: This is done by *reversing* the inequalities in the first three clauses for $\mathcal{C}$ above and replacing the last clause by $\mathcal{C}^-(\Lambda, \gamma) = \{\gamma \leq \perp\}$.

Let P_i be the path pattern that falsifies clause C_i. The constraints generated for C_i are $\mathcal{C}(P_i, \beta) \cup \mathcal{C}^-(P_i, \alpha)$. The constraints $C(I)$ generated for I is the union of the constraints generated for the individual clauses in I.

Now, to see that $C(I)$ is a reduction of SAT to NENT we have to check:

1. If I is satisfiable then $C(I) \not\models_L \alpha \leq \beta$, and

2. if $C(I) \not\models_L \alpha \leq \beta$ then I is satisfiable.

Let us consider 1 then. Assume I is satisfiable. Then there is a truth assignment corresponding to a path π that makes all the clauses in I true. We show that π satisfies the property

$$(*) \quad \left\{ \begin{array}{ll} (i) & \alpha \not\leq^{\pi}_{C} \beta, \text{ and} \\ (ii) & \bigwedge \uparrow_{C,\pi}(\alpha) \not\leq_L \bigvee \downarrow_{C,\pi}(\beta) \end{array} \right.$$

As for part (i) we have already noted that this holds for all complete paths, including π. As for part (ii), we know that π does *not* match any of the path patterns that falsify a clause in I. By construction of $C(I)$ it follows that $\uparrow_{C,\pi}(\alpha) = \emptyset$ and $\downarrow_{C,\pi}(\beta) = \emptyset$. Thus $\bigwedge \uparrow_{C,\pi}(\alpha) = \top \not\leq_L \perp = \bigvee \downarrow_{C,\pi}(\beta)$ and, invoking Lemma 4.2, we are done.

Let us consider 2 now. Assume $C(I) \not\models \alpha \leq \beta$. By Lemma 4.2 and by construction of $C(I)$ this implies that there is a complete path π such that

$$\bigwedge \uparrow_{C(I),\pi}(\alpha) \not\leq_L \bigvee \downarrow_{C(I),\pi}(\beta).$$

We show that π is a satisfying truth assignment of I. Consider any clause C_i of I and the unique path pattern P_i that falsifies C_i. Assuming π matches a prefix of P_i implies that π has length n (since π is complete) and $\bigwedge \uparrow_{C(I),\pi} (\alpha) = \bot \leq_L \top = \bigvee \downarrow_{C(I),\pi} (\beta)$, but this contradicts the assumption that $\bigwedge \uparrow_{C(I),\pi} (\alpha) \not\leq_L \bigvee \downarrow_{C(I),\pi} (\beta)$. Consequently π does *not* match P_i and thus no extension of π to length n falsifies C_i; i.e. π (arbitrarily extended to length n) makes C_i true. Since it does so for all clauses C_i in I this shows that π satisfies I. $\square$

References

[1] A. Aiken, E.L. Wimmers, and J. Palsberg. Optimal representations of polymorphic types with subtyping. Technical Report UCB/CSD-96-909, University of California, Berkeley, July 1996.

[2] Marcin Benke. Efficient type reconstruction in the presence of inheritance. In *Mathematical Foundations of Computer Science (MFCS)*, pages 272–280. Springer Verlag, LNCS 711, 1993.

[3] Marcin Benke. Some complexity bounds for subtype inequalities. Technical Report TR 95-20 (220), Warsaw University, Institute of Informatics, Warsaw University, Poland, December 1995.

[4] P. Curtis. Constrained quantification in polymorphic type analysis. Technical Report CSL-90-1, Xerox Parc, February 1990.

[5] J. Eifrig, S. Smith, and V. Trifonov. Sound polymorphic type inference for objects. In *Proceedings OOPSLA '95*, 1995.

[6] M. Fähndrich and A. Aiken. Making set-constraint program analyses scale. In *Workshop on Set Constraints, Cambridge MA*, 1996.

[7] Y. Fuh and P. Mishra. Polymorphic subtype inference: Closing the theory-practice gap. In *Proc. Int'l J't Conf. on Theory and Practice of Software Development*, pages 167–183, Barcelona, Spain, March 1989. Springer-Verlag.

[8] M. Garey and D. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[9] M. Hoang and J.C. Mitchell. Lower bounds on type inference with subtypes. In *Proc. 22nd Annual ACM Symposium on Principles of Programming Languages (POPL)*, pages 176–185. ACM Press, 1995.

[10] S. Kaes. Type inference in the presence of overloading, subtyping and recursive types. In *Proc. ACM Conf. on LISP and Functional Programming (LFP), San Francisco, California*, pages 193–204. ACM Press, June 1992. also in LISP Pointers, Vol. V, Number 1, January-March 1992.

[11] P. Lincoln and J.C. Mitchell. Algorithmic aspects of type inference with subtypes. In *Proc. 19th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programmin Languages (POPL), Albuquerque, New Mexico*, pages 293–304. ACM Press, January 1992.

[12] J.C. Mitchell. Coercion and type inference (summary). In *Proc. 11th ACM Symp. on Principles of Programming Languages (POPL)*, pages 175–185, 1984.

[13] J.C. Mitchell. Type inference with simple subtypes. *Journal of Functional Programming*, 1(3):245–285, July 1991.

[14] F. Pottier. Simplifying subtyping constraints. In *Proceedings ICFP '96, International Conference on Functional Programming*, pages 122–133. ACM Press, May 1996.

[15] V. Pratt and J. Tiuryn. Satisfiability of inequalities in a poset. *Studia Logica (to appear)*.

[16] Jakob Rehof. Minimal typings in atomic subtyping. To appear in proceedings POPL '97, 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Paris, France, January 1997.

[17] G. S. Smith. Principal type schemes for functional programs with overloading and subtyping. *Science of Computer Programming*, 23:197–226, 1994.

[18] J. Tiuryn. Subtype inequalities. In *Proc. 7th Annual IEEE Symp. on Logic in Computer Science (LICS), Santa Cruz, California*, pages 308–315. IEEE Computer Society Press, June 1992.

[19] Jerzy Tiuryn and Mitchell Wand. Type reconstruction with recursive types and atomic subtyping. In M.-C. Gaudel and J.-P. Jouannaud, editors, *Proc. Theory and Practice of Software Development (TAPSOFT), Orsay, France*, volume 668 of *Lecture Notes in Computer Science*, pages 686–701. Springer-Verlag, April 1993.

[20] V. Trifonov and S. Smith. Subtyping constrained types. In *Proceedings SAS '96, Static Analysis Symposium, Aachen, Germany*, pages 349–365. Springer, 1996. Lecture Notes in Computer Science, vol.1145.