Internship report: Uniform and non-structural subtyping

Stéphane Glondu
supervised by Zhendong Su
University of California
Davis, USA

June–August, 2005

Abstract

We expose a new approach to tackle non-structural subtyping problems. We introduce uniform subtyping as a means to capture some properties of non-structural subtyping. In the uniform theory, we show that the validity of a first-order sentence is decidable, and entailment is PSPACE-hard. In addition, we give decidable approximations to entailment and subtyping constrained types — two problems which are still open in the non-structural theory.

Contents

1 Introduction

1.1 Context of the internship

This report is relative to the 11-week internship I made with Professor Zhendong Su at the Computer Science Department of the University of California, Davis.

His research interests span programming languages, software engineering and computer security, focusing on programming language-based techniques and tools for improving and ensuring software reliability and security.

I worked mainly on non-structural subtype entailment, a problem he tackled during his PhD, but which is a side project now. Even though this problem is still open, many intermediate results have been proved in the domain of subtyping while trying to solve it. My work was to investigate Prof. Su's new intuitions on subtyping — and maybe get new ones by myself.

1.2 Introduction of the subject

Typing is a powerful static analysis tool which detects unsound code. The basic idea is that every value in a programme must have a type. There is a form of typing in (almost) every programming language. For example, an integer constant cannot (or should not) be used as a function. With *type checking*, the programmer must provide some types for his programme, and the compiler checks the validity of these types. It is implemented in most of the languages. With *type inference*, the programmer does not have to worry about types, the compiler infers antomatically all the types from the constants. Some languages such as Objective Caml use very sophisticated typing systems with inference.

However, a value can have several types. Sometimes, some structures such as lists can be manipulated regardless of the type of their contents, but they do have a different type. In this case, they have a *polymorphic type*, which is a parametrized type (such as $\alpha\ \mathsf{list} \to \alpha$ for a function which extracts the first element of a list). This phenomenon is pretty well understood now, and it is implemented in practice in the last version of Java for example.

In other cases, we want to modelize a subset relationship between some types: an integer can be seen as a real, or an object with a method move can be seen as an object with no method at all. Many languages use ad-hoc implicit *casting* to overcome this problem. But the theoretical aspects are not yet totally understood.

I worked on the latter point, called *subtyping*. Intuitively, a type τ_1 is a *subtype* of a type τ_2 — written $\tau_1 \leqslant \tau_2$ — if one can (soundly) use a value of type τ_1 where a value of type τ_2 is expected.

2 Preliminaries on subtyping

In this section, we review basic concepts on subtyping. We discuss the various choices of type expression languages, their signatures, subtype orders, and the

notion of subtype constraints. We also introduce some problems related to subtyping: first-order theories and entailment.

2.1 Types as trees

Types can be viewed as trees over some ranked alphabet Σ, the *signature* of the given type language. A signature consists of a finite set of function symbols, the *type constructors*. Each symbol f has an associated $\mathsf{arity}(f) \geqslant 0$ — indicating its number of arguments — and, for all $1 \leqslant i \leqslant \mathsf{arity}(f)$, a *polarity* $\mathsf{pol}(f,i) \in \{-1,1\}$. We call a position i *covariant* if $\mathsf{pol}(f,i) = 1$, and *contravariant* otherwise. Symbols with arity zero are *type constants*. Non-constant types are *constructed*.

We identify *nodes* π of trees with relative addresses from the root of the tree. A word πi addresses the i-th child of node π, and $\pi\pi'$ the π' descendant of π. The root is represented by the empty word ϵ. We define a *tree* τ over Σ as a partial function $\tau : (\mathbb{N} - \{0\})^* \to \Sigma$. Tree domains are prefixed closed, non-empty and arity consistent. A tree τ is *finite* if its domain D_τ is finite, and *infinite* otherwise. We write $\mathcal{T}_\Sigma$ for the set of possibly infinite trees over Σ.

Given a function symbol f with $n = \mathsf{arity}(f)$ and trees $\tau_1, \ldots, \tau_n \in \mathcal{T}_\Sigma$, we define $f(\tau_1, \ldots, \tau_n)$ as the unique tree τ with $\tau(\epsilon) = f$ and $\tau(i\pi) = \tau_i(\pi)$. We define the *polarities* of nodes in trees as follows:

$$\mathsf{pol}_\tau(\epsilon) \stackrel{\text{def}}{=} 1$$
$$\mathsf{pol}_{f(\tau_1,\ldots,\tau_n)}(i\pi) \stackrel{\text{def}}{=} \mathsf{pol}(f,i) \times \mathsf{pol}_{\tau_i}(\pi)$$

For partial orders $\leqslant$, we denote by $\leqslant^1$ the order $\leqslant$ itself, and by $\leqslant^{-1}$ the reversed relation $\geqslant$. Subtype orders are partial orders on trees over some signature Σ. We will denote types by variations of τ and paths by variations of π.

2.2 Structural subtyping

A *structural* subtype order is parametrized by a (partially) ordered set $(B, \leqslant_B)$ representing constant types and a set of non-constant constructors F. The signature is $\Sigma = B \cup F$. We define inductively the relation $\leqslant_\mathsf{S}$ on *finite* Σ-trees:

- $b \leqslant_\mathsf{S} b'$ iff $b \leqslant_B b'$ for all $b, b' \in B$;
- $f(\tau_1, \ldots, \tau_n) \leqslant_\mathsf{S} f(\tau'_1, \ldots, \tau'_n)$ iff $f \in F$, $\mathsf{arity}(f) = n$, and for all $1 \leqslant i \leqslant n$, $\tau_i \leqslant_\mathsf{S}^{\mathsf{pol}(f,i)} \tau'_i$.

More generally, we define $\mathsf{cut}_k(\tau)$ by:

$$\mathsf{cut}_k(\tau)(\pi) \stackrel{\text{def}}{=} \begin{cases} \tau(\pi) & \text{if } \pi \in D_\tau \text{ and } |\pi| < k \\ \tau(\pi) & \text{if } \pi \in D_\tau,\ \tau(\pi) \in B \text{ and } |\pi| = k \\ \star & \text{if } \pi \in D_\tau,\ \tau(\pi) \in F \text{ and } |\pi| = k \end{cases}$$

where $\star$ is some arbitrary, fixed symbol in B. For *any* Σ-trees τ_1 and τ_2, $\tau_1 \leqslant_\mathsf{S} \tau_2$ iff $\mathsf{cut}_k(\tau_1) \leqslant_\mathsf{S} \mathsf{cut}_k(\tau_2)$ for all $k \in \mathbb{N}$.

Example 1. Consider $B = \{\mathsf{int}, \mathsf{real}\}$, with $\mathsf{int} \leqslant \mathsf{real}$, and $F = \{\rightarrow, \times\}$ (the usual constructors). From a function $f : \mathsf{real} \rightarrow \mathsf{real}$, you can build a sequence $(u_n) : \mathsf{int} \rightarrow \mathsf{real}$. This can be summarized in $\mathsf{real} \rightarrow \mathsf{real} \leqslant \mathsf{int} \rightarrow \mathsf{real}$. Two integers can be seen as two reals: $\mathsf{int} \times \mathsf{int} \leqslant \mathsf{real} \times \mathsf{real}$.

2.3 Non-structural subtyping

In *non-structural* subtyping, two distinguished constants are added to structural type languages, a *smallest type* $\bot$ and a *largest type* $\top$. The order $\leqslant_{\mathsf{NS}}$ is defined as $\leqslant_{\mathsf{S}}$ with the additional rule $\bot \leqslant_{\mathsf{NS}} \tau \leqslant_{\mathsf{NS}} \top$ for all τ.

Example 2. $\bot$ can be the type of a computation that does not terminate properly, such as an exception raise or an ill-founded recursion. In the ML system, this is denoted by an unbound type variable. $\top$ can be the type of something which contains no data, such as an empty object or `()` in Objective Caml.

Example 3. Non-structural subtyping modelizes objects: two comparable objects do not have necessarily the same set of methods.

2.4 Uniform subtyping

Like in [4], we use *uniform* subtyping as an intermediate notion. In uniform subtyping, the signature Σ has a partial order $\leqslant_\Sigma$, all the constructors have the same polarities and the same arity $k \geqslant 2$ (there is no constant). Therefore, all types are infinite trees — actually total functions from $\{1, \ldots, k\}^*$ to Σ. Moreover, the polarity $\mathsf{pol}_\tau(\pi)$ of a node is independant of τ: we will write $\mathsf{pol}(\pi)$. The order $\leqslant_{\mathsf{U}}$ is defined on Σ-trees by:

$$\tau_1 \leqslant_{\mathsf{U}} \tau_2 \iff \forall \pi \in \{1, \ldots, k\}^*, \quad \tau_1(\pi) \leqslant_\Sigma^{\mathsf{pol}(\pi)} \tau_2(\pi)$$

Remark 4. Classical infinite types are recursive types. We denote by $\mu\alpha.\tau$ — where α is a type variable and τ is a finite term using variables (possibly α) — the solution of the equation $\alpha = \tau$ (we will not discuss its existence and uniqueness here). If $g \in \Sigma$, $g^\infty \stackrel{\text{def}}{=} \mu\alpha.g(\alpha, \alpha)$, i.e. g^∞ is the infinite binary tree whose all nodes are labelled g.

Example 5. Consider $\Sigma = \{\bot, \top\}$, with $\bot \leqslant \top$, binary constructors, contravariant in the first argument, and covariant in the second one. Let $\tau_\bot$ and $\tau_\top$ be the solutions to the system:

$$\tau_\bot = \bot(\tau_\top, \tau_\bot)$$
$$\tau_\top = \top(\tau_\bot, \tau_\top)$$

Then $\tau_\bot \leqslant \tau_\top$ holds.

Remark 6. By putting an appropriate order on Σ, we could have defined structural and non-structural subtype orders in this way:

$$\tau_1 \leqslant \tau_2 \iff \forall \pi \in D_{\tau_1 \cap \tau_2}, \quad \tau_1(\pi) \leqslant_\Sigma^{\mathsf{pol}_{\tau_1}(\pi)} \tau_2(\pi)$$

2.5 Constraints and first-order logics

Atomic constraints are equations or inequations between type variables. One can combine constraints and logical connectors to get *formulas*. Here, we consider only first-order formulas. In the subsequent proofs, we will use the following syntax for atomic constraints C and formulas ϕ:

$$C ::= x \leqslant y \mid x = f(x_1, \ldots, x_n)$$
$$\phi ::= C \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid \exists x.\phi_1$$

where $x, x_1, \ldots, x_n$ denote variables, f denote an n-ary constructor. One can check that other constraints and formulas (such as disjunctions or universal quantifications) can be expressed in this syntax.

A *constraint* is a conjunction of atomic constraints. S_1, NS_1 and U_1 will denote the first-order logics in the structural, non-structural and uniform theories respectively.

Given a signature Σ, an *assignment* is a mapping from variables to Σ-terms. Assignments will be denoted by variations of σ and ρ. We define the relation $\sigma \models \phi$ (σ *satisfies* ϕ) in the usual way:

$$\begin{aligned}
\sigma \models x \leqslant y &\iff \sigma(x) \leqslant \sigma(y)\\
\sigma \models x = f(x_1, \ldots, x_n) &\iff \sigma(x) = f(\sigma(x_1), \ldots, \sigma(x_n))\\
\sigma \models \phi_1 \wedge \phi_2 &\iff \sigma \models \phi_1 \text{ and } \sigma \models \phi_2\\
\sigma \models \neg\phi_1 &\iff \sigma \not\models \phi_1\\
\sigma \models \exists x.\phi_1 &\iff \text{there exists } \tau \text{ such that } \sigma[x := \tau] \models \phi_1
\end{aligned}$$

We will also say that σ *k-satisfies* ϕ (written $\sigma \models^k \phi$) if $\mathsf{cut}_k \circ \sigma \models \phi$. The symbol $\models$ will be tagged by NS or U in the non-structural and uniform theories respectively.

A *sentence* is a closed formula (i.e. without free variables), and a formula is *satisfiable* if there exists an assignment which satisfies it. A sentence is *valid* if it is satisfied by the empty assignment.

2.6 Constrained types and entailment

Corresponding to polymorphic type schemes in Hindley/Milner style type systems, polymorphic subtype systems have so-called *constrained types*, in which a type is restricted by a system of constraints. An ML style polymorphic type can be viewed as a constrained type with no constraint.

Example 7. Let $\mathsf{double} \stackrel{\text{def}}{=} \lambda f.\lambda x.f(fx)$. The usual type for this function in the ML system is $(\alpha \to \alpha) \to \alpha \to \alpha$. However, double has also the constrained type — more general — $(\alpha \to \beta) \to \alpha \to \beta \backslash \{\beta \leqslant \alpha\}$.

In practice, constrained types can be large and complicated. Therefore, it is important to simplify the types to make them and the associated constraints smaller. Type and constraint simplification is related to the following decision problem of *constraint entailment*: a constraint C *entails* a constraint $\tau_1 \leqslant \tau_2$ if for every valuation σ such that $\sigma \models C$, $\sigma \models \tau_1 \leqslant \tau_2$. W.l.o.g., we will consider problems of the form $C \models^? x \leqslant y$.

Remark 8. The problem $C \models^? x \leqslant y$ can be expressed by the first-order sentence $\forall x_1, \ldots, x_n. C \Rightarrow x \leqslant y$, where $x_1, \ldots, x_n$ are the free variables of $C \Rightarrow x \leqslant y$.

3 Decidability and complexity results

In [6], there is a deep study of NS_1, and validity in this theory has been proved to be undecidable. In [3], it is shown that S_1 is decidable when assignments assign to *finite* (non-recursive) terms. In this section, we prove that U_1 is decidable, and we adapt a proof from [2] to give a lower bound on the complexity of entailment in this theory.

3.1 The logic $\mathsf{S}k\mathsf{S}$

We prove the decidability of U_1 via a reduction to $\mathsf{S}k\mathsf{S}$. $\mathsf{S}k\mathsf{S}$ is the monadic second-order logic of the infinite k-ary tree (see [7] or [1] for an introduction). We will use the following syntax:

$$\begin{aligned} i &::= 1 \mid \cdots \mid k \\ t &::= \epsilon \mid \pi i \\ \phi &::= \pi = t \mid \pi \in X \mid \forall \pi. \phi_1 \mid \exists X. \phi_1 \mid \phi_1 \wedge \phi_2 \mid \neg \phi_1 \mid \cdots \end{aligned}$$

First-order variables represent addresses and are denoted by variations of π, second-order variables represent (possibly infinite) sets of addresses and are denoted by variations of X or L. Rabin proved the following result:

Theorem 1 (Rabin's Tree Theorem [5]). *The validity of a sentence in $\mathsf{S}k\mathsf{S}$ is decidable.*

3.2 Decidability of U_1

In table 1, we encode a U_1-sentence ϕ into an $\mathsf{S}k\mathsf{S}$ one $[\![\phi]\!]$. We consider the signature $\Sigma = \{f_1, \ldots, f_n\}$, where function symbols have arity k and are covariant in all arguments. A variable x is encoded into n second-order variables $L_{x=f_1}, \ldots, L_{x=f_n}$, where $L_{x=g}$ denotes the *inverted* positions of nodes of x labeled with g. Notice that inversion is essential to encode $x = g(x_1, \ldots, x_k)$. With these explanations, we can prove the following lemma:

Lemma 2. *ϕ is valid if and only if $[\![\phi]\!]$ is valid.*

Proof. If ϕ is a U_1-formula, the free variables of $[\![\phi]\!]$ are $L_{x=f_1}, \ldots, L_{x=f_n}$, where x ranges over the free variables of ϕ. If σ (ρ resp.) is an assignment on free variables of ϕ (resp. $[\![\phi]\!]$), we define an assignment $r(\sigma)$ ($s(\rho)$ resp.) by

$$\begin{aligned} r(\sigma)(L_{x=g}) &\stackrel{\text{def}}{=} \{\widetilde{\pi} \mid \sigma(x)(\pi) = g\} \\ s(\rho)(x)(\pi) &\stackrel{\text{def}}{=} g \quad \text{such that} \quad \widetilde{\pi} \in \rho\,(L_{x=g}) \end{aligned}$$

where $\widetilde{\pi}$ denotes π reversed. We also define

$$\mathsf{UTerms}(\phi) \stackrel{\text{def}}{=} \bigwedge_{x \text{ free in } \phi} \mathsf{UTerm}(x)$$

$$\mathsf{Partition}(X_1,\ldots,X_n) \stackrel{\text{def}}{=} \forall\pi.\ \bigvee_{j=1}^{n}\left(\pi\in X_j \wedge \bigwedge_{\substack{l=1\\ l\neq j}}^{n}\pi\notin X_l\right)$$

$$\mathsf{UTerm}(x) \stackrel{\text{def}}{=} \mathsf{Partition}(L_{x=f_1},\ldots,L_{x=f_n})$$

$$[\![x\leqslant y]\!] \stackrel{\text{def}}{=} \forall\pi.\ \bigvee_{\substack{g,h\in\Sigma\\ g\leqslant_\Sigma h}}(\pi\in L_{x=g}\wedge\pi\in L_{y=h})$$

$$[\![x=g(x_1,\ldots,x_k)]\!] \stackrel{\text{def}}{=} \epsilon\in L_{x=g}\wedge\bigwedge_{i=1}^{k}\forall\pi.\ \bigvee_{h\in\Sigma}(\pi\in L_{x_i=h}\wedge\pi i\in L_{x=h})$$

$$[\![\exists x.\phi]\!] \stackrel{\text{def}}{=} \exists L_{x=f_1}\ldots\exists L_{x=f_n}.\mathsf{UTerm}(x)\wedge[\![\phi]\!]$$

$$[\![\phi_1\wedge\phi_2]\!] \stackrel{\text{def}}{=} [\![\phi_1]\!]\wedge[\![\phi_2]\!]$$

$$[\![\neg\phi]\!] \stackrel{\text{def}}{=} \neg\,[\![\phi]\!]$$

Table 1: U_1 into $\mathsf{S}k\mathsf{S}$

We now prove by induction on ϕ that if $\sigma\models\phi$, then $r(\sigma)\models[\![\phi]\!]$, and conversely, if $\rho\models[\![\phi]\!]\wedge\mathsf{UTerms}(\phi)$ then $s(\rho)\models\phi$ (notice that $s(\rho)$ is always well defined in this case):

- if $\phi\equiv x=g(x_1,\ldots,x_k)$ or $\phi\equiv x\leqslant y$, it is a direct consequence of the definitions;
- if $\phi\equiv\exists x.\phi_1$, let σ' be an extension of σ to x such that $\sigma'\models\phi_1$. Clearly, $r(\sigma')\models\mathsf{UTerm}(x)$, and by induction, $r(\sigma')\models[\![\phi_1]\!]$, hence $r(\sigma)\models[\![\phi]\!]$. Conversely, let ρ' be an extension of ρ to $L_{x=f_1},\ldots,L_{x=f_n}$ such that $\rho'\models\mathsf{UTerm}(x)\wedge[\![\phi_1]\!]\wedge\mathsf{UTerms}(\exists x.\phi_1)$. Notice that $\mathsf{UTerm}(x)$ and $\mathsf{UTerms}(\exists x.\phi_1)$ cover all free variables of ϕ_1 so that we can apply the inductive hypothesis to get $s(\rho')\models\phi_1$. Therefore, $\rho\models\phi$;
- if $\phi\equiv\phi_1\wedge\phi_2$, then $\sigma\models\phi_1$ and $\sigma\models\phi_2$, then $r(\sigma)\models[\![\phi_1]\!]$ and $r(\sigma)\models[\![\phi_2]\!]$ (by induction), then $r(\sigma)\models[\![\phi_1\wedge\phi_2]\!]$. The converse is similar;
- if $\phi\equiv\neg\phi_1$, then $\sigma\not\models\phi_1$. We proceed by contradiction: if $r(\sigma)\models[\![\phi_1]\!]$, then $s(r(\sigma))\models\phi_1$ (by induction), but clearly $s(r(\sigma))=\sigma$. Therefore, $r(\sigma)\not\models[\![\phi_1]\!]$, and $r(\sigma)\models[\![\phi]\!]$. The converse is similar.

This ends the proof of lemma 2 since $\mathsf{UTerms}(\phi)$ is True when ϕ is closed. □

Remark 9. Because of inversion, this embedding cannot be extended to formulas involving possibly finite trees.

Remark 10. We can prove the same result with contravariant constructors as well by adding a quantified second-order variable representing the set of all covariant positions.

The following is a corollary of this lemma and theorem 1:

Theorem 3. *U_1 is decidable.*

3.3 Complexity of uniform entailment

As in the non-structural case, uniform subtype entailment is PSPACE-hard. The proof is the same as Henglein and Rehof's [2], by reduction from the CLOSED-UNIV problem — *Given a prefix-closed NFA $\mathcal{A}$ (i.e. all states are final) over a nontrivial alphabet Σ, does $L(\mathcal{A}) = \Sigma^*$ hold?* — which is known to be PSPACE-complete.

From a prefix-closed NFA over $\{1,2\}$, we build a constraint over the uniform signature $\Sigma = \{\bot, \top, f\}$, where constructors have arity 2 and are covariant in both arguments. Let $\mathcal{A} = (Q, \{1,2\}, \delta, q_0, Q)$ be a prefix-closed automaton on $\{1,2\}$. We assume that $Q = \{q_0, \ldots, q_{|Q|-1}\}$ and that there are n simple transitions in δ ordered in some arbitrary, fixed sequence. Let $\alpha_0, \ldots, \alpha_{|Q|-1}, \beta, \delta_1, \ldots, \delta_n$ be distinct variables. We define C_k by:

- if the k-th simple transition is $q_i \mapsto^1 q_j$, then $C_k \stackrel{\text{def}}{=} \alpha_i \leqslant f(\alpha_j, \delta_k)$;
- if the k-th simple transition is $q_i \mapsto^2 q_j$, then $C_k \stackrel{\text{def}}{=} \alpha_i \leqslant f(\delta_k, \alpha_j)$;
- if the k-th simple transition is $q_i \mapsto^\epsilon q_j$, then $C_k \stackrel{\text{def}}{=} \alpha_i \leqslant \alpha_j$.

We define $C_\mathcal{A}$ by:

$$C_\mathcal{A} \stackrel{\text{def}}{=} \beta = f(\beta, \beta) \wedge \bigwedge_{k=1}^{n} C_k$$

Clearly, $C_\mathcal{A}$ can be constructed from $\mathcal{A}$ in logarithmic space. Moreover, the following lemma shows that this is a correct reduction:

Lemma 4. *$L(\mathcal{A}) = \{1,2\}^*$ if and only if $C_\mathcal{A} \models \alpha_0 \leqslant \beta$.*

Proof.

- $\Leftarrow$ We suppose $L(\mathcal{A}) \neq \{1,2\}^*$ and prove $C_\mathcal{A} \not\models \alpha_0 \leqslant \beta$. Let $w \in \{1,2\}^*$ such that $w \notin L(\mathcal{A})$. Since we can assume w.l.o.g. that $\mathcal{A}$ has at least one state, and since $\mathcal{A}$ is prefix-closed, $\epsilon \in L(\mathcal{A})$, so $L(\mathcal{A}) \neq \emptyset$. Then there exists a prefix w' of w of maximal length such that $w' \in L(\mathcal{A})$; we can assume that w can be written as $w = w'1w''$ — the case $w = w'2w''$ is similar. Since $\mathcal{A}$ is prefix-closed, for any state q_k such that $q_0 \mapsto^{w'} q_k$, there is no state q_l such that $q_k \mapsto^1 q_l$. Hence, for any k such that $q_0 \mapsto^{w'} q_k$, the constructed upper bounds on α_k in the transitive closure of $C_\mathcal{A}$ are — by construction of $C_\mathcal{A}$ — of the form $\alpha_k \leqslant f(\delta_m, \alpha_s)$, where δ_m is unbounded in $C_\mathcal{A}$. Therefore[1], the largest solution[2] $\sigma^\vee$ to $C_\mathcal{A}$ satisfies $\sigma^\vee(\alpha_0)(w'1) = \top$, thereby showing that $C_\mathcal{A} \not\models \alpha_0 \leqslant \beta$.
- $\Rightarrow$ We suppose $L(\mathcal{A}) = \{1,2\}^*$ and prove $C_\mathcal{A} \models \alpha_0 \leqslant \beta$. Let $w \in \{1,2\}^*$. Then $w1 \in L(\mathcal{A})$, and there is a transition $q_0 \mapsto^w q_j \mapsto^1 q_k$ for some q_j, q_k. By construction of $C_\mathcal{A}$, there is $\alpha_j \leqslant f(\alpha_k, \delta_m)$ in the transitive closure of $C_\mathcal{A}$ for some δ_m. Therefore[2], for any solution σ to $C_\mathcal{A}$, $\sigma(\alpha_0)(w) \leqslant_\Sigma f$, thereby showing $C_\mathcal{A} \models \alpha_0 \leqslant \beta$. □

We have proved the following:

Theorem 5. *Uniform subtype entailment is PSPACE-hard.*

[1] these parts are the only differences with Henglein and Rehof's proof
[2] we will not discuss its existence and uniqueness here

4 Non-structural entailment

We consider the signature $\Sigma = \{\bot, \top, f\}$, where f has arity 2 and is covariant in both arguments. The decidability of non-structural subtype entailment with such a simple signature is still an open problem. In this section, we give a sound (but incomplete) approximation to this problem.

4.1 Preliminaries

We define some shorthands for transforming non-structural objects into uniform ones (we suppose that $x_\bot$ are $x_\top$ are reserved variable names):

- if τ is an NS-term, τ^{U} is τ where each occurrence of $\bot$ (resp. $\top$) is replaced by $\bot^\infty$ (resp. $\top^\infty$). Notice that τ and τ^{U} agree on D_τ;
- if C is an NS-constraint,

$$C^{\mathsf{U}} \stackrel{\text{def}}{=} C' \wedge x_\bot = \bot(x_\bot, x_\bot) \wedge x_\top = \top(x_\top, x_\top)$$

  where C' is C where each occurrence of $\bot$ (resp. $\top$) is replaced by $x_\bot$ (resp. $x_\top$);
- if σ assigns variables to NS-terms,

$$\sigma^{\mathsf{U}}(y) \stackrel{\text{def}}{=} \begin{cases} \bot^\infty & \text{if } y = x_\bot \\ \top^\infty & \text{if } y = x_\top \\ \sigma(y)^{\mathsf{U}} & \text{otherwise} \end{cases}$$

and, conversely, we define shorthands for transforming uniform objects into non-structural ones:

- if τ is a U-term, τ^{NS} is τ where any child subtree of a node labeled $\bot$ or $\top$ is removed. Notice that τ and τ^{NS} agree on $D_{\tau^{\mathsf{NS}}}$, and $\left(\tau^{\mathsf{U}}\right)^{\mathsf{NS}} = \tau$;
- if σ assigns variables to U-terms, $\sigma^{\mathsf{NS}}(y) \stackrel{\text{def}}{=} \sigma(y)^{\mathsf{NS}}$.

Then, we prove two lemmas:

Lemma 6.

1. *If τ_1 and τ_2 are two U-terms such that $\tau_1 \leqslant_{\mathsf{U}} \tau_2$, then $\tau_1^{\mathsf{NS}} \leqslant_{\mathsf{NS}} \tau_2^{\mathsf{NS}}$.*
2. *If τ_1 and τ_2 are two NS-terms such that $\tau_1 \leqslant_{\mathsf{NS}} \tau_2$, then $\tau_1^{\mathsf{U}} \leqslant_{\mathsf{U}} \tau_2^{\mathsf{U}}$.*

Proof.

1. We have, for all $\pi \in \{1, \ldots, k\}^*$, $\tau_1(\pi) \leqslant_\Sigma \tau_2(\pi)$, and in particular for $\pi \in D_{\tau_1^{\mathsf{NS}}} \cap D_{\tau_2^{\mathsf{NS}}}$.
2. We prove the second part by contradiction. Assume $\tau_1^{\mathsf{U}} \not\leqslant_{\mathsf{U}} \tau_2^{\mathsf{U}}$. Let π be a minimal path such that $\tau_1^{\mathsf{U}}(\pi) \not\leqslant_\Sigma \tau_2^{\mathsf{U}}(\pi)$. Clearly, $\pi \neq \epsilon$, so let us decompose π into $\pi' i$, with $i \in \{1, \ldots, k\}$. By minimality of π, we have $\tau_1^{\mathsf{U}}(\pi') \leqslant_\Sigma \tau_2^{\mathsf{U}}(\pi')$. Let us check that each case leads to a contradiction:
   - if $\pi \in D_{\tau_1} \cap D_{\tau_2}$, then we get a direct contradiction with $\tau_1 \leqslant_{\mathsf{NS}} \tau_2$;

- if $\pi \in D_{\tau_1} \setminus D_{\tau_2}$, then $\tau_2^{\mathsf{U}}(\pi')$ must be $\top$, and we get a contradiction with $\tau_1^{\mathsf{U}}(\pi) \not\leqslant_\Sigma \tau_2^{\mathsf{U}}(\pi)$;
- the case $\pi \in D_{\tau_2} \setminus D_{\tau_1}$ is symmetric;
- if $\pi \notin D_{\tau_1} \cup D_{\tau_2}$, then $\tau_1^{\mathsf{U}}(\pi')$ and $\tau_2^{\mathsf{U}}(\pi')$ must be constants, and we get another contradiction with $\tau_1^{\mathsf{U}}(\pi) \not\leqslant_\Sigma \tau_2^{\mathsf{U}}(\pi)$. □

Lemma 7. *Let C be an NS-constraint.*

1. *If $\sigma \models_{NS} C$, then $\sigma^U \models_U C^U$.*
2. *If $\sigma \models_U C^U$, then $\sigma^{NS} \models_{NS} C$.*

Proof. We prove the first part by induction on C:

- if $C \equiv x = t$, where $t \in \{\bot, \top, f(x_1, x_2)\}$, it is clear;
- if $C \equiv x \leqslant y$, then $\sigma(x) \leqslant_{\mathsf{NS}} \sigma(y)$, and lemma 6 yields the result;
- if $C \equiv C_1 \wedge C_2$, then $\sigma \models_{\mathsf{NS}} C_1$ and $\sigma \models_{\mathsf{NS}} C_2$, and we get the result by induction.

We would prove the second part in a very similar way. □

Remark 11. This result cannot be extended to all formulas because of negations. For example, if $\sigma \stackrel{\text{def}}{=} [x := \bot(f^\infty, f^\infty),\ y := \bot^\infty]$ and $\phi \stackrel{\text{def}}{=} \neg(x \leqslant y)$, we have $\sigma \models_{\mathsf{U}} \phi^{\mathsf{U}}$, but $\sigma^{\mathsf{NS}} \not\models_{\mathsf{NS}} \phi$.

4.2 An approximation to non-structural entailment

Proposition 8. *Let C be an NS-constraint. If $C^U \models_U x \leqslant y$, then $C \models_{NS} x \leqslant y$.*

Proof. If $\sigma \models_{\mathsf{NS}} C$, then $\sigma^{\mathsf{U}} \models_{\mathsf{U}} C^{\mathsf{U}}$ (by lemma 7), then $\sigma(x)^{\mathsf{U}} \leqslant_{\mathsf{U}} \sigma(y)^{\mathsf{U}}$ (by hypothesis), then $\sigma(x) \leqslant_{\mathsf{NS}} \sigma(y)$ (by lemma 6). □

Remark 12. Unfortunately, the converse is false. For example, let us consider the constraint $C \stackrel{\text{def}}{=} x \leqslant f(y,y) \wedge f(z,z) \leqslant y \wedge f(u,u) \leqslant z \wedge u = \top$. The entailment $C \models_{\mathsf{NS}} x \leqslant y$ holds, but $C^{\mathsf{U}} \models_{\mathsf{U}} x \leqslant y$ does not: consider $\sigma = [x := f(y,y),\ y := \top(z,z),\ z := f(u,u),\ u := \top^\infty]$.

Remark 13. We can also prove a similar proposition with a contravariant constructor.

4.3 Comparison with a previously known algorithm

In this section, we compare the algorithm suggested by proposition 8 with the *primitive subtyping* algorithm described by Trifonov and Smith in [8]. We prove that our algorithm is (at least) more powerful than theirs. We still do not know whether it is strictly more powerful or gives exactly the same results.

Let us first recall some definitions:

$(\bot) \quad K \vdash \bot \leqslant \tau$	$(\top) \quad K \vdash \tau \leqslant \top$
$(=) \quad K \vdash x \leqslant x, \quad \text{if } x \in D_K$	$(f) \quad \dfrac{K \vdash \tau_1 \leqslant \tau_1' \quad K \vdash \tau_2 \leqslant \tau_2'}{K \vdash f(\tau_1, \tau_2) \leqslant f(\tau_1', \tau_2')}$
$(\uparrow) \quad \dfrac{(K, x \leqslant \tau, x \leqslant \tau') \vdash \tau \leqslant \tau'}{(K, x \leqslant \tau) \vdash x \leqslant \tau'}$ if $(K, x \leqslant \tau, x \leqslant \tau')$ is contractive	$(\downarrow) \quad \dfrac{(K, x \geqslant \tau, x \geqslant \tau') \vdash \tau' \leqslant \tau}{(K, x \geqslant \tau) \vdash \tau' \leqslant x}$ if $(K, x \geqslant \tau, x \geqslant \tau')$ is contractive

Table 2: Rules for primitive subtyping

Definition 1. A *constraint map* is a finite map K assigning to each type variable x in its domain two sets, the upper and lower bounds on x respectively. We use the more intuitive notations $x \leqslant \tau \in K$ and $x \geqslant \tau \in K$ for $\tau \in \pi_1(K(x))$ and $\tau \in \pi_2(K(x))$ respectively (π_i is the i-th projection). Notice that K is not required to be antisymmetric; e.g. $x \leqslant y \in K$ is different from $y \geqslant x \in K$.

A constraint map is just a practical way of representing a constraint:

Definition 2. The *kernel* $Ker(C)$ of a constraint C (seen as a set of atomic constraints) is the constraint map defined by the set of constraints

$$\{\tau \leqslant \tau' \in Cl(C) \mid \tau \text{ or } \tau' \text{ is a variable}\}$$

where a constraint of the form $x \leqslant y$ sets the appropriate bounds on both variables, and $Cl(C)$ is the transitivity and decomposition closure of C.

Definition 3. A constraint map K is *contractive* if there is no $\{x_1, \ldots, x_n\} \subseteq D_K$ such that $x_n = x_1$ and $x_i \leqslant x_{i+1} \in K$ (resp. $x_i \geqslant x_{i+1}$) for each $i \in \{1, \ldots, n-1\}$.

Definition 4. A constraint map K is *canonical* if

- K assigns exactly one upper and one lower constructed bound — the *canonical bounds* — to each variable in its domain;
- if $x \leqslant y \in K$ and $y \leqslant z \in K$, then $x \leqslant z \in K$, and similarly for the lower bounds;
- if $(x \leqslant y, x \leqslant \tau, y \leqslant \tau') \subseteq K$, where τ and τ' are constructed, then $K \vdash \tau \leqslant \tau'$, and similarly for the lower bounds.

For a consistent constraint C, $Ker(C)$ and C are equivalent. Trifonov and Smith also give an algorithm to compute a canonical constraint map $Can(C)$ from a constraint C which is equivalent to C. Given an entailment judgement $C \models^?_{\mathsf{NS}} \tau \leqslant \tau'$, the idea of the algorithm is to find a proof of $Can(C) \vdash \tau \leqslant \tau'$ using the rules in table 2. Moreover, the relation $K \vdash \tau \leqslant \tau'$ is decidable. For more details, see [8].

Actually, Trifonov and Smith's proof of soundness of their algorithm — which we recall here — suits to show that $\vdash$ is decided by our algorithm.

Lemma 9. *If K is contractive, $K \vdash \tau \leqslant \tau'$ has a proof, and $\sigma \models^k_U K^U$, then:*

1. *if the proof of* $K \vdash \tau \leqslant \tau'$ *has an instance of a rule other than* $(\uparrow)$ *or* $(\downarrow)$ *at its root, then* $\sigma \models_U^{k+1} (\tau \leqslant \tau')^U$*;*

2. *otherwise,* $\sigma \models_U^k (\tau \leqslant \tau')^U$*.*

Proof. By induction on the structure of the proof of $K \vdash \tau \leqslant \tau'$.

1. If the last rule is $(\bot)$, $(\top)$ or $(=)$, it is obvious[3]; if it is (f), it is a direct consequence of the inductive hypothesis.

2. We now consider the case where the last rule is $(\uparrow)$ (the case $(\downarrow)$ is similar). So τ is some type variable, and the antecedent of the rule is of the form $(K, \tau \leqslant \tau') \vdash \tau_1 \leqslant \tau'$, where $\tau \leqslant \tau_1 \in K$. If one or both of τ_1 and τ' are variable, the last rule in the proof of this sequent may be $(\uparrow)$ or $(\downarrow)$ again, so the proof of $K \vdash \tau \leqslant \tau'$ ends in a chain of one or more instances of these two rules.

   Let $\langle \tau_0, \ldots, \tau_n \rangle$ and $\langle \tau'_0, \ldots, \tau'_m \rangle$ be respectively the sequences of left and right hand sides of the conclusions of these instances, from the root. Here, $\tau_0 = \tau$, n is equal of the number of instances of $(\uparrow)$, and τ_i is a variable if $i \in \{0, \ldots, n-1\}$. Similarly, $\tau'_0 = \tau'$, m is equal of the number of instances of $(\downarrow)$, and τ'_i is a variable if $i \in \{0, \ldots, m-1\}$. At the other end of the chain, there is a proof of some $(K, K') \vdash \tau_n \leqslant \tau'_m$ ending in a rule other than $(\uparrow)$ or $(\downarrow)$. Notice that K' consists exactly of the constraints in the conclusions of rules in this chain.

   Moreover, one can prove by induction on i that $\tau_i \leqslant \tau_{i+1} \in K$ for each $i \in \{0, \ldots, n-1\}$. Indeed, for $i \in \{0, \ldots, n-2\}$, the $(i+1)$-th instance of $(\uparrow)$ (from the root) is of the form:

$$(\uparrow) \quad \frac{(K, K'_i, \tau_i \leqslant \tau'_j) \vdash \tau_{i+1} \leqslant \tau'_j}{(K, K'_i) \vdash \tau_i \leqslant \tau'_j}$$

   for some j and $K'_i \subseteq K'$, where $\tau_i \leqslant \tau_{i+1} \in (K, K'_i)$. If $\tau_i \leqslant \tau_{i+1} \notin K$, then $\tau_i \leqslant \tau_{i+1} \in K'$, so $\tau_i \leqslant \tau_{i+1}$ is the conclusion of the $(l+1)$-th instance of $(\uparrow)$, for some $l < i$. But this implies $\tau_i = \tau_l$, and since — by induction — $\{\tau_l \leqslant \tau_{l+1}, \ldots, \tau_{i-1} \leqslant \tau_i\} \subseteq K$, K is not contractive, which contradicts the assumptions. Therefore, $\tau_i \leqslant \tau_{i+1} \in K$. Similarly, one can prove that $\tau'_j \geqslant \tau'_{j+1} \in K$ for each $j \in \{0, \ldots, m-1\}$.

   We now establish that $\sigma \models_{\mathsf{U}}^l K'^{\mathsf{U}}$ for each $l \in \{0, \ldots, k\}$ — and therefore $\sigma \models_{\mathsf{U}}^k (\tau \leqslant \tau')^{\mathsf{U}}$ — by induction on l:

   - any assignment 0-satisfies any constraint;
   - if $\sigma \models_{\mathsf{U}}^l K'^{\mathsf{U}}$, where $l < k$, then $\sigma \models_{\mathsf{U}}^l (K^{\mathsf{U}}, K'^{\mathsf{U}})$, and — by (the first) inductive hypothesis, and since the proof of $(K, K') \vdash \tau_n \leqslant \tau'_m$ has a rule other than $(\uparrow)$ or $(\downarrow)$ at its root — $\sigma \models_{\mathsf{U}}^{l+1} (\tau_n \leqslant \tau'_m)^{\mathsf{U}}$. Moreover, all the constraints in K' are of the form $\tau_i \leqslant \tau'_j$ or $\tau'_j \geqslant \tau_i$. Since $l + 1 \leqslant k$, we have $\sigma \models_{\mathsf{U}}^{l+1} K^{\mathsf{U}}$, so the following holds in the uniform model induced by σ truncated at level $l + 1$:

$$\tau_0 \leqslant \cdots \leqslant \tau_n \leqslant \tau'_m \leqslant \cdots \leqslant \tau'_0$$

     which covers all constraints in K'^{U}. □

[3] however, the "obviousness" is not exactly the same as in Trifonov and Smith's proof

Proposition 10. *If K is contractive and $K \vdash \tau \leqslant \tau'$, then $K^{\mathsf{U}} \models_{\mathsf{U}} (\tau \leqslant \tau')^{\mathsf{U}}$.*

Proof. If $\sigma \models_{\mathsf{U}} K^{\mathsf{U}}$, then for all k, $\sigma \models^{k}_{\mathsf{U}} K^{\mathsf{U}}$, then at least $\sigma \models^{k}_{\mathsf{U}} (\tau \leqslant \tau')^{\mathsf{U}}$ (by lemma 9), then $\sigma \models_{\mathsf{U}} (\tau \leqslant \tau')^{\mathsf{U}}$. □

Remark 14. When $K = (x \leqslant f(x, \bot), x \leqslant f(\bot, x))$, $K \vdash x \leqslant f(\bot, \bot)$ does not hold whereas $K \models x \leqslant f(\bot, \bot)$ does. In the uniform translation, it also holds. But [8] gives a "canonicalization" algorithm which seems to cover all the cases covered by the uniform approach. However, we still do not know whether there exists a judgement $K \models^{?} x \leqslant y$ which is true, decided by our algorithm and not by Trifonov and Smith's.

5 Subtyping constrained types

Like entailment, subtyping constrained types is still an open problem in the non-structural theory. The problem

$$\alpha_1 \backslash C_1 \leqslant^{?}_{\mathsf{NS}} \alpha_2 \backslash C_2$$

— where both types use disjoint sets of variables — can be expressed by the first-order sentence

$$\forall \alpha_2, x_1, \ldots, x_n. C_2 \Rightarrow \exists \alpha_1, y_1, \ldots, y_m. C_1 \Rightarrow \alpha_1 \leqslant \alpha_2$$

where $\alpha_2, x_1, \ldots, x_n$ are the free variables of C_2 and $\alpha_1, y_1, \ldots, y_m$ the ones of C_1.

The main purpose of [8] is to give an approximation to this problem. We can also use the previous ideas to give another approximation algorithm:

Proposition 11. *Let $x_1 \backslash C_1$ and $x_2 \backslash C_2$ be two constrained types in the non-structural theory. If $x_1 \backslash C_1^{U} \leqslant_{U} x_2 \backslash C_2^{U}$, then $x_1 \backslash C_1 \leqslant_{\mathsf{NS}} x_2 \backslash C_2$.*

Proof. If $\sigma_2 \models_{\mathsf{NS}} C_2$, then $\sigma_2^{\mathsf{U}} \models_{\mathsf{U}} C_2^{\mathsf{U}}$ (by lemma 7), then there exists σ_1' such that $\sigma_1' \models_{\mathsf{U}} C_1^{\mathsf{U}}$ and $\sigma_1'(x_1) \leqslant_{\mathsf{U}} \sigma_2(x_2)^{\mathsf{U}}$ (by hypothesis). Let σ_1 be $(\sigma_1')^{\mathsf{NS}}$. Therefore, $\sigma_1 \models_{\mathsf{NS}} C_1$ (by lemma 7), and $\sigma_1(x_1) \leqslant_{\mathsf{NS}} \sigma_2(x_2)$ (by lemma 6). □

6 Conclusion

During this internship, we showed the decidability of the first-order theory of uniform subtype constaints, and realized that there is a close link between non-structural and uniform subtyping. We also gave a new approximation algorithm to non-structural subtype entailment and subtyping constrained types. This algorithm is conceptually simple, but the difficulty is hidden behind Rabin's Theorem and we do not have a working implementation so far.

In the end, I am grateful to Zhendong Su for having supervised me. This internship was really enriching scientifically. Davis is mainly a university town, but it is approximately 120 km away from San Francisco and the Silicon Valley, and 220 km away from Lake Tahoe, in the Sierras, which gives the possibility to go to both places in day or week-end trips. Besides, the Sierras are a beautiful place to do hiking, camping, or just see nature.

References

[1] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Application*, chapter Logic, Automata and Relations, pages 86–94. GRAPPA, Université Lille 3, available on: `http://www.grappa.univ-lille3.fr/tata`, October 2002.

[2] Fritz Henglein and Jakob Rehof. Constraint automata and the complexity of recursive subtype entailment. In *ICALP '98: Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, pages 616–627, London, UK, 1998. Springer-Verlag.

[3] Viktor Kuncak and Martin Rinard. Structural subtyping of non-recursive types is decidable. In *Eighteenth Annual IEEE Symposium on Logic in Computer Science*, 2003.

[4] Joachim Niehren, Tim Priesnitz, and Zhendong Su. The complexity of subtype satisfiability over posets. In *14th European Symposium on Programming*, volume 3444 of *LNCS*, pages 357–373. Springer Verlag, April 2005.

[5] Michael O. Rabin. Decidability of second-order theories and automata on infnite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[6] Zhendong Su, Alexander Aiken, Joachim Niehren, Tim Priesnitz, and Ralf Treinen. The first-order theory of subtyping constraints. In *ACM TOPLAS*, December 2004.

[7] Wolfgang Thomas. *Handbook of theoretical computer science (vol. B): formal models and semantics*, chapter Automata on infinite objects, pages 133–191. MIT Press, Cambridge, MA, USA, 1990.

[8] Valery Trifonov and Scott F. Smith. Subtyping constrained types. In *SAS '96: Proceedings of the Third International Symposium on Static Analysis*, pages 349–365, London, UK, 1996. Springer-Verlag.