Subtyping Functional+Nonempty Record Types [⋆]

Sergei Vorobyov

Max-Planck Institut für Informatik
Im Stadtwald, D-66123 Saarbrücken, Germany
(e-mail: `sv@mpi-sb.mpg.de`)

Abstract. *Solving systems of subtype constraints* (or *subtype inequalities*) is in the core of efficient *type reconstruction* in modern object-oriented languages with subtyping and inheritance, two problems known *polynomial time equivalent.* It is important to know how different combinations of type constructors influence the complexity of the problem.
We show the *NP-hardness* of the satisfiability problem for subtype inequalities between object types built by using simultaneously both the functional and the nonempty record type constructors, but without any atomic types and atomic subtyping.
The class of constraints we address is intermediate with respect to known classes. For pure functional types with atomic subtyping of a special non-lattice (*crown*) form solving subtype constraints is PSPACE-complete (Tiuryn 1992, Frey 1997). On the other hand, if there are no atomic types and subtyping on them, but the largest $\top$ type is included, then both pure functional and pure record (separately) subtype constraints are *polynomial time solvable* (Kozen, Palsberg & Schwartzbach 1994, Palsberg 1995), which is mainly due to the lattice type structure. We show that combining the functional and nonempty record constructors yields NP-hardness *without any atomic subtyping*, and the same is true for just a single type constant with a nonempty record constructor.

1 Introduction

Subtyping is a fundamental feature of the contemporary object-oriented languages. In this paper we address the inherent computational complexity of the *subtyping constraints satisfaction problem* for object types built by using simultaneously the functional and the nonempty record type constructors, but without any atomic types and subtyping relation on them. The motivation for subtyping record+functional types comes from type reconstruction and type-checking in Object-Oriented Programming, where an object (record) can be emulated by another object that has more refined methods (functions). To handle this phenomenon, the so-called *subsumption* rule with record+functional

[⋆] G. Gottlob, E. Grandjean, K. Seyr (Eds). **CSL'98**: *Annual Conference of the European Association for Computer Science Logic*, Lecture Notes in Computer Science, vol. 1584, pp, 280 – 294, 1999. ©Springer-Verlag, Berlin-Heidelberg, 1999 types is introduced. The primary importance of the problem stems from the fact that satisfiability of systems of subtype inequalities is known to be *polynomial time equivalent* to the *type reconstruction* problem for lambda and object terms (Kozen et al. 1994, Hoang & Mitchell 1995, Palsberg 1995).

The *Satisfiability of Subtype Constraints Problem* (further, the SSCP for short) is defined as follows (J. Tiuryn calls it SSI, *Satisfiability of Subtype Inequalities*, (Tiuryn 1992, Tiuryn 1997)).

SSCP: *Given a finite set of subtype inequalities* $\left\{\sigma_i \leq \tau_i\right\}_{i=1}^{n}$ *between type expressions* σ_i, τ_i *containing free type variables, does there exist a substitution of type expressions for these variables making all inequalities true?* □

The SSCP is obviously parameterized by:

- the choice of type constructors involved in type expressions,
- the choice of subtyping rules,
- presence/absence of atomic types with a subtype relation on them.

The most common features involved in type construction considered in the literature are as follows:

1. the functional type constructor $\sigma \to \tau$,
2. the record type constructor $[l_1 \colon \tau_1 \ldots, l_n \colon \tau]$,
3. atomic types with a subtype relation on them, e.g., $int \leq real$,
4. the largest type $\top$ (also denoted Ω [1]), and the least type $\bot$.

In Section 2 we survey known cases of SSCP for different types built by combinations of the type constructors above, well investigated in the literature. These include *partial types*, *object types*, *functional types with atomic subtyping* and different assumptions on atomic type orderings.

> *In this paper we consider the SSCP for types built by using simultaneously the functional and the* nonempty *record type constructors, but without atomic types and atomic subtyping.*

Remark 1. We *explicitly exclude the empty record* (playing the role of the $\top$ type for record types) from consideration for the following reasons.

Different combinations of type constructors and solvability of corresponding systems of subtype inequalities are investigated in the literature (Aiken, Wimmers & Lakshman 1994, Palsberg & O'Keefe 1995, Palsberg & Smith 1996, Trifonov & Smith 1996, Brandt & Henglein 1997). However, usually the $\top$ (largest, or Ω) and $\bot$ (least) types are introduced in the system from the very beginning. This immediately transforms the type structure into a lattice and makes it too coarse by introducing too many undesired, non-informative, or even 'senseless' solutions, like $\bot \to \top$. On the positive side of introducing $\top$, $\bot$ is the efficiency

[1] We keep both notations $\top$ and Ω to be consistent with cited papers.

of the typechecking algorithms (usually polynomial[2]). But once we are satisfied with efficiency, the next step is to consider a more refined question: "we know this term is typable by *some type*, but is it typable by some '*meaningful*' type?" At this stage one might wish to get rid of $\top$, $\bot$ types and all the types they generate. Another consequence of introducing $\top$, $\bot$ is that they make the system quite insensitive to combinations of different type constructors and make it quite impossible to analyze the relative costs of different features. Note that the introduction of $\top$, $\bot$ (turning types into lattices) is orthogonal to the whole line of research on non-lattice base types subtyping (Tiuryn 1992, Lincoln & Mitchell 1992, Hoang & Mitchell 1995, Pratt & Tiuryn 96, Tiuryn 1997). □

2 Related Work and Contribution of the Paper

We briefly survey the known results on complexity of the SSCP for different type systems in more detail.

Partial Types. (Kozen et al. 1994) consider the so-called *partial types* introduced by (Thatte 1988), built from a single top type Ω by using the single functional type constructor $\to$, i.e., defined by the grammar

$$\tau ::= \Omega \mid \tau_1 \to \tau_2$$

and the subtype relation $\leq$ defined by the rules

$$\begin{array}{rl} \tau \leq \Omega & \text{for any type } \tau, \\ \sigma \to \tau \leq \sigma' \to \tau' & \text{iff } \sigma' \leq \sigma \text{ and } \tau \leq \tau'. \end{array} \tag{1}$$

(The latter is the standard subtyping rule for functional types.)

By using a nice automata-theoretic technique (Kozen et al. 1994) prove that the SSCP for these partial types is solvable in *deterministic time* $O(n^3)$. (O'Keefe & Wand 1992) were the first to show decidability of the problem by giving an exponential algorithm.

Object Types. (Palsberg 1995) considers the so-called *object types* built by using the single record type constructor, i.e., defined by the grammar

$$\tau \ ::= \ [l_1 \colon \tau_1, \ldots, l_n \colon \tau_n] \quad (\text{for } n \geq 0),$$

where the field labels l_i's are drawn from an infinite set. (Note that in case $n = 0$ we get the *empty record* type [ ]. Obviously, $\tau \leq [\,]$ for each type τ. Thus the empty record [ ] plays the role of the largest type Ω or $\top$).

The subtype relation on these object types is defined by

$$\sigma \leq \tau \text{ iff } \Big(\tau \text{ has field } l \colon \rho \Rightarrow \sigma \text{ has field } l \colon \rho\Big). \tag{2}$$

[2] (Tiuryn 1992, Benke 1993, Pratt & Tiuryn 96, Tiuryn 1997) show that the lattice structure is helpful in getting polynomial algorithms.

Note that this subtyping rule is different from a better known rule

$$\sigma \leq \tau \text{ iff } \Big(\tau \text{ has field } l\colon \tau' \;\Rightarrow\; \sigma \text{ has field } l\colon \sigma' \text{ such that } \sigma' \leq \tau'\Big). \qquad (3)$$

(Palsberg 1995) shows, also by using the automata-theoretic techniques similar to (Kozen et al. 1994), that the SSCP for the object types is decidable in deterministic time $O(n^3)$, and proves that the problem is PTIME-complete[3].

Functional Types with Atomic Subtyping. The SSCP for functional types defined starting from a set of *atomic types* with a *given subtype relation* on them extended to the set of all functional types by the standard subtyping rule (1) attracted most attention in the literature (Lincoln & Mitchell 1992, Tiuryn 1992, Pratt & Tiuryn 96, Benke 1993, Hoang & Mitchell 1995, Tiuryn 1997, Frey 1997). When the subtyping relation on atomic types is identity, the whole subtype relation is also identity, and the SSCP becomes the *unification problem for simple types*, known to be PTIME-complete (Dwork, Kanellakis & Mitchell 1984). (Lincoln & Mitchell 1992), improving (Wand & O'Keefe 1989), demonstrated that for some fixed ordering of atomic types the SSCP is NP-hard. (Tiuryn 1992, Pratt & Tiuryn 96) improved it further to PSPACE-hardness for some simple fixed orderings of atomic types called *crowns.* (Lincoln & Mitchell 1992, Tiuryn 1992) gave the NEXPTIME upper bound for the problem. Recently (Frey 1997) improved it to PSPACE. Thus the SSCP for functional types with atomic types is PSPACE-complete, in general. When the subtype relation on atomic types is a disjoint union of lattices (Tiuryn 1992) or a tree-like partial order (Benke 1993), then the SSCP is in PTIME, i.e., becomes tractable (Tiuryn 1992, Pratt & Tiuryn 96).

It is interesting to note that the partial types subtyping of (Kozen et al. 1994) and the object types subtyping of (Palsberg 1995) form lattices[4]. But the precise relation between the PTIME results of (Tiuryn 1992, Benke 1993, Pratt & Tiuryn 96, Tiuryn 1997) on the one hand, and the results of (Kozen et al. 1994, Palsberg 1995) on the other, seemingly, remains unexplored (do they 'imply' each other?).

Contribution of This Paper. The complexity of the SSCP for types built by using *simultaneously* the functional and the nonempty record type constructors (both in presence or absence of atomic types) remained unknown, although type systems combining functions and records are quite natural in object-oriented programming. The papers cited above concentrate either solely on functional or only on record types, and/or immediately introduce the $\top$ and $\bot$ type. The main result of this paper is that

> *Even without any atomic types the SSCP for functional+nonempty record types is NP-hard. The same holds for the types formed by the nonempty record type constructor+a single atomic type constant (or any other type constructor).* $\square$

[3] Reportedly, F. Henglein improved it to $O(n^2)$ by exploiting non-contravariance of record subtyping (J. Palsberg, private communication).

[4] With $glb(\sigma_1 \to \sigma_2, \tau_1 \to \tau_2) = lub(\sigma_1, \tau_1) \to glb(\sigma_2, \tau_2)$, etc.

Thus, the absence of subtyping on atomic types does not lead to tractability, as contrasted to PSPACE/PTIME complexity for functional types with/without atomic subtyping, recall (Tiuryn 1992, Lincoln & Mitchell 1992, Tiuryn 1997).

Moreover, functions+records without atomic types, seemingly, do not allow to model the *crown structures* of (Tiuryn 1992, Pratt & Tiuryn 96), proved to be a useful uniform tool in showing NP and PSPACE lower bounds. Our proofs are not done by reduction from results of (Tiuryn 1992, Pratt & Tiuryn 96), we use a different method.

Our result shows that the automata-based deterministic polynomial time algorithms of (Kozen et al. 1994, Palsberg 1995) for subtyping partial types and object types (separately) cannot be combined to yield a polynomial deterministic algorithm for functional+nonempty record types. The intrinsic interplay between the nonempty record and the functional (or another) constructor adds to the computational complexity of the SSCP (even without atomic subtyping).

The remainder of the paper is organized as follows. In Section 3 we formally introduce the type system, the problem, and state the results. In Sections 4 – 7 we prove the Main Theorem and give some generalizations. Section 8–9 are devoted to discussion, related results, conclusions.

3 Functional and Record Types

Let $V = \{\alpha, \beta, \gamma, \ldots\}$ be an infinite set of *type variables* and $L = \{l_1, \ldots, l_n, \ldots\}$ be a finite or infinite set of *field labels.*

In this paper we consider the following types.

Definition 1 ((Types)). *Define:*

– ***Functional+Nonempty Record Types*** *by the grammar:*

$$\tau \;::=\; V \mid \tau' \to \tau'' \mid [l_1{:}\,\tau_1, \ldots, l_n{:}\,\tau_n], \tag{4}$$

where $n > 0$ and l_i's are different labels from L.

– ***Pure Nonempty Record Types*** *by the grammar:*

$$\tau \;::=\; V \mid [l_1{:}\,\tau_1, \ldots, l_n{:}\,\tau_n], \tag{5}$$

where $n > 0$ and l_i's are different labels from L. □

That is, types are constructed inductively, starting from type variables, by using the functional $\to$ and the nonempty record $[\ldots]$ type constructors. The subexpressions $l_i{:}\,\tau_i$ in the record construction are called *record fields.* Note that in the record construction the set of fields is *always nonempty*, i.e., we *explicitly exclude the empty record.*

Notation. Types will be denoted by Greek letters τ, σ, ρ, etc. To stress the outermost type constructor of a type we will sometimes superscript a type by the corresponding outermost constructor, like $\sigma^{\to}$ or $\tau^{[\,]}$. A substitution θ of types for variables is defined as usual and denoted by $\theta(\tau)$.

Definition 2. Subtype relation $\leq$ *is defined by the following standard rules:*

> **Reflexivity:** $\tau \leq \tau$,
> **Transitivity:** $\sigma \leq \tau$ *and* $\tau \leq \rho$ *imply* $\sigma \leq \rho$,
> **Functional:** $\sigma \to \tau \leq \sigma' \to \tau'$ *iff* $\sigma' \leq \sigma$ *and* $\tau \leq \tau'$,
> **Record:** $\sigma^{[\,]} \leq \tau^{[\,]}$ *iff for every field* $l\!:\!\tau'$ *in* $\tau^{[\,]}$ *there is a field* $l\!:\!\sigma'$ *in* $\sigma^{[\,]}$ *such that* $\sigma' \leq \tau'$.

A subtype judgment $\sigma \leq \tau$ *is* true *iff it is derivable by these rules.* □

Remark 1 *Note that a functional type is never a subtype of a record type, nor vice versa. All provable subtyping judgments are either* $\alpha \leq \alpha$ *for a type variable* α, *or of the form* $\sigma^{\to} \leq \tau^{\to}$, *or of the form* $\sigma^{[\,]} \leq \tau^{[\,]}$, *i.e., the subtyping relation is* strictly structural.

In fact, our lower complexity bound results are independent of the functional subtyping rule, which we adopt only as a standard one. The two essential things, our results really depend on, are: nonemptiness *of the record type constructor and the* strict structuredness *of the subtyping relation mentioned above.* □

Here is the problem we are interested in, both for functional+nonempty record, pure nonempty record types, nonempty object types.

Satisfiability of Subtype Constraints Problem (SSCP).

> *Given a finite system of subtype inequalities* $\left\{ \sigma_i \leq \tau_i \right\}_{i=1}^{n}$, *does there exist a substitution of types for variables occurring in* σ_i*'s,* τ_i*'s making all inequalities simultaneously true?* □

Our main result is as follows.

Main Theorem. *The SSCP is NP-hard for the following types:*

1. *nonempty record types with subtyping rule (3) and at least some other type constructor,*
2. *nonempty record types with subtyping rule (2) and possibly other type constructors,*

provided that:

- *record types are comparable with record types only,*
- *non-record types are comparable with non-record types only,*
- *there are* ≥ 2 *field labels.*

In particular, the SSCP is NP-hard for the following types:

1. *functional + nonempty record types,*
2. *nonempty record constructor + a single type constant,*
3. *nonempty record constructor + another type constructor,*
4. *Palsberg's system of object types with subtyping rule (2) without empty record.*

4 Proof of the Main Theorem

We first prove the claim for functional+nonempty record types, then in Section 5 sketch the modifications necessary for the case of pure nonempty record types with a constant and subtyping rule (2), and in Section 6 give the general pattern of the proof (due to an anonymous referee).

The proof is by reduction from the well-known NP-complete problem:

SATISFIABILITY. *Given a Boolean formula in Conjunctive Normal Form (CNF), does there exist an assignment of truth values to variables making the formula true?* □

We therefore proceed to representing truth values, clauses, and satisfiability in terms of types and subtyping.

4.1 Truth Values

Fix a type variable γ. Define the *truth values* $\mathbf{t}$ (true) and $\mathbf{f}$ (false) as types

$$\mathbf{t} \equiv_{df} [1{:}\,\gamma], \quad \mathbf{f} \equiv_{df} \gamma \to \gamma, \tag{6}$$

where 1 is an arbitrary label from L.

Clearly, neither $\mathbf{f} \leq \mathbf{t}$, nor $\mathbf{t} \leq \mathbf{f}$, because one is functional and the other is record (recall that the subtyping is strictly structural; see Remark 1). Note also that $\mathbf{t}$, $\mathbf{f}$ do not have common supertypes. This is because the empty record [ ] is excluded from consideration by Definition 1. It seems tempting to get rid of the functional types altogether by defining $\mathbf{f} \equiv_{df} [2{:}\,\gamma]$, which seems as good as $\mathbf{f} \equiv_{df} \gamma \to \gamma$, however, in Section 6 we show that that this does not work.

4.2 Representing Clauses

A *clause* is a disjunction of literals. A *literal* is either a propositional variable, or a negation of a propositional variable. Without loss of generality we assume that a clause never contains complementary pairs of literals. Propositional variables $A_1, \ldots, A_k$ are represented by labels $1, \ldots, k$. A clause

$$C \equiv A_{i_1} \vee \ldots \vee A_{i_m} \vee \neg A_{j_1} \vee \ldots \vee \neg A_{j_n},$$

where $\{i_1, \ldots, i_m\} \cap \{j_1, \ldots, j_n\} = \emptyset$ and $\{i_1, \ldots, i_m\} \cup \{j_1, \ldots, j_n\} \subseteq \{1, \ldots, k\}$, is *represented as a type* (it is essential here that a clause does not contain complementary literals)

$$C^* \equiv_{df} [\, i_1{:}\,\mathbf{t}, \ldots, i_m{:}\,\mathbf{t},\; j_1{:}\,\mathbf{f}, \ldots, j_n{:}\,\mathbf{f}\,].$$

The following proposition relates satisfiability of clauses and subtype judgments.

Proposition 1. *A truth assignment $\nu : \{A_1, \ldots, A_k\} \perp\!\!\rightarrow \{\boldsymbol{t}, \boldsymbol{f}\}$ satisfies a clause C if and only if $i\colon \nu(A_i)$ occurs in C^* for some $i \in \{1, \ldots, k\}$.*

Proof. ν satisfies C iff for some propositional variable A_i one has:

- either $\nu(A_i) = \mathbf{t}$ and $C = \ldots \vee A_i \vee \ldots$; by definition of C^*, $C = \ldots \vee A_i \vee \ldots$ iff $i\colon \mathbf{t}$ occurs in C^*,
- or $\nu(A_i) = \mathbf{f}$ and $C = \ldots \vee \neg A_i \vee \ldots$; by definition of C^*, $C = \ldots \vee \neg A_i \vee \ldots$ iff $i\colon \mathbf{f}$ occurs in C^*. □

4.3 Satisfiability of a Propositional Formula

Definition 3. *The* translation *of a propositional formula in CNF, i.e., a conjunction of clauses (containing no complementary pairs of literals)*

$$\Phi \equiv \bigwedge_{i=1}^{l} C_i$$

is defined as a set of subtyping judgments

$$\Phi^* \equiv_{df} \left\{ C_i^* \leq \beta_i,\ \alpha \leq \beta_i \right\}_{i=1}^{l}, \tag{7}$$

where α, β_i are fresh pairwise distinct type variables.

A solution *to Φ^* is a substitution of types for free type variables making all subtyping judgments in Φ^* true.* □

The main technical result we need to prove the Main Theorem is as follows.

Lemma 1. *Φ is satisfiable if and only if Φ^* has a solution.*

Proof of ($\Rightarrow$)**.** Let an assignment $\nu(A_j) = v_j$ of truth values $v_j \in \{\mathbf{t}, \mathbf{f}\}$ ($1 \leq j \leq k$) satisfy Φ. Then, by record subtyping rule (3), the substitution

$$\begin{aligned} \alpha &\leftarrow [1\colon v_1, \ldots, j\colon v_j, \ldots, k\colon v_k], \\ \beta_i &\leftarrow [j\colon v_j \mid \nu(A_j) = v_j \text{ and } j\colon v_j \in C_i^*] \end{aligned}$$

is a solution to Φ^*. Note that β_i's are substituted by *nonempty* records. □

Proof of ($\Leftarrow$)**.** Suppose, a type substitution θ is a solution to Φ^*. Construct the truth assignment ν by defining for every $j = 1, \ldots, k$

- $\nu(A_j) = \mathbf{t}$ if $j\colon \psi$ occurs in $\theta(\alpha)$ for some *record* type ψ, and
- $\nu(A_j) = \mathbf{f}$ otherwise.

We claim that this ν satisfies $\Phi \equiv \wedge_{i=1}^{l} C_i$, i.e., ν satisfies the clause C_i for every $i = 1, \ldots, l$. Since θ is a solution to Φ^*, we have $\theta(C_i^*) \leq \theta(\beta_i) \geq \theta(\alpha)$.

By definition, C_i^* is a record type, hence $\theta(C_i^*)$ and $\theta(\beta_i)$ are also record types. Let $\theta(\beta_i) = [\ldots, j\colon \tau, \ldots]$ (nonempty!), where the type τ is:

1. either a record type (this happens when $C_i^* = [\ldots, j\colon \mathbf{t}, \ldots]$, consequently, when $C_i = \ldots \vee A_j \vee \ldots$),
2. or a functional type (this happens when $C_i^* = [\ldots, j\colon \mathbf{f}, \ldots]$, consequently, when $C_i = \ldots \vee \neg A_j \vee \ldots$).

Since $\theta(\beta_i) \geq \theta(\alpha)$, we have $\theta(\alpha) = [\ldots, j\colon \sigma \ldots]$, where the type σ is, respectively (corresponding to the two cases above):

1. a record type; in this case, by definition, $\nu(A_j) = \mathbf{t}$, consequently, $C_i = \ldots \vee A_j \vee \ldots$ is true in the truth assignment ν, or
2. a functional type; in this case, by definition, $\nu(A_j) = \mathbf{f}$, consequently, $C_i = \ldots \vee \neg A_j \vee \ldots$ is true in the truth assignment ν.

Thus, the assignment ν satisfies C_i for all $1 \leq i \leq l$, and the proof is finished. □

Since the solvability of a system of subtyping judgments is equivalent to the solvability of a single judgment (by using records), and the translation * of propositional formulas into systems of subtype constraints (7) is obviously polynomial time computable, we thus proved the first claim of the Main Theorem.

Remark 2. The presented proof makes transparent the following distinction between solving subtype inequalities *with and without* the empty record. If it is allowed, we may always guess this empty record as a supertype of any record type (leads to PTIME). If it is forbidden, we should make a nondeterministic choice between all possible nonempty subrecords (leads to NP).

Remark 3 ((Just Two Labels Suffice)). The proof above uses the number of record field labels equal to the number of propositions in an input propositional formula. In fact, just two labels, say 0 and 1, suffice. Instead of flat records $[1.v_1, \ldots, k.v_k]$ used in the proof we could have used *nested records* like

$$[0\colon[0\colon[0\colon u_{000}, 1\colon u_{001}], 1\colon[0\colon u_{010}, 1\colon u_{011}]], 1\colon[0\colon[0\colon u_{100}, 1\colon u_{101}], 1\colon[0\colon u_{110}, 1\colon u_{111}]]],$$
$$[0\colon[0\colon[0\colon v_{000}, 1\colon v_{001}], 1\colon[0\colon v_{010}, 1\colon v_{011}]], 1\colon[0\colon[0\colon v_{100}, 1\colon v_{101}], 1\colon[0\colon v_{110}, 1\colon v_{111}]]].$$

It is clear that two these record types are in the subtype relation iff for all $i, j, k \in \{0, 1\}$ one has $u_{ijk} \leq v_{ijk}$.

Thus the NP-hardness result holds already in the case of a two-label set. In contrast, for just one label the SSCP is deterministic polynomial (even linear) time decidable. This follows from the linearity of typability of an untyped term by simple types. □

Remark 4 ((Narrow Records Suffice)). Another generalization comes from the well-known fact that the particular case of *SATISFIABILITY*, *3-SATISFIABILITY*, restricted to formulas with at most three literals per clause is also NP-complete. It follows that the SSCP remains NP-hard for systems of constraints containing at most three fields per record. □

Remark 5 ((On Functional Constructor)). Our proof uses no assumptions about the $\rightarrow$ constructor, except for its difference from the record constructor; see Remark 1. Consequently, the functional subtyping rule (1) may be changed, e.g., made domain-covariant. This should be contrasted to the results of (Tiuryn 1992), which exploit the domain-contravariance of $\rightarrow$.

5 Satisfiability for Palsberg's Object Types without Empty Record is NP-Hard

If the empty record type [ ] is excluded from the type system of (Palsberg 1995) (by adding either a type constant or type variables to make the set of types nonempty), there is a jump in complexity (as compared with deterministic $O(n^3)$):

Lemma 2. *The SSCP for the pure nonempty record types (5) with the record subtyping rule (2) is NP-hard.*

Proof Sketch. The proof is similar to the proof of Lemma 1. It suffices to represent the truth values without functional type constructor as

$$\mathbf{t} \equiv_{df} [1\colon [1\colon \gamma]] \text{ and } \mathbf{f} \equiv_{df} [1\colon [2\colon \gamma]]. \tag{8}$$

Without the empty record type and with the subtyping rule (2), one has:

1. For every substitution θ, every supertype of $\theta([\ldots, j\colon \mathbf{t}, \ldots])$ (respectively, of $\theta([\ldots, j\colon \mathbf{f}, \ldots]))$ is of the form $[\ldots, j\colon [1\colon [1\colon \tau]], \ldots]$ (resp., $[\ldots, j\colon [1\colon [2\colon \tau]], \ldots]$).
2. The types of the form $[\ldots, j\colon [1\colon [1\colon \sigma]], \ldots]$, $[\ldots, j\colon [1\colon [2\colon \tau]], \ldots]$ cannot have common subtypes.

Now modify the proof of ($\Leftarrow$) in Lemma 1 by defining for every $j = 1, \ldots, k$:

- $\nu(A_j) = \mathbf{t}$ if some $\theta(\beta_i)$ has a field $j\colon [1\colon [1\colon \tau]]$ for some type τ, and
- $\nu(A_j) = \mathbf{f}$ otherwise.

The subsequent case analyses depend on whether a type has form $[1\colon [1\colon \tau]]$ or $[1\colon [2\colon \tau]]$. The remainder of the proof works without any substantial changes. □

It follows that with subtyping rule (2) deriving pure nonempty types is *more complicated* than deriving object types (unless $P = NP$). Both kinds of typability differ, and once we know whether a term has an object type, the next question to ask is whether a term has a nonempty record type. Both represent important interesting problems worth consideration.

6 General Proof Pattern

The proof of Section 4 for functional+nonempty record types with subtyping rule (3) is based on the simultaneous use of both functional and record type constructors in the definitions (6) of the truth value types **t** and **f**. The argument in Section 5, for nonempty object types with subtyping rule (2), suggests that the use of functional types *might also be avoided* with subtyping rule (3).

The first idea to avoid using functional types in the proof of Section 4, is to define truth values **t** and **f** as in (8). However, with this choice Lemma 1 *fails*, because for every conjunction of clauses Φ its translation Φ^* to the set of subtyping judgments defined by (7) *is satisfiable*. Indeed, let $\beta_i = C_i^*$ and

$\alpha = \Big[i\colon [1\colon [1\colon \gamma, 2\colon \gamma]] \Big]_{i=1}^{k}$ (where $A_1, \ldots, A_k$ are all variables in Φ). We see that with the choice (8) the proof of ($\Leftarrow$) in Lemma 1 breaks in the case analysis, where we construct the truth assignment by selecting *true* or *false* depending on whether a type has one of the two mutually exclusive structures.

To better explain this phenomenon, one of the anonymous referees suggested the following '*General pattern*' of the proofs of ($\Leftarrow$) in Lemmas 1 and 2. Both proofs rely on the existence of a property T of types (depending on the codings of $\mathbf{t}$, $\mathbf{f}$) such that for each $\rho \in \{\mathbf{t}, \mathbf{f}\}$, all substitutions θ, all types σ, τ with $\theta(\rho) \leq \tau \geq \sigma$ the following properties are satisfied:

$$\begin{cases} T(\tau) \;\Rightarrow (\rho = \mathbf{t} \wedge \;T(\sigma)), \\ \neg T(\tau) \Rightarrow (\rho = \mathbf{f} \wedge \neg T(\sigma)). \end{cases} \tag{9}$$

(Intuitively, $T(\xi)$ and $\neg T(\xi)$ stand for 'ξ represents *true* or *false*', respectively.)

When θ solves $\Phi^* = \{C_i^* \leq \beta_i \geq \alpha\}_{i=1}^{l}$, we define the truth assignment ν by:

- $\nu(A_j) = \mathbf{t}$ if $\theta(\alpha)$ contains a field $j\colon \sigma$ such that $T(\sigma)$, and
- $\nu(A_j) = \mathbf{f}$ otherwise.

This assignment ν satisfies $\bigwedge_{i=1}^{l} C_i$.

In the proof ($\Leftarrow$) of Lemma 1 we selected $T(x) =$ '*x is a record type*', appropriate for the truth values encodings (6) and subtyping rule (3). Clearly, this choice satisfies (9).

In the proof ($\Leftarrow$) of Lemma 2 we selected $T(x) =$ '*x is a subtype of* $[1\colon [1\colon \xi]]$ (for some type ξ)', appropriate for the truth values encodings (8) and subtyping rule (2). Clearly, this choice satisfies (9), because the empty record is excluded, and the rule (2) guarantees that the types $[1\colon [1\colon \phi]]$ and $[1\colon [2\colon \psi]]$ do not have a common subtype. However, as we saw above, these types *do have a common subtype* with subtyping rule (3).

It is clear that no choice of pure record types for $\mathbf{t}$, $\mathbf{f}$ in the presence of subtyping rule (3) satisfies (9). This can be proved by recursively constructing the lower bound for any two pure record types. Thus, *the use of nonempty record types jointly with functional types*[5] *is essential* in the proof of the Main Theorem in Section 4. This does not imply, however, that the problem is in PTIME, nor does it prevent that the problem (for pure record types) is also NP-hard (by a different proof). But to the author's knowledge, until now this problem is open.

7 Pure Nonempty Records with a Single Atomic Constant

If we allow a *single type constant*[6] c, such that $c \leq \tau$ and $\sigma \leq c$ are only possible for $\sigma = \tau = c$, then the proof of our main result from Section 4 works (slightly

[5] or any other type constant or constructor; see Section 7 below.
[6] Before we had only type variables and two type constructors $\rightarrow$, $[\,]$.

modified according to the 'General pattern' of Section 6) for the truth values defined by

$$\mathbf{t} \equiv_{df} c, \quad \mathbf{f} \equiv_{df} [1\colon c].$$

Indeed, these types satisfy all the needed properties (9) if we let $T(\xi) \equiv_{df}$ 'ξ equals c'. Exactly the same argument works for nonempty records + another type constructor (provided that types with different type constructors are incomparable).

It is interesting to compare this result with the results on complexity of functional types subtyping with subtyping on atomic types.

1. If the atomic subtyping is a lattice (which is the case of a single type constant) then the satisfiability of functional type constraints is PTIME decidable. It becomes PSPACE-complete for n-crowns ($n \geq 2$), (Tiuryn 1992).
2. In the pure nonempty record subtyping it is NP-hard already in the case of a single (atomic) type constant (which is clearly a lattice), in contrast to (Tiuryn 1992, Tiuryn 1997).

What are the precise lower and upper bounds for the SSCP in the case of pure record types in presence of a nontrivial atomic subtyping, lattice and non-lattice? A classification similar to (Tiuryn 1992, Benke 1993, Tiuryn 1997) would be interesting.

8 Upper Bound

(Kozen et al. 1994, Hoang & Mitchell 1995, Palsberg 1995) show that satisfiability of systems of subtype inequalities (for partial types or record types) is polynomial time equivalent to the type reconstruction problem (i.e., given a term can it be assigned some type?). By similar arguments we can prove that in any reasonable type system based on functional+record types with subtyping as defined in Definitions 1, 2 the type reconstruction problem is polynomial time equivalent to the SSCP, hence also is NP-hard.

Therefore, unlike the results of (Kozen et al. 1994, Palsberg 1995), which show deterministic cubic time tractability of the SSCP for either functional or record types (separately), our Main Theorem shows that subtyping and type reconstruction in presence of *both functional and nonempty record* types is NP-hard, i.e., presumably intractable. It follows that the automata-theoretic decidability techniques of (Kozen et al. 1994, Palsberg 1995) (for the functional and record subtyping, separately) do not carry over straightforwardly to the combined case of functional+nonempty record types. Extra effort is necessary even to establish decidability of the SSCP for functional+record types. Here we just claim without a proof the following complexity result

Theorem 2. *The SSCP for functional+nonempty record types of Definitions 1, 2 is in NEXPTIME.* □

Recall that NEXPTIME is the class of problems solvable by nondeterministic Turing machines in time $O(2^{n^k})$ for some fixed k, where n denotes the length of input. The proof of Theorem 2 is by a tedious pumping argument (outside the scope of this paper) showing that whenever an instance of SSCP of size n has a solution, then it necessarily has a solution of depth polynomial in n. Thus, given an SSCP instance, it suffices to nondeterministically guess a polynomially deep solution tree (forest), with the resulting tree size $O(2^{poly(n)})$, and to check that it is indeed a solution in deterministic exponential time. This proves the NEXPTIME membership. Of course, this is not a very efficient algorithm. It remains an open problem whether the SSCP for functional+record types is in PSPACE or NP. We believe that more sophisticated data structures, like DAGs and automata on DAGs, may lead to improvement of the above NEXPTIME upper bound to PSPACE, or even NP. It is also possible that the sophisticated techniques of (Tiuryn 1992) may raise the lower bound to PSPACE. This remains an intriguing subject for further investigations, we will report on elsewhere.

9 Conclusions

We presented the NP-lower bound for the *Satisfiability of Subtype Constraints Problem* (SSCP) for the types built by using simultaneously the functional $\rightarrow$ and the nonempty record $[\ldots]$ type constructors. Earlier research concentrated exclusively either on the SSCP for pure functional, or for pure record types, but not for both simultaneously, or else immediately included $\top$ and $\bot$ types turning the type structure into a lattice. Both in the case of the pure functional types (Lincoln & Mitchell 1992, Tiuryn 1992, Kozen et al. 1994) and pure record types (Palsberg 1995), deterministic polynomial time algorithms are possible. In the case of the pure functional types constructed from atomic types with nontrivial subtyping relation on them the SSCP, in general, is NP-hard (Lincoln & Mitchell 1992) and even PSPACE-hard (Tiuryn 1992) (and, in fact, PSPACE-complete (Frey 1997)). In contrast, our result shows that even without any atomic types, the SSCP for functional+nonempty record types is NP-hard. We give the NEXPTIME upper bound for the problem, but conjecture that by using more sophisticated data structures and more subtle arguments this upper bound can be improved to PSPACE (or even NP). It is also quite possible that the techniques of (Tiuryn 1992) may lead to the PSPACE lower bound.

All these topics constitute an interesting and promising direction for future research, together with the investigation of the related partial type systems with possible simplification of the SSCP, as suggested by results of (Kozen et al. 1994, Palsberg 1995).

We conclude by summarizing several earlier mentioned technical problems remaining open:

1. Improve the NP lower and/or the NEXPTIME upper bounds for the SSCP for functional+nonempty record types with subtyping rule (3).

2. We gave the NP lower bound for pure nonempty record types with a single atomic constant. Does the same hold without the constant, or does the complexity drop to PTIME?
3. The SSCP for Palsberg's object types without empty record is NP-hard. What is the upper bound?
4. Pure functional+nonempty record types seemingly do not allow for modeling crowns. Give a strict proof, so as to demonstrate that our results are independent of (Tiuryn 1992, Pratt & Tiuryn 96).
5. Results of (Tiuryn 1992, Tiuryn 1997) and (Kozen et al. 1994, Palsberg 1995, Brandt & Henglein 1997) on PTIME decidability of constraints of different kinds heavily exploit the fact that the type structure is a lattice. Could one construct a uniform polynomial time algorithm, which works for an arbitrary combination of type features, once types form a lattice?
6. What are the precise lower and upper bounds for the SSCP in the case of pure record types in the presence of a nontrivial atomic subtyping, lattice and non-lattice? A classification similar to (Tiuryn 1992, Benke 1993, Tiuryn 1997) would be interesting.
7. Does there exist a functional+record system of partial types, which: 1) has the PTIME decidable SSCP and type reconstruction problem, 2) types all normal forms, 3) all typable terms are SN, 4) a typable term never goes wrong (appropriately defined)? Could PTIME decidability of such a system be obtained by a generalization of the automata-based decision procedures of (Kozen et al. 1994, Palsberg 1995)?

Acknowledgments. Thanks to David McAllester for motivating this research, for fruitful discussions, careful proofreading, thoughtful and illuminating remarks. Thanks to the anonymous referees for reporting gaps in the proofs, careful reading, numerous constructive suggestions for improvements, sharp remarks. One of the referees suggested substantial generalizations of the main theorem and the general proof pattern of Section 6.

References

Aiken, A., Wimmers, E. & Lakshman, T. (1994), Soft typing with conditional types, *in* '21st ACM Symp. on Principles of Programming Languages (POPL'94)', pp. 163–173.

Benke, M. (1993), Efficient type reconstruction in presence of inheritance, *in* 'Mathematical Foundations of Computer Science'93', Vol. 711 of *Lect. Notes Comput. Sci.*, Springer-Verlag, pp. 272–280.

Brandt, M. & Henglein, F. (1997), Coinductive axiomatization of recursive type equality and subtyping, *in* 'Typed Lambda Calculi and Applications, TLCA'97, Nancy, France', Vol. 1210 of *Lect. Notes Comput. Sci.*, Springer-Verlag, pp. 63–81.

Dwork, C., Kanellakis, P. & Mitchell, J. (1984), 'On the sequential nature of unification', *J. Logic Programming* **1**, 35–50.

Frey, A. (1997), Satisfying subtype inequalities in polynomial space, *in* 'Static Analysis (SAS'97)', Vol. 1302 of *Lect. Notes Comput. Sci.*, Springer-Verlag, pp. 265–277.

Hoang, M. & Mitchell, J. C. (1995), Lower bounds on type inference with subtypes, *in* '22nd ACM Symp. on Principles of Programming Languages (POPL'95)', pp. 176–185.

Kozen, D., Palsberg, J. & Schwartzbach, M. I. (1994), 'Efficient inference of partial types', *J. Comput. Syst. Sci.* **49**, 306–324.

Lincoln, P. & Mitchell, J. C. (1992), Algorithmic aspects of type inference with subtypes, *in* '19th ACM Symp. on Principles of Programming Languages (POPL'92)', pp. 293–304.

O'Keefe, P. M. & Wand, M. (1992), Type inference for partial types is decidable, *in* 'European Symposium on Programming (ESOP'92)', Vol. 582 of *Lect. Notes Comput. Sci.*, Springer-Verlag, pp. 408–417.

Palsberg, J. (1995), 'Efficient inference of object types', *Information and Computation* **123**, 198–209. Preliminary version in LICS'94.

Palsberg, J. & O'Keefe, P. (1995), 'A type system equivalent to flow analysis', *ACM Trans. Progr. Lang. Sys.* **17**(4), 576–599. Preliminary version in POPL'95.

Palsberg, J. & Smith, J. (1996), 'Constrained types and their expressiveness', *ACM Trans. Progr. Lang. Sys.* **18**(5), 519–527.

Pratt, V. & Tiuryn, J. (96), 'Satisfiability of inequalities in a poset', *Fundamenta Informaticæ* **28**, 165–182.

Thatte, S. (1988), Type inference with partial types, *in* 'International Colloquium on Automata, Languages, and Programming (ICALP'88)', Vol. 317 of *Lect. Notes Comput. Sci.*, Springer-Verlag, pp. 615–629.

Tiuryn, J. (1992), Subtype inequalities, *in* '7th Annual IEEE Symp. on Logic in Computer Science (LICS'92)', pp. 308–315.

Tiuryn, J. (1997), Subtyping over a lattice (abstract), *in* '5th Kurt Gödel Colloquium (KGC'97)', Vol. 1289 of *Lect. Notes Comput. Sci.*, Springer-Verlag, pp. 84–88.

Trifonov, V. & Smith, S. (1996), Subtyping constrained types, *in* '3rd Static Analysis Symposium (SAS'96)', Vol. 1145 of *Lect. Notes Comput. Sci.*, Springer-Verlag, pp. 349–365.

Wand, M. & O'Keefe, P. (1989), On the complexity of type inference with coercion, *in* 'Functional Programming Languages and Computer Architecture'89', pp. 293–298.