Constraint Automata and the Complexity of Recursive Subtype Entailment

Fritz Henglein and Jakob Rehof

DIKU, Department of Computer Science
Universitetsparken 1, DK-2100 Copenhagen Ø, Denmark
Electronic mail: {henglein,rehof}@diku.dk
Fax: +45 35321401

Abstract. We study entailment of structural and nonstructural recursive subtyping constraints. Constraints are formal inequalities between type expressions, interpreted over an ordered set of possibly infinite labeled trees. The nonstructural ordering on trees is the one introduced by Amadio and Cardelli for subtyping with recursive types. The structural ordering compares only trees with common shape. A constraint set *entails* an inequality if every assignment of meanings (trees) to type expressions that satisfies all the constraints also satisfies the inequality.
In this paper we prove that nonstructural subtype entailment is PSPACE-hard, both for finite trees (simple types) and infinite trees (recursive types). For the structural ordering we prove that subtype entailment over infinite trees is PSPACE-complete, when the order on trees is generated from a lattice of type constants.
Since structural subtype entailment over finite trees has been shown to be coNP-complete these are the first complexity-theoretic separation results that show that, informally, nonstructural subtype entailment is harder than structural entailment, and recursive entailment is harder than nonrecursive entailment.

1 Introduction

1.1 Constraint entailment

Notions of *entailment* for subtype inequality constraints and set inclusion constraints are currently receiving much attention. Recent work in subtyping systems and in set constraints using notions of constraint entailment in the sense of the present paper includes [1, 5, 15, 22, 6, 2, 8, 13]. In subtyping systems, a program is typed under a set of *suptyping constraints* consisting of inequalities of the form $\tau \leq \tau'$, where τ and τ' are type expressions; the constraints express hypotheses about subtype relations which must hold between the types for the program to be well-typed. If C is a constraint set and ϕ is an inequality (such as a subtype inequality or a set inclusion), we say that C *entails* ϕ, written $C \models \phi$, if every assignment of meanings to expressions that satisfies all the constraints in C also satisfies ϕ; an inequality is satisfied under the assignment, if it is true

in the intended model. The intended model for subtyping constraints consists of ordered labeled trees.

Constraint entailment and algorithms for deciding entailment have many important applications. Until now, perhaps the most important practical rôle of entailment has been to support, justify and reason about *constraint simplification*. Simplification removes redundant information, thereby making constraint sets more compact, more readable and algorithmically more manageable. If simplification is automated, then the simplification algorithm typically has to decide entailment problems in order to justify (or reject) a potential simplification step. It is therefore interesting to develop entailment algorithms and to know the computational complexity of deciding entailment. Works using entailment based simplification of set constraints include [1, 6, 2, 8]. In subtyping systems, quite similar problems are currently attacked using entailment for subtyping constraints; recent work includes [15, 22, 13], see also [17] for further references to previous work in subtype simplification.

1.2 Contribution and related work

The main objective of this paper is to understand the computational complexity of deciding the *entailment problem* for recursively constrained subtyping, both in the structural and the nonstructural version:

- Given a set of subtyping constraints C and type variables α, β, does $C \models \alpha \leq \beta$ hold in the set of ordered trees?

Here, C is a set of subtype inequalities between simple type expressions, where recursive type equations such as $\alpha = \alpha \to \beta$ have a solution.

The problem of finding a complete algorithm to decide recursive subtype entailment over nonstructurally ordered trees [3] remains open. However, no nontrivial lower bounds on the problem have been given up until now. Structural subtyping was introduced by Mitchell [14] and has been used in many subsequent subtyping systems, both with finite and recursive types. We study structural subtyping over a *lattice* of base types (*i.e.*, type constants such as, *e.g.*, `int`, `real`), because it is known [20] that even the satisfiability problem is PSPACE-hard for many nonlattices, but in PTIME for large classes of partial order, including lattices of base types, which are of practical interest. This makes subtyping over lattices of base types particularly interesting both from a practical and a complexity-theoretic (specifically, lower bound) point of view. Our results are summarized in the table:

	structural	nonstructural
finite types	$\frac{\text{coNP}}{\text{coNP}}$	$\frac{\text{PSPACE}}{?}$
infinite types	$\frac{\text{PSPACE}}{\text{PSPACE}}$	$\frac{\text{PSPACE}}{?}$

Here a complexity class above a line indicates a lower bound ("hard for ...") and above a line an upper bound ("contained in ..."). All the PSPACE–results are

shown in this paper; the coNP-results have been reported before [10]. The results show that, unless NP = PSPACE, adding either recursive types or nonstructural subtyping to a basic setting with structural subtyping over finite types makes entailment strictly harder to solve. Note that our PSPACE-completeness result settles the complexity of entailment for structural subtyping over lattices of base types, for both finite and infinite types. Interestingly, nonstructural subtyping, adding just $\bot$ and $\top$, already makes entailment PSPACE-hard, even for finite types.

The study of entailment by Flanagan and Felleisen [8] is related to the present study, although they work in a different model of complete infinite trees labeled with sets of constructors, and their notion of entailment is different from ours. Even though the exact relation between their work and ours remains unclear, we have to some extent been inspired by their methods. Thus, Flanagan [7] proves PSPACE-hardness of their entailment predicate by reduction from the NFA containment problem (see [9].) However, the proof relies on a complicated (and impressive), complete axiomatization of entailment, whereas our proof uses nonsyntactic methods and a different reduction from a special form of the NFA universality problem (their reduction appears not to be directly applicable to our case.) Their axiomatization leads to a complete algorithm for entailment, but since it runs in exponential time and consumes exponential space they do not give a tight classification of complexity.

Due to space limitations, details of some proofs had to be left out. They can be found in [19, 18].

2 Preliminaries

Let Σ be the ranked alphabet of *constructors*, $\Sigma = B \cup \{\times, \to\}$; here $L = (B, \leq_B)$ is a lattice of *constants* (constructors of arity 0, to be thought of as base types such as, *e.g.*, `int`, `real`) and $\times$, $\to$ are binary constructors. The order on L defines the subtype ordering on type constants in the standard way (see, *e.g.*, [14] for a full introduction.) Let $\mathbf{A}$ be the alphabet $\mathbf{A} = \{f, s, d, r\}$, and let $\mathbf{A}^*$ denote the set of finite strings over $\mathbf{A}$; elements of $\mathbf{A}^*$ are called *addresses*. We consider types as *labeled trees* in the style of [12] to which the reader is referred for full information. A *tree* over Σ is a partial function $t : \mathbf{A}^* \to \Sigma$ with nonempty, prefix-closed tree domain $\mathcal{D}(t)$. The elements *d*,*r* select the domain and range, respectively, of a tree with root labeled $\to$, and f, s select the first and second component, respectively, of a tree with root labeled $\times$. Base types (constants) in L are ranged over by b, typical elements of $\mathbf{A}$ are ranged over by a, and typical elements of $\mathbf{A}^*$ are ranged over by w. (see, *e.g.*, [12] for more information.) A tree t is *finite* (resp. *infinite*) if and only if $\mathcal{D}(t)$ is a finite (resp. infinite) set. If $w \in \mathcal{D}(t)$ and w is not a proper prefix of any string in $\mathcal{D}(t)$, then w is called a *leaf address* of t (*i.e.*, $t(w) \in L$). If $w \in \mathcal{D}(t)$ and w is not a leaf address, then w is said to be an *interior address* of t. Whenever w is an interior address in t, then $t(w)$ is completely determined by $\mathcal{D}(t)$, *e.g.*, $t(w) = \times$ if and only if $\{wf, ws\} \subseteq \mathcal{D}(t)$.

We let $\mathcal{T}_\Sigma$ denote the set of trees over Σ. For a partial order $\leq$, we let $\leq^0 = \leq$ and $\leq^1 = \geq$ (the reversed relation.) For $w \in \mathbf{A}^*$, we define the *polarity* of w, denoted πw, to be 0 if w contains an even number of d's and $\pi w = 1$ otherwise. We write $\leq^w$ as a shorthand for $\leq^{\pi w}$. The reversal of order in accordance with polarity captures contravariance of the subtype order with respect to the function space constructor $\rightarrow$ (see [12].) If Σ is equipped with a partial order $\leq_\Sigma$, we induce a partial order on $\mathcal{T}_\Sigma$ by setting

$$t \leq t' \text{ if and only if } \forall w \in \mathcal{D}(t) \cap \mathcal{D}(t').\ t(w) \leq^w_\Sigma t'(w)$$

Let $\mathcal{V}$ be a denumerable set of variables, distinct from elements of Σ. *Terms* or *type expressions* over Σ are *finite* trees in $\mathcal{T}_{\Sigma \cup \mathcal{V}}$, where elements of $\mathcal{V}$ are regarded as constructors of arity 0 (however, only members of Σ of arity 0 are called constants); terms are ranged over by τ. The set of terms is denoted $\mathcal{T}_\Sigma(\mathcal{V})$. A *constraint set* is a finite set of formal inequalities of the form $\tau \leq \tau'$, where τ and τ' are finite terms. We write $\mathsf{Var}(C)$ to denote the set of variables that occur in C. The notation $\leq^w$ may be extended to formal inequalities (denoting reversed formal inequality if $\pi w = 1$.) A constraint set C is said to be *closed* if and only if the following conditions are satisfied:

- (transitivity) $\tau_1 \leq \tau_2 \in C, \tau_2 \leq \tau_3 \in C \Rightarrow \tau_1 \leq \tau_3 \in C$
- (decomposition) $\tau_1 \times \tau_2 \leq \tau_3 \times \tau_4 \in C \Rightarrow \{\tau_1 \leq \tau_3, \tau_2 \leq \tau_4\} \subseteq C$
- (decomposition) $\tau_1 \rightarrow \tau_2 \leq \tau_3 \rightarrow \tau_4 \in C \Rightarrow \{\tau_3 \leq \tau_1, \tau_2 \leq \tau_4\} \subseteq C$

We define the *closure of* C to be the least closed constraint set containing C.

Nonstructural subtyping is obtained by fixing Σ to be the set $\{\times, \rightarrow, \top, \bot\}$ organized as a lattice by the order $\bot \leq \sigma$, $\sigma \leq \top$ for $\sigma \in \Sigma$. See [12] for further details. This organizes $\mathcal{T}_\Sigma$ as a complete lattice under the nonstructural order.

Structural subtyping over a lattice is obtained by extending the lattice $L = (B, \leq_B)$ to the partial order $(B \cup \{\times, \rightarrow\}, \leq_B)$; that is, by adding $\times$ and $\rightarrow$ as new, incomparable elements to L.

Let $\Sigma^\# = \{B, \times, \rightarrow\}$ be a ranked alphabet, where B is nullary and both $\times$ and $\rightarrow$ are binary. We can extend the *signature mapping* $\_^\#$ given by $b^\# = B$ for all $b \in B$, $\times^\# = \times$ and $\rightarrow^\# = \rightarrow$ to a mapping $\_^\#$ from $\mathcal{T}_\Sigma$ to $\mathcal{T}_{\Sigma^\#}$ in the following fashion:

$$t^\#(w) = (t(w))^\#$$

We call $t^\#$ the *shape* of t. This shape mapping can be extended to terms containing variables by adding $t^\#(w) = \alpha$ if $t(w) = \alpha$. The *shape constraints* $C^\#$ are the set of formal equalities (over $\mathcal{T}_{\Sigma^\#}(\mathcal{V})$) obtained from C by mapping every inequality $t_1 \leq t_2 \in C$ to $t_1^\# = t_2^\#$.

One can easily show that, in structural subtyping, we have $t \leq t'$ if and only if $\mathcal{D}(t) = \mathcal{D}(t')$, for every interior address w of t and t', one has $t(w) = t'(w)$, and for every leaf address w of t and t', one has $t(w) \leq^w_B t'(w)$. This implies that the set of structurally ordered trees is a disjoint union of lattices L_s, each lattice L_s consisting of the set of trees having the same shape s. See [20, 21] for more information. In particular, t and t' must have the same shape if $t \leq t'$.

As a standard consequence (which we will be very brief about here, see, *e.g.*, [21, 14, 20] for more details) one has that any constraint set that is satisfiable over structurally ordered trees, must be *weakly unifiable*. This means that $C^\#$ is unifiable over $\mathcal{T}_{\Sigma^\#}(\mathcal{V})$. Let U_C be a most general unifier of $C^\#$. Let $\_^B$ be the mapping from $\mathcal{T}_{\Sigma^\#}(\mathcal{V})$ to $\mathcal{T}_{\Sigma^\#}$ that maps all variables in a shape tree to B. Each variable α in a satisfiable set C can be assigned a unique *shape* $s_C(\alpha)$ by defining $s_C(\alpha) = (U_C(\alpha))^B$.

Both structural and nonstructural subtyping have finite and infinite variants, according to whether variables range over finite or infinite trees; in the infinite case, we talk about *recursive* subtyping. We call a constraint set C *structural* (resp. *nonstructural*) if its intended interpretation is in structurally (resp. nonstructurally) ordered trees. A *valuation* is a ground substitution, *i.e.*, a map $v : \mathcal{V} \to \mathcal{T}_\Sigma$, extended canonically to $\mathcal{T}_\Sigma(\mathcal{V}) \to \mathcal{T}_\Sigma$. A valuation *satisfies* a constraint $\tau \leq \tau'$, written $v \models \tau \leq \tau'$, if $v(\tau) \leq v(\tau')$ is true in the (appropriate, structural or nonstructural) ordered structure $\mathcal{T}_\Sigma$. A constraint set C *entails* a constraint $\tau \leq \tau'$, written $C \models \tau \leq \tau'$, if every valuation satisfying all the constraints in C also satisfies the constraint $\tau \leq \tau'$.

3 Constraint automata

A constraint set can be regarded as a nondeterministic finite automaton: inequalities are viewed as ϵ-transitions and constructed types have transitions on elements of $\mathbf{A}$. For a given constraint set C, we first define a labeled digraph, called the *constraint graph* associated with C and denote it $\mathcal{G}_C = (V, E)$. The vertex set V is the set of subterms occurring in C. The edge relation E in $\mathcal{G}_C$ is a set of edges labeled with elements of $\mathbf{A} \cup \{\epsilon\}$; an edge from τ to τ' labeled with $a \in \mathbf{A} \cup \{\epsilon\}$ is written $\tau \mapsto_a \tau'$, for $\tau, \tau' \in V$. The set E of labeled edges is defined as the least relation $E \subseteq V \times (\mathbf{A} \cup \{\epsilon\}) \times V$ satisfying the following conditions:

- For all $\tau \in V$, $\tau \mapsto_\epsilon \tau$
- If $\tau \leq \tau' \in C$, then $\tau \mapsto_\epsilon \tau'$
- If $\tau = \tau_1 \times \tau_2$ is in V, then $\tau \mapsto_f \tau_1$ and $\tau \mapsto_s \tau_2$
- If $\tau = \tau_1 \to \tau_2$ is in V, then $\tau \mapsto_d \tau_1$ and $\tau \mapsto_r \tau_2$

The relation $\mapsto_\epsilon$ represents (the reflexive closure of) the inequalities in C, by the first two conditions. The next two conditions represent the syntactic structure of terms by the relations $\mapsto_a$, $a \in \mathbf{A}$. The constraint graph $\mathcal{G}_C$ can be directly regarded as a nondeterministic finite automaton (NFA) over the alphabet $\mathbf{A}$, where the labeled edge relation E defines the transition relation δ_N. The *nondeterministic constraint automaton* (NCA for short) associated with a set C and with a specified start state q_0, is denoted $\mathcal{A}_N^C(q_0) = (Q, \mathbf{A}, \delta_N, q_0, Q)$. The state set Q is V; the alphabet Σ is $\mathbf{A}$; every state is an accept state. The iterated transition relation, denoted $\widehat{\delta}_N$, is defined in the standard way (see [11]).

Let C be structural and weakly unifiable. A *C-shaped valuation* is one which maps every variable α in C to a tree of shape $s_C(\alpha)$. One can show (details in

[19], left out in this summary) that a structural set C is satisfiable if and only if it is satisfied by a C-shaped valuation. Moreover, one can show that, if C is satisfiable and $\alpha, \beta \in \mathsf{Var}(C)$ satisfy some easily (polynomial time) recognizable conditions (called *nontriviality conditions* in [19]) in C, then $C \models \alpha \leq \beta$ if and only if it holds for any *C-shaped* valuation v that $v \models \alpha \leq \beta$ whenever $v \models C$; and in case the nontriviality conditions are not satisfied, one can easily (in polynomial time) decide entailment. This amounts to saying that, in order to decide structural subtype entailment — over finite or infinite trees — it is sufficient to consider entailment restricted to C-shaped valuations.

A set C which does not mention the constructor $\rightarrow$ is called *monotone*. Any monotone, nonstructural set has a largest solution, and any monotone structural set has a largest C-shaped solution.

4 Nonstructural subtyping

An NFA (nondeterministic finite automaton) is called *prefix-closed*, if all its states are accept states. Thus, the only way a prefix-closed NFA can fail to accept a word w is by getting into a stuck state on input w. Let CLOSED-UNIV denote the computational problem:

- Given a *prefix-closed* NFA A over a nontrivial alphabet Σ, is it the case that $L(A) = \Sigma^*$?

By a construction due to Vaughan Pratt [16] we can show that CLOSED-UNIV is PSPACE-complete for any nontrivial alphabet, by reduction from the standard universality problem for NFA's [9].

Consider the two letter alphabet $\{f, s\} \subseteq \mathbf{A}$ and let ϵ denote the empty string. We will show that, for any closed NFA A over $\{f, s\}$, we can construct a constraint set C_A such that $L(A) = \{f, s\}^*$ if and only if $C_A \models \alpha_0 \leq \beta$, where α_0 and β are distinct variables used in the construction of C_A. The types in C_A will be built from variables and the pairing constructor only. The resulting construction will be a log-space reduction of CLOSED-UNIV to the entailment problem $C \models \alpha \leq \beta$, thereby proving the latter problem PSPACE-hard. The main idea is to simulate A by the constraint automaton $\mathcal{A}_N^{C_A}$.

We proceed to describe the construction of C_A. Assume we are given the closed NFA $A = (Q, \Sigma^\times, \delta, q_0, Q)$ (state set is Q, alphabet Σ is $\{f, s\}$, transition relation is δ, start-state is q_0, and — since the automaton is prefix-closed — the set of accept states is the entire set of states, Q.) We write $q_i \mapsto^w q_j$ to indicate that there is a w-transition from state q_i to state q_j in A. If $w \in \{f, s\} \cup \{\epsilon\}$, we say that a transition $q_i \mapsto^w q_j$ is *simple*. A is completely determined by its simple transitions, since these just define its transition relation.

Suppose that A has n simple transitions, and assume them to be ordered in some aritrary, fixed sequence. For the k'th simple transition $q_i \mapsto^w q_j$ we define a constraint set C_k; we associate a distinct variable α_i to each state q_i (so, in particular, the variable α_0 corresponds to the start state of A), and we associate a distinct variable δ_k to each k. The construction of C_k is as follows.

If the k'th simple transition in A is $q_i \mapsto^f q_j$, then $C_k = \{\alpha_i \leq \alpha_j \times \delta_k\}$; if the k'th simple transition in A is $q_i \mapsto^s q_j$, then $C_k = \{\alpha_i \leq \delta_k \times \alpha_j\}$; if the k'th simple transition in A is $q_i \mapsto^\epsilon q_j$, then $C_k = \{\alpha_i \leq \alpha_j\}$. Now define C_A to be

$$C_A = (\bigcup_{k=1}^{n} C_k) \cup \{\beta = \beta_1 \times \beta_2, \beta = \beta_1, \beta = \beta_2\}$$

where the notation $t = t'$ abbreviates the two inequalities $t \leq t'$ and $t' \leq t$. Notice that any solution to C_A must map β to the complete, infinite binary tree $t^\infty = \mu\gamma.\gamma \times \gamma$, hence β is just a name for that tree.

Theorem 1. *Nonstructural recursive subtype entailment is PSPACE-hard.*

Proof. By reduction from CLOSED-UNIV. We prove the following property. Let A be a prefix-closed NFA over $\{f, s\}$. Then

$$L(A) = \{f, s\}^* \text{ if and only if } C_A \models \alpha_0 \leq \beta$$

First notice that, by construction of C_A, there is a transition $q_i \mapsto^w q_j$ in A if and only if there is a transition $\alpha_i \mapsto^w \alpha_j$ in the constraint automaton $\mathcal{A}_N^{C_A}$. We first prove the implication

$$L(A) \neq \{f, s\}^* \Rightarrow C_A \not\models \alpha_0 \leq \beta$$

So assume that there exists a word $w \in \{f, s\}^*$ such that $w \notin L(A)$. Since we can assume w.l.o.g. that A has at least one state, and since A is prefix-closed, we have $\epsilon \in L(A)$, so $L(A) \neq \emptyset$. Then there is a prefix w' of w of maximal length such that $w' \in L(A)$; we can assume that w can be written as $w = w'fw''$ (the alternative case $w = w'sw''$ is similar.) Now, *because A is a prefix-closed automaton*, it follows that for *any* state q_k such that $q_0 \mapsto^{w'} q_k$, there can be no state q_l such that $q_k \mapsto^f q_l$; inspection of the construction of C_A then shows that, for any k such that $q_0 \mapsto^{w'} q_k$, the only nonvariable upper bounds in the transitive closure of C_A on α_k are of the form $\alpha_k \leq \delta_m \times \alpha_s$, where δ_m is unbounded in C_A; one can then show that the largest solution v^* to C_A (which is monotone) satisfies either $v^*(\alpha_0)(w') = \top$ or else $v^*(\alpha_0)(w'f) = \top$. In either case, $v^*(\alpha_0) \not\leq \beta$, thereby showing $C_A \not\models \alpha_0 \leq \beta$.

To prove the implication

$$L(A) = \{f, s\}^* \Rightarrow C_A \models \alpha_0 \leq \beta$$

let w be an arbitrary element in $\{f, s\}^*$. Then $w\ell \in L(A)$, $\ell \in \{f, s\}$, assuming the left hand side of the implication. Then there is a transition $q_0 \mapsto^w q_j \mapsto^\ell q_k$ for some q_j, q_k; by construction of C_A, there is a transition $\alpha_0 \mapsto^{wf} \alpha_k$ or $\alpha_0 \mapsto^{ws} \alpha_k$ in $\mathcal{A}_N^{C_A}$. Then, using the special form of C_A (only $\times$ and variables occur there), one can verify that $w \in \mathcal{D}(v^*(\alpha_0))$ with $v^*(\alpha_0)(w) = \times$, where v^* is the largest solution to C_A. It follows that, for any $w \in \{f, s\}^*$, one has $v(\alpha_0)(w) \leq \times$, whenever v is a solution to C_A, thereby showing $C_A \models \alpha_0 \leq \beta$.

4.1 PSPACE-hardness for finite subtyping

Entailment over finite, structural trees ordered over a lattice of type constants was shown to be coNP–complete in [10]. Intuitively, membership in coNP follows from the fact that the nonentailment problem $C \not\models \alpha \leq \beta$ has a succinct certificate in the form of a word $w \in \mathbf{A}^*$ of length at most n (size of constraint set) identifying a common leaf in $v(\alpha)$ and $v(\beta)$ for a possible solution v to C such that $v(\alpha)(w) \not\leq^w v(\beta)(w)$. It turns out that the corresponding problem with non–structural order is PSPACE-hard. The reason is that infinite trees can be approximated in the nonstructurally ordered space of finite trees. In order to prove PSPACE-hardness, we shall need to talk about approximations of infinite trees by finite trees. To this end, recall (from [4, 12]) the definition of the *level-k truncation* of a tree t, denoted $t\lceil_k$; it has domain $\mathcal{D}(t\lceil_k) = \{w \in \mathcal{D}(t) \mid |w| \leq k\}$ and is defined by

$$t\lceil_k(w) = \begin{cases} t(w) & \text{if } |w| < k \\ \bot & \text{if } |w| = k \end{cases}$$

This definition is simplified, since we shall restrict ourselves to the monotonic alphabet without $\rightarrow$. A monotone constraint set C is called *directed* if and only if every inequality in C has either one of the following forms: $\alpha \leq \alpha'$ or $\alpha \leq \alpha_1 \times \alpha_2$. Then it is easy to see that the sets C_A are directed, when the equations defining β are removed. Let $\mathcal{T}_\Sigma^{\mathrm{F}}$ denote the set of finite trees with nonstructural order. If v is a valuation in $\mathcal{T}_\Sigma$, we define for all $k \geq 0$ the valuation $v\lceil_k$ in $\mathcal{T}_\Sigma^{\mathrm{F}}$ by $v\lceil_k(\alpha) = v(\alpha)\lceil_k$.

We can now show that entailment over $\mathcal{T}_\Sigma^{\mathrm{F}}$ is PSPACE-hard. Given an NFA A, we define a constraint set C_A^{F}; it is defined exactly as C_A in Section 4, except that, instead of the equations $\beta = \beta_1 \times \beta_2, \beta = \beta_1, \beta = \beta_2$, we now take the single inequality $\beta \times \beta \leq \beta$.

Theorem 2. *Nonstructural subtype entailment over finite trees is PSPACE-hard.*

Proof. (Sketch) By reduction from CLOSED-UNIV, using the following property. Let C be directed, and let v be a valuation in $\mathcal{T}_\Sigma$ such that $v \models C$. Then one has

$$\forall k \geq 1.\ v\lceil_k \models C$$

in $\mathcal{T}_\Sigma^{\mathrm{F}}$. Using this property, one can show that, if A is a prefix-closed NFA over $\{f, s\}$, then

$$L(A) = \{f, s\}^* \text{ if and only if } C_A^{\mathrm{F}} \models \alpha_0 \leq \beta$$

holds over $\mathcal{T}_\Sigma^{\mathrm{F}}$. If $L(A) = \{f, s\}^*$ and $v \models C_A^{\mathrm{F}}$, then $v(\alpha_0) \leq t^\infty$ must hold in $\mathcal{T}_\Sigma$, as shown in the proof of Theorem 1. But $t^\infty \leq v(\beta)$ must hold, yielding $v(\alpha_0) \leq v(\beta)$ in $\mathcal{T}_\Sigma^{\mathrm{F}}$. If $L(A) \neq \{f, s\}^*$, then $v \models C_A^{\mathrm{F}}$ with $v(\alpha_0) \not\leq v(\beta)$ for some v in $\mathcal{T}_\Sigma$, by the argument in Theorem 1; by passing to $v\lceil_k$ for sufficiently large k (using the property above), one can then show $C_A^{\mathrm{F}} \not\models \alpha_0 \leq \beta$ in $\mathcal{T}_\Sigma^{\mathrm{F}}$. More details can be found in [19, 18].

The problem of giving an upper bound on nonstructural entailment remains open. In Section 5.2 we will show that structural recursive subtype entailment is in PSPACE. We believe that an elaboration of the ideas in the PSPACE–algorithm sketched there can be used to give a PSPACE decision procedure for nonstructural entailment also.

Conjecture 1. The problem of subtype entailment over nonstructural trees is PSPACE–complete, both in the finite and in the general case

5 Structural Recursive Subtyping

5.1 PSPACE-hardness

The reduction used to prove PSPACE-hardness for nonstructural recursive subtyping in Section 4 does not transfer directly to the structural case. To see the difference, let A be the NFA with all states accepting, start state q_0 and transitions $q_0 \mapsto^s q_1$, $q_1 \mapsto^f q_1$, $q_0 \mapsto^s q_2$, $q_2 \mapsto^s q_2$, $q_0 \mapsto^f q_3$, $q_3 \mapsto^f q_3$, $q_0 \mapsto^s q_4$, $q_4 \mapsto^s q_4$, and consider the constraint set C_A as defined in Section 4. A accepts (in addition to the empty string) just the strings over $\{f, s\}$ of either one of the forms ff^n, fs^n, ss^n, sf^n, $n \geq 0$. Thus, we certainly have $L(A) \neq \{f, s\}^*$. However, by unifying C_A (taking inequalities as equations under unification) the reader can easily verify that, due to the structural shape requirements on α_0 in C_A, any solution to C_A must map α_0 to the complete, infinite binary tree $t^\infty = \mu\gamma.\gamma \times \gamma$. What happens under unification in C_A corresponds to collapsing the states q_1, q_2, q_3, q_4 of A into a single state. This means that, for this particular set C_A, we have $C_A \models \alpha_0 \leq t^\infty$ but not $L(A) = \{f, s\}^*$, and our previous reduction is seen to be incorrect for the structural case. However, the previous reduction can be extended to cover the structural case, as we will now show.

The basic idea is to allow the start state α_0 to have one particluar, fixed shape, which will be compatible with all inequalities generated from A. As can be seen from our example, that shape must presumably be infinite. Define the infinite trees $t^\bot$ and $t^\top$ by

$$t^\bot = \mu\gamma.(\gamma \times \bot) \times (\gamma \times \bot) \text{ and } t^\top = \mu\gamma.(\gamma \times \top) \times (\gamma \times \top)$$

Our reduction, then, will be such that in all cases α_0 will satisfy

$$t^\bot \leq \alpha_0 \leq t^\top \tag{1}$$

The encoding of an automaton A will only depend upon whether or not the leaves in the value of α_0 must be $\bot$, in any solution to C_A. We now give the definition of a constraint set C_A constructible in logspace from a given NFA A. As before, we construct C_A from sets C_k corresponding to each k'th simple transition in A. The C_k are defined as follows. If the k'th simple transition in A is $q_i \mapsto^f q_j$, then $C_k = \{\alpha_i \leq (\alpha_j \times \bot) \times \gamma_k\}$; if the k'th simple transition in A is $q_i \mapsto^s q_j$, then $C_k = \{\alpha_i \leq \gamma_k \times (\alpha_j \times \bot)\}$; if the k'th simple transition in A is $q_i \mapsto^\epsilon q_j$, then $C_k = \{\alpha_i \leq \alpha_j\}$. Here the γ_k are fresh variables. The set C_A

is defined to be the inequality shown in (1) together with the union of all the C_k. In addition to the constraints above, we add the constraint (1) to C_A. The trees $t^\top$ and $t^\bot$ can be defined by regular sets of equations.) One can then show (details in [19, 18]) that, if A is a prefix-closed NFA over $\Sigma = \{f, s\}^*$, then

$$L(A) = \{f, s\}^* \text{ if and only if } C_A \models \alpha_0 \leq t^\bot$$

This leads to

Theorem 3. *Structural recursive subtype entailment is PSPACE–hard.*

5.2 PSPACE upper bound

Our PSPACE upper bound for deciding structural recursive subtype entailment proceeds by exhibiting a nondeterministic decision algorithm for nonentailment (*i.e.*, deciding $C \not\models \alpha \leq \beta$?) that runs in polynomial space. The PSPACE result follows, because PSPACE = NPSPACE = coNPSPACE by Savitch's Theorem. Our algorithm, in turn, is founded on a characterization theorem, which we proceed to present.

In order to state the characterization, we first define a relation R,

$$R \subseteq V \times V \times \{\downarrow, \uparrow\} \times \mathbf{A}^*$$

which intuitively captures properties of the closure of a constraint set. In order to define R we first define, for $a \in \mathbf{A}$, the operation a on $\{\downarrow, \uparrow\}$ by: for $a \neq d$, $a(\uparrow) = \uparrow$ and $a(\downarrow) = \downarrow$, and for $a = d$ we set $a(\uparrow) = \downarrow$ and $a(\downarrow) = \uparrow$. These operations will be used to make sure that contravariance is respected by the algorithm. Intuitively, the relation R holds of $(v, v', \uparrow, w)$ if there is a w-path in $\mathcal{G}_C$ from v to v' where ϵ-edges are followed in reversed direction according to the polarity of w. The relation R is defined by induction on $w \in \mathsf{A}^*$, as follows:

$$\begin{aligned}
R(v, v', \uparrow, \epsilon) &\Leftrightarrow v \mapsto_\epsilon v' \\
R(v, v', \downarrow, \epsilon) &\Leftrightarrow v' \mapsto v \\
R(v, v', \uparrow, aw) &\Leftrightarrow \exists v_1, v_2.\ v \mapsto_\epsilon v_1 \wedge v_1 \mapsto_a v_2 \wedge R(v_2, v', a(\uparrow), w) \\
R(v, v', \downarrow, aw) &\Leftrightarrow \exists v_1, v_2.\ v_1 \mapsto_\epsilon v \wedge v_1 \mapsto_a v_2 \wedge R(v_2, v', a(\downarrow), w)
\end{aligned}$$

Let $\uparrow_C^w (v) = \{v' \in V \mid R(v, v', \uparrow, w)\}$ and $\downarrow_C^w (v) = \{v' \in V \mid R(v, v', \downarrow, w)$ and define $\alpha \preceq_C^w \beta$ to hold if and only if there exists a prefix w' of w such that $\uparrow_C^{w'} (\alpha) \cap \downarrow_C^{w'} (\beta) \neq \emptyset$. Let $\bigwedge^w$ be the greatest lower bound operator in L, if $\pi w = 0$, and the lest upper bound operator if $\pi w = 1$; dually, let $\bigvee^w$ denote least upper bound in the first case and greatest lower bound in the second case. We can prove the following theorem which characterizes entailment over C-shaped valuations. The theorem is a (nontrivial) generalization of a similar theorem for atomic constraints over L, proved by the authors in [10]. A proof can be found in [19, 18].

Theorem 4. *$C \models \alpha \leq \beta$ (over C-shaped valuations) if and only if one of the following conditions holds for every address w of maximal length in the common shape of α and β in C:*

1. $\alpha \preceq_C^w \beta$ *or*
2. $\bigwedge^w \uparrow_C^w (\alpha) \leq_B^w \bigvee^w \downarrow_C^w (\beta)$

This theorem yields the central algorithmic idea for our algorithm: Given a weakly unifiable, consistent constraint set C in which α and β occur nontrivially, we guess a path $w \in \mathcal{D}(s_C(\alpha))$ and check that both conditions of Theorem 4 are violated. If so, we accept (which corresponds to reporting "not-entails"), otherwise we reject. More precisely, we iterate the following loop, with w initialized to ϵ:

1. If $\alpha \preceq_C^w \beta$, then reject; otherwise
2. If w is maximal and $\bigwedge^w \uparrow_C^w (\alpha) \leq_B^w \bigvee^w \downarrow_C^w (\beta)$, then reject; otherwise
3. If w is a maximal address then accept; otherwise
4. If w is not maximal, guess $a \in \{f, s, d, r\}$ such that $wa \in \mathcal{D}(s_C(\alpha))$, set $w := wa$ and go to Step 1.

What makes this a (nondeterministic) PSPACE algorithm for deciding nonentailment is that we do not actually need to store w, only $\uparrow_C^w (\alpha), \downarrow_C^w (\alpha), \uparrow_C^w (\beta)$ and $\downarrow_C^w (\beta)$ because both conditions of Theorem 4 require only $\uparrow_C^w (\alpha), \downarrow_C^w (\beta)$ and the $s_C(\alpha)$. Furthermore, $\uparrow_C^{wa} (\alpha), \downarrow_C^{wa} (\alpha), \uparrow_C^{wa} (\beta), \downarrow_C^{wa} (\beta)$ can be computed from a and $\uparrow_C^w (\alpha), \downarrow_C^w (\alpha), \uparrow_C^w (\beta), \downarrow_C^w (\beta)$ in polynomial time and space. Thus the algorithm requires only space for $\uparrow_C^w (\alpha), \downarrow_C^w (\alpha), \uparrow_C^w (\beta), \downarrow_C^w (\beta)$ and $s_C(\alpha)$, which is polynomial in the number of vertices in the constraint graph and thus also in the size of the input. Full details of the algorithm are given in [19, 18].

Theorem 5. *The problem of structural subtype entailment over infinite trees is PSPACE-complete.*

References

1. A. Aiken and E.L. Wimmers. Type inclusion constraints and type inference. In *Proceedings FPCA '93, Conference on Functional Programming Languages and Computer Architecture, Copenhagen, Denmark*, pages 31–42, June 1993.
2. A. Aiken, E.L. Wimmers, and J. Palsberg. Optimal representations of polymorphic types with subtyping. In *Proceedings TACS '97, Theoretical Aspects of Computer Software, Sendai, Japan*, pages 47–77. Springer Lecture Notes in Computer Science, vol. 1281, September 1997.
3. R. Amadio and L. Cardelli. Subtyping recursive types. In *Proc. 18th Annual ACM Symposium on Principles of Programming Languages (POPL), Orlando, Florida*, pages 104–118. ACM Press, January 1991.
4. R. Amadio and L. Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, September 1993.
5. W. Charatonik and A. Podelski. The independence property of a class of set constraints. In *Conference on Principles and Practice of Constraint Programming*, pages 76–90. Springer-Verlag, 1996. Lecture Notes in Computer Science, Vol. 1118.
6. M. Fahndrich and A. Aiken. Making set-constraint program analyses scale. In *Workshop on Set Constraints, Cambridge MA*, 1996.
7. C. Flanagan. Personal communication, March 1997.

8. C. Flanagan and M. Felleisen. Componential set-based analysis. In *Proceedings of the ACM SIGPLAN '97 Conference on Programming Language Design and Implementation*. ACM, June 1997.
9. M. Garey and D. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman, 1979.
10. F. Henglein and J. Rehof. The complexity of subtype entailment for simple types. In *Proceedings LICS '97, Twelfth Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland*, pages 352–361. IEEE Computer Society Press, June 1997.
11. J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
12. Dexter Kozen, Jens Palsberg, and Michael Schwartzbach. Efficient recursive subtyping. In *Proc. 20th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, pages 419–428. ACM, ACM Press, January 1993.
13. S. Marlow and P. Wadler. A practical subtyping system for erlang. In *2nd International Conference on Functional Programming, Amsterdam*. ACM, June 1997.
14. J.C. Mitchell. Type inference with simple subtypes. *Journal of Functional Programming*, 1(3):245–285, July 1991.
15. F. Pottier. Simplifying subtyping constraints. In *Proceedings ICFP '96, International Conference on Functional Programming*, pages 122–133. ACM Press, May 1996.
16. V. Pratt. Personal communication, May 1997.
17. J. Rehof. Minimal typings in atomic subtyping. In *Proceedings POPL '97, 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Paris, France*, pages 278–291. ACM, January 1997.
18. J. Rehof. The complexity of simple subtyping systems. Technical report, DIKU, Dept. of Computer Science, University of Copenhagen, Denmark. Available at http://www.diku.dk/research-groups/topps/personal/ rehof/publications.html, April 1998. Thesis submitted for the Ph.D. degree.
19. Jakob Rehof. Report on constraint automata and recursive subtype entailment. Technical report, DIKU, Dept. of Computer Science, University of Copenhagen, Denmark. Available at http://www.diku.dk/research-groups/topps/personal/ rehof/publications.html, 1998.
20. 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.
21. J. Tiuryn and M. 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.
22. 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.