# Lower bounds on type inference with subtypes


My Hoang* and John C. Mitchell*
Computer Science Department
Stanford University
Stanford, CA 94305
{hoang, mitchell}@cs.stanford.edu



## Abstract

We investigate type inference for programming languages with subtypes. As described in previous work, there are several type inference problems for any given expression language, depending on the form of the subtype partial order and the ability to define new subtypes in programs. Our first main result is that for any specific subtype partial order, the problem of determining whether a lambda term is typable is algorithmically (polynomial-time) equivalent to a form of satisfiability problem over the same partial order. This gives the first exact characterization of the problem that is independent of the syntax of expressions. In addition, since this form of satisfiability problem is PSPACE-hard over certain partial orders, this equivalence strengthens the previous lower bound of NP-hard to PSPACE-hard. Our second main result is a lower bound on the length of most general types when the subtype hierarchy may change as a result of additional type declarations within the program. More specifically, given any input expression, a type inference algorithm tries to find a *most general* (or *principal*) typing. The property of a most general typing is that it has all other possible typings as instances. However, there are several sound notions of instance in the presence of subtyping. Our lower bound is that no sound definition of instance would allow the set of additional subtyping hypotheses about a term to grow less than linearly in the size of the term.


## 1 Introduction

Subtyping is a basic feature of typed object-oriented languages. The main importance of subtyping is that it allows substitutivity: if $A$ is a subtype of $B$, then elements of type $A$ can be used anywhere that an element of type $B$ is required. Among the many implications for statically-typed languages, this allows data structures such as heterogeneous lists, where elements of the list come from arbitrary subtypes of some given type. This paper studies the problem of type inference in the presence of subtyping. Type inference, used in languages such as ML [GMW79, Mil85], Haskell [HF92, H$^+$92] and Miranda [Tur85], is the process of inferring type information that has been omitted from expressions. Type inference allows type errors to be detected at compile time, without forcing programmers to include type annotations in programs.

Although an algorithm for type inference with subtyping was published in 1984 [Mit84, Mit91b], this algorithm has seen little if any practical use. Apart from the fact that languages which could take advantage of this algorithm are only now emerging, the main problems seem to be that the algorithm is inefficient and the output, even for relatively simple input expressions, appears excessively long and cumbersome to read. Some attempts to make the algorithm more practical appear in [FM89, FM90]; some studies of the inherent difficulty of the problem are [WO89, LM92, Tiu92, Ben94]. The previous studies show that some simplifications can be made to the output of the algorithm, the problem is at least NP-hard in the general case (even assuming that the basic operations that occur in programs have relatively simple types) but some special cases could be solved more efficiently.

Our first main result is the algorithmic equivalence between typability with subtyping and a satisfiability problem over partial orders. In particular, for any subtype partial order, deciding whether an expression has any typing at all is equivalent to determining whether a form of satisfiability problem is solvable over this partial order. This gives us a characterization of the decision problem for typing in the presence of subtypes that is independent of the syntax of expressions. One reason why this is important is that, when considering programming languages with particular restrictions on the subtype partial order, we can focus on the satisfiability problem and rest assured that any satisfiability problem could arise in practice. Since this particular satisfiability problem over partial orders has been shown PSPACE-hard, over partial orders in general or certain fixed partial orders, our equivalence also strengthens the best previous lower bound of NP-hard to PSPACE-hard. As noted in [LM92] the naive upper bound is exponential time.

Our equivalence between typability and partial order satisfiability holds even with very restricted assumptions about the types of basic symbols that appear in program expressions. More specifically, it is shown in [LM92] that it is NP-hard to decide whether a lambda term has a type even if all term constants are restricted to having only atomic types. This is done by showing how the satisfaction of inequalities of the form $b \leq t, s \leq t$ over a partial order can be represented as typability of terms using constants only


*Supported in part by NSF Grant CCR-9303099 and the TRW Foundation.

of atomic type. (We use $b$ for an element of the partial order of types and $s$ and $t$ for type variables.) In this paper, we show how arbitrary inequalities of the form $s \leq t \rightarrow u$ and $s \geq t \rightarrow u$ also arise in typing lambda terms using only constants of atomic types. Using earlier results on the complexity of the satisfaction of subtype inequalities [Tiu92], we can use this equivalence to show that the typability problem is PSPACE-hard, even when all constants in expressions have only atomic types.

Our equivalence clearly implies that the only way to devise a practical, polynomial-time type inference algorithm in the presence of subtyping is to restrict the programming language so that only certain forms of subtype partial orders are definable. This is in fact reasonable since, for example, single inheritance always results in forests of trees. In [Ben94], it is claimed that the satisfiability problem is solvable in polynomial time for this case. Therefore, if most programs use only single inheritance, we might expect polynomial-time behavior in practice. However, a practical type inference algorithm must print more than a simple yes/no answer in response to an input language expression. This is particularly important when a program may declare additional types and subtypes. Since a function declared at the top of the program may be called in several different lower contexts, the initial type-checking of the function must tell the programmer which uses of the function will be type correct and which will be erroneous. Otherwise, it will be very difficult to determine, when the type checker rejects a later application of this function, whether the problem lies in the function declaration or its use. Unfortunately, an efficient satisfiability algorithm for special partial orders still does not help us optimize the output of a type inference algorithm.

Given any input expression, a type inference algorithm tries to find a *most general* (or *principal*) typing. The property of a most general typing is that it has all other possible typings as instances. Without subtyping, "instance" boils down to "substitution instance." A consequence is that the most general typing of any given expression is also the syntactically shortest, since no substitution can decrease the size of an expression. However, with subtyping, "instance" involves both substitution and entailment of subtyping hypotheses. Since substitution can render a set of subtyping hypotheses tautologous, a most general typing that involves any subtyping hypotheses about type variables will never be the shortest typing for the expression.

Given a fixed notion of instance, there may be most general typings of different lengths. In [FM89, FM90], an attempt is made to optimize the algorithm from [Mit84] so that the shortest most general typing is produced. However, simple examples given in Section 5 of the present paper show that this is not the best one can do. Specifically, by adopting a more powerful notion of instance than used in previous studies, we can reduce the length of the shortest most general type. In fact, for some expressions, we can eliminate subtyping hypotheses altogether from their most general types. If we were able to do this for all expressions, this would dramatically simplify the output of the type inference algorithm. However, we show that no sound definition of instance would allow the set of additional subtyping hypotheses about a term to grow less than linearly in the size of the term.

The rest of the paper is organized as follows. In Section 2, we define the type system incorporating subtyping. Besides establishing notation, this allows us to define the decision problems of typability and subtype inequality satisfiability. Section 3 and Section 4 are devoted to proving the polynomial-time equivalence of these two decision problems. In Section 5, we investigate the size of most general typings of terms with respect to any sensible definition of instance. Finally, we end with some directions for future work in Section 6.

## 2 Preliminaries

We study a type system for typing untyped lambda terms, possibly containing constant symbols. The set of untyped lambda-terms are generated by the following grammar

$$M \ ::= x \mid c \mid \lambda x.\, M \mid M\, M$$

where $x$ may be any variable and $c$ a constant symbol.

The types of lambda terms are formed using type variables and type constants. Let $B$ be a set of base types (*int*, *bool*, ...). Then the set of types over $B$ is generated by the following grammar

$$\sigma \ ::= b \mid t \mid \sigma \rightarrow \sigma$$

where $t$ is a type variable and $b \in B$. We let $Type^B$ be the set of types over $B$ with no type variables. These are also called the set of ground types over $B$.

Given a set of base types $B$, a *subtype assertion* or *containment* is a formula of the form $\sigma \preceq \tau$, where $\sigma, \tau$ are types over $B$. A subtype assertion $\sigma \preceq \tau$ is said to be *atomic* if $\sigma$ and $\tau$ are either type variables or base types. Let $\leq_B$ be a partial order on $B$. Intuitively, this ordering indicates the subtype ordering on base types in $B$. Let $C$ be a set of subtype assertions. The following proof system defines the relation $C \vdash \sigma \preceq \tau$ which can be read, "$\sigma$ is a subtype of $\tau$ under the additional subtype assumptions of $C$". If $C, C'$ are sets of subtype assertions, we use $C \vdash C'$ to denote that $C \vdash \sigma \preceq \tau$ for every subtype assertion $\sigma \preceq \tau \in C'$.

$$(asmp) \qquad C \vdash \sigma \preceq \tau \quad \text{if } \sigma \preceq \tau \in C \text{ or } \sigma \leq_B \tau$$

$$(ref) \qquad C \vdash \sigma \preceq \sigma$$

$$(trans) \qquad \frac{C \vdash \sigma_1 \preceq \sigma_2 \qquad C \vdash \sigma_2 \preceq \sigma_3}{C \vdash \sigma_1 \preceq \sigma_3}$$

$$(\rightarrow) \qquad \frac{C \vdash \sigma_2 \preceq \sigma_1 \qquad C \vdash \tau_1 \preceq \tau_2}{C \vdash \sigma_1 \rightarrow \tau_1 \preceq \sigma_2 \rightarrow \tau_2}$$

Given a partial order $\langle B, \leq_B \rangle$ on a set of base types $B$, we define the partial order $\preceq_B$ on the set of types over $B$, as $\sigma \preceq_B \tau$ iff $\phi \vdash \sigma \preceq \tau$. We call the relation $\preceq_B$ the *subtyping relation induced by* $\leq_B$. This is the subtyping relation without additional subtyping hypotheses.

We are now ready to define the problem of satisfaction of subtype inequalities. A system of inequalities over a partial order $\mathcal{B} = \langle B, \leq_B \rangle$ is a finite set of formulas of the form $\sigma_1 \leq \sigma_2$, where $\sigma_1$ and $\sigma_2$ are types over $B$. Let $V$ be the set of type variables that appear in a system of inequalities, $\mathcal{I}$. We say that $\mathcal{I}$ is *satisfiable* in the partial order $\mathcal{B}$ if there is a substitution $\varphi\colon V \rightarrow Type^B$ such that $\varphi(\sigma_1) \preceq_B \varphi(\sigma_2)$ for every inequality $\sigma_1 \leq \sigma_2$ in $\mathcal{I}$. We then have the following decision problems for satisfiability of subtype inequalities (abbreviated SSI):

**(SSI)** Given a finite partial order $\mathcal{B}$ and a system $\mathcal{I}$ of inequalities, is $\mathcal{I}$ satisfiable in $\mathcal{B}$?

**($\mathcal{B}$-SSI)** Given a system $\mathcal{I}$ of inequalities, is $\mathcal{I}$ satisfable in a fixed partial order $\mathcal{B}$?

The difference between the two problems is that in (SSI), a problem instance is a pair $\langle \mathcal{B}, \mathcal{I} \rangle$, while an instance of ($\mathcal{B}$-SSI) is a set $\mathcal{I}$ of inequalities to be satisfied over the fixed partial order $\mathcal{B}$. In other words, there is a problem ($\mathcal{B}$-SSI) for each partial order $\mathcal{B}$.

To be able to type lambda terms, we need to know types for the term constants. And, to incorporate subtyping, we also need to know the subtype ordering on base types. A signature $\Sigma = \langle B, \leq_B, T \rangle$ consists of a set $B$ of base types, a partial order $\leq_B$ on $B$, and a set $T$ of pairs of the form $\langle c, \sigma \rangle$ with $c$ a term constant and $\sigma \in Type^B$. A *typing judgement* is a formula $C, \Gamma \triangleright M\colon \sigma$ where $C$ is a set of atomic subtype assertions and $\Gamma$ is a set of assumptions of the form $x\colon\sigma$, with $x$ a variable. Intuitively, *the context* $\Gamma$ represents the type declarations of the free variables used in a program and the *coercion set* $C$ reflects the additional subtyping declarations that may appear in a program. The following proof system is used to identify the set of well-typed terms.

$$(var) \qquad C, \Gamma \triangleright x\colon \sigma \qquad \text{if } x\colon\sigma \in \Gamma$$

$$(const) \qquad C, \Gamma \triangleright c\colon \sigma \qquad \text{if } \langle c, \sigma \rangle \in T$$

$$(abs) \qquad \frac{C, \Gamma\,[x\colon\sigma] \triangleright M\colon \tau}{\Gamma \triangleright \lambda x.\, M\colon \sigma \to \tau}$$

$$(app) \qquad \frac{C, \Gamma \triangleright M\colon \sigma \to \tau \qquad C, \Gamma \triangleright N\colon \sigma}{C, \Gamma \triangleright M\ N\colon \tau}$$

$$(subtp) \qquad \frac{C, \Gamma \triangleright M\colon \sigma}{C, \Gamma \triangleright M\colon \tau} \qquad \text{if } C \vdash \sigma \preceq \tau$$

In the $(abs)$ rule, we use $\Gamma\,[x\colon\sigma]$ to denote the context given by $\Gamma\,[x\colon\sigma] = (\Gamma - \{x\colon\tau\}) \cup \{x\colon\sigma\}$ if $x\colon\tau \in \Gamma$, and $\Gamma\,[x\colon\sigma] = \Gamma \cup \{x\colon\sigma\}$, otherwise. We call the above proof system $ST_{\preceq}$, for simply-typed $\lambda$-calculus with subtyping. We use $\Sigma \vdash_{ST_{\preceq}} C, \Gamma \triangleright M\colon\sigma$ to denote that the typing judgement $C, \Gamma \triangleright M\colon \sigma$ is derivable over signature $\Sigma$. The reason for including a set $C$ of containments not given by the signature, and allowing type variables, is to represent sets of possible typings. For example, if a signature has $int \preceq real$ and $char \preceq string$, then typing $s \preceq t, \phi \triangleright \lambda x.\, x\colon s \to t$ will have both $\lambda x.\, x\colon int \to real$ and $\lambda x.\, x\colon char \to string$ as instances, according to the definition we give in Section 5. We say that a term $M$ is *typable* or well-typed (over a given signature $\Sigma$) if there exists a context $\Gamma$ and type $\sigma$ such that $\Sigma \vdash_{ST_{\preceq}} \phi, \Gamma \triangleright M\colon \sigma$. As a notational convenience, we often drop the coercion set $C$ from typing judgements if it is empty, *i.e.*, we use $\Gamma \triangleright M\colon\sigma$ for the judgement $\phi, \Gamma \triangleright M\colon\sigma$. Just as for SSI, we have the following two decision problems for type inference in the presence of subtyping (abbreviated TIS):

**(TIS)** Given a signature $\Sigma$ and a term $M$, is $M$ typable over the signature $\Sigma$?

**($\Sigma$-TIS)** Given a term $M$, is $M$ typable over a fixed signature $\Sigma$?

The difference between the two problems is that in (TIS), a problem instance is a pair $\langle \Sigma, M \rangle$, while an instance of ($\Sigma$-TIS) is a typability problem over the fixed signature $\Sigma$. In other words, there is a problem ($\Sigma$-TIS) for each signature $\Sigma$.

As in [Mit91b], we can prove that the type of an expression $M$ only depends on the type assumptions about its free variables. We use the notation $\Gamma(x)$ to denote the unique $\sigma$ such that $x\colon\sigma \in \Gamma$.

**Lemma 2.1** *Assume that* $\vdash_{ST_{\preceq}} C, \Gamma \triangleright M\colon \sigma$. *Suppose* $\Gamma'$ *is a context such that for all variables* $x$ *that are free in* $M$, $\Gamma'(x) = \Gamma(x)$. *Then* $\vdash_{ST_{\preceq}} C, \Gamma' \triangleright M\colon \sigma$.

A useful consequence of restricting the assertions in the coercion set of typing judgements to be atomic is the following property which states that one may normalize proofs in $ST_{\preceq}$ so that the only uses of $(subtp)$ rule occur immediately after $(var)$ and $(const)$.

**Lemma 2.2** *For every provable typing statement* $C, \Gamma \triangleright M\colon \sigma$, *there is a proof in which rule (subtp) is only used immediately after the typing axioms (var) and (const).*

## 3 Type Inference Reduced to Inequality Satisfaction

It is well-known that the typability problem for simply-typed $\lambda$-calculus can be reduced to unification [ASU86, Wan87]. In this section we show that the analagous constraint satisfaction problem for typability in the presence of subtying is the satisfiability of subtype inequalities. More precisely, we exhibit a polynomial time reduction from TIS to SSI in which the poset constructed only depends on the signature. This reduction was implicit in [Mit91b] and [Tiu92].

In the rest of the paper we will use formulas of the form $\sigma = \tau$, when defining a system of inequalities, as an abbreviation for the pair of inequalities $\sigma \leq \tau$ and $\tau \leq \sigma$. Since $\preceq_B$ is a partial order (as opposed to a preorder) the equation $\sigma = \tau$ is satisfiable iff $\sigma \leq \tau$ and $\tau \leq \sigma$ are satisfiable.

If $\Sigma = \langle B, \leq_B, T \rangle$ is a signature with subtyping, then we define the partial order $\mathcal{P}_\Sigma$ to be $\langle B, \leq_B \rangle$.

**Lemma 3.1** *Given a signature (with subtyping)* $\Sigma$, *the decision problem for typability over* $\Sigma$ *is polynomial time reducible to the satisfiability problem for subtype inequalities over* $\mathcal{P}_\Sigma$, *i.e.,* $\Sigma$*-TIS* $\leq_m$ $\mathcal{P}_\Sigma$*-SSI.*

**Proof** Let $\Sigma = \langle B, \leq_B, T \rangle$ be a signature. For any term $M$ and a context $A$ such that $A(x)$ is a type variable for every $x \in \mathrm{Dom}(A)$, we define $SI(M, A) = \langle t, \mathcal{I} \rangle$ by induction on the structure of $M$ as follows, with $t$ a type variable and $\mathcal{I}$ a system of inequalities.

$$\begin{array}{lll}
SI(x, A) & = & \langle t, \{A(x) \leq t\} \rangle \\
 & & \quad \text{where } t \text{ is a fresh type variable not in } A. \\
SI(c, A) & = & \langle t, \{\sigma \leq t\} \rangle \\
 & & \quad \text{where } \langle c, \sigma \rangle \in C \text{ and } t \text{ is fresh.} \\
SI(MN, A) & = & \textbf{let } \langle t_1, \mathcal{I}_1 \rangle = SI(M, A) \\
 & & \qquad \langle t_2, \mathcal{I}_2 \rangle = SI(N, A) \\
 & & \textbf{in} \\
 & & \quad \langle t, \mathcal{I}_1 \cup \mathcal{I}_2 \cup \{t_1 = t_2 \to t\} \rangle,\ t \text{ is fresh.} \\
SI(\lambda x.\, M, A) & = & \textbf{let } \langle t_1, \mathcal{I}_1 \rangle = SI(M, A \cup \{x\colon s\}) \\
 & & \quad \text{where } s \text{ is fresh} \\
 & & \textbf{in} \\
 & & \quad \langle t, \mathcal{I}_1 \cup \{t = s \to t_1\} \rangle \\
 & & \quad \text{where } t \text{ is fresh}
\end{array}$$

The reduction from $\Sigma$-TIS to $\mathcal{P}_\Sigma$-SSI can then be given as follows. For any term $M$, we produce the set of inequalities $\mathcal{I}$, where $SI(M, A) = \langle t, \mathcal{I} \rangle$ with $A$ a context that maps each variable $x$, free in $M$, to a distinct type variable. Using Lemma 2.2 it can be easily seen that this is a reduction. Since each inductive clause in the definition of $SI$ adds only two new inequalities (a constant number) $|\mathcal{I}| \leq c\,|M|$, and this is a polynomial time reduction. ■

**Corollary 3.2** *The problem of deciding whether a given term is typable over a given signature, with subtyping, is polynomial time reducible to the satisfiability of subtype inequalities problem, i.e., TIS $\leq_m$ SSI.*

In the next section we will focus attention on systems of inequalities in which all formulas are of the form $b \leq t$, $s \leq t$, $s \leq t \rightarrow u$, or $t \rightarrow u \leq s$ where $s, t, u \in V$, and $b \in B$. We call the satisfiability problem for this restricted system of inequalities $\mathrm{SSI}_{restr}$. If the signature $\Sigma = \langle B, \leq_B, T \rangle$ is such that all constants have atomic types, *i.e.*, for all $\langle c, \sigma \rangle \in T$, $\sigma \in B$, we call the TIS problem for such input signature $\mathrm{TIS}_{atom}$. Note that $\Sigma$-$\mathrm{TIS}_{atom}$ really restricts the signature $\Sigma$ rather than instances of the problem.

**Corollary 3.3** *The decision problem for typability over a signature $\Sigma$ in which all constants have atomic types is polynomial time reducible to the satisfiability problem for restricted systems of subtype inequalities over $\mathcal{P}_\Sigma$, i.e., $\Sigma$-$TIS_{atom} \leq_m \mathcal{P}_\Sigma$-$SSI_{restr}$. Hence, $TIS_{atom} \leq_m SSI_{restr}$.*

**Proof** The inequalities added in the inductive cases of $SI$ when the term is an application or $\lambda$-abstraction are of the form $s = t \rightarrow u$. When the term is a variable the inequality is of the form $s \leq t$. When the term is a constant the inequality added is $\sigma \leq t$ which is of the form $b \leq t$ when $\sigma$ is atomic. ■

## 4 Reducing Inequality Satisfaction to Typability

In Section 3, we showed how a solution to the type-inference problem in the presence of subtypes can be obtained from a solution to the problem of satisfaction of subtype inequalities. In this section, we establish the converse reduction, namely, we prove that SSI is polynomial time reducible to TIS. As corollaries of this reduction, we are able to translate lower-bound results about SSI to TIS. Together with results in Section 3, this shows the equivalence of the two problems TIS and SSI. When there are no subtype assumptions on base types, type-checking for simply-typed lambda calculus with subtyping reduces to that of simply-typed lambda calculus. And satisfaction of subtype inequalities over a discrete partial order is the same as unification. This gives the well known equivalence between type inference for simply-typed lambda calculus and unification [Tys88, Mit91a, Wan87] as a consequence of our results.

Although interest in type inference with subtypes gave rise to the earlier studies of satisfaction of subtype inequalities over partial orders [PT91, Tiu91, Tiu92], the precise connection between the two problems was not previously understood. In fact, comments in [Tiu92] suggest the belief that the satisfaction of inequalities would turn out to be algorithmically more complex. This suggests some of the subtlety of our proof that the two problems are in fact polynomial-time equivalent.

### 4.1 $\mathrm{SSI}_{restr}$ reduced to $\mathrm{TIS}_{atom}$

We begin by showing how inequalities of the *restricted* form can be simulated by corresponding terms. More precisely, we will exhibit a polynomial time reduction from $\mathrm{SSI}_{restr}$ to $\mathrm{TIS}_{atom}$. Before doing that, we state two lemmas that we will need in the proof of correctness of the reduction.

**Lemma 4.1** *Suppose $\mathcal{D}$ is a derivation of $\Gamma \triangleright M : \sigma$ in $ST_{\preceq}$. Let $\varphi$ be any substitution. Define the derivation $\varphi(\mathcal{D})$ to be the one obtained by replacing every line of the form $\Gamma' \triangleright M' : \sigma'$ by $\varphi(\Gamma') \triangleright M' : \varphi(\sigma')$. Then $\varphi(\mathcal{D})$ is a valid derivation in $ST_{\preceq}$.*

**Proof** We can prove by induction on $\tau_1$ that if $\tau_1 \preceq_B \tau_2$ then for any substitution $\varphi$, $\varphi(\tau_1) \preceq_B \varphi(\tau_2)$. The lemma now easily follows by inspection of each axiom and inference rule of $ST_{\preceq}$. ■

In particular by choosing $\varphi$ to be a ground substitution, *i.e.*, one that maps all variables to ground types, Lemma 4.1 shows that if any term $M$ is typable then it has a typing derivation in which all type occurrences are ground types. Furthermore, by Lemma 2.2 we can assume that the derivation uses *(subtp)* only after *(var)* or *(const)*.

The following lemma shows that typing an application term forces certain subtype inequalities to be satisfied. This is an easy consequence of Lemma 2.2.

**Lemma 4.2** *Let $\Sigma = \langle B, \preceq_B, C \rangle$ be a signature, suppose $\Sigma \vdash_{ST_{\preceq}} \Gamma \triangleright xy : \sigma$. Then $\Gamma(x) = \rho_1 \rightarrow \tau_1$, $\Gamma(y) = \rho_2$ with $\rho_2 \preceq_B \rho_1$ and $\tau_1 \preceq_B \sigma$.*

**Proof** Consider any derivation of $\Gamma \triangleright xy : \sigma$ of the form given by Lemma 2.2

$$\frac{\Gamma \triangleright x : \delta_1 \rightarrow \sigma \qquad\qquad \Gamma \triangleright y : \delta_1}{\Gamma \triangleright xy : \sigma}$$

with $\Gamma(x) \preceq_B \delta_1 \rightarrow \sigma$ and $\Gamma(y) \preceq_B \delta_1$. Then $\Gamma(x) = \rho_1 \rightarrow \tau_1$ and with $\delta_1 \preceq_B \rho_1$ and $\tau_1 \preceq_B \sigma$ which gives us the statement of the lemma. ■

Let $\mathcal{P} = \langle P, \leq_P \rangle$ be any poset. Define the signature $\Sigma_{\mathcal{P}}^{atom} = \langle P, \leq_P, T \rangle$ where $T = \{c_p : p \mid p \in P\}$, *i.e.*, for each element of the partial order, we define a constant of that type. Note that $\Sigma_{\mathcal{P}}^{atom}$ assigns atomic types to all constants. We are now ready to establish the converse to Corollary 3.3.

**Lemma 4.3** *The satisfiability problem for restricted systems of subtype inequalities over a poset $\mathcal{P}$ is polynomial time reducible to the typability problem over the atomic signature $\Sigma_{\mathcal{P}}^{atom}$, i.e., $\mathcal{P}$-$SSI_{restr} \leq_m \Sigma_{\mathcal{P}}^{atom}$-TIS.*

The reduction and the proof of its correctness appear in the Appendix.

**Corollary 4.4** *The satisfiability problem for restricted system of inequalities is polynomial time reducible to the problem of deciding whether a given term is typable over a given signature in which all constants have atomic types, i.e., $SSI_{restr} \leq_m TIS_{atom}$.*

**Corollary 4.5** *There exists a signature $\Sigma$ which maps constants to atomic types such that $\Sigma$-$TIS_{atom}$ is* PSPACE*-hard. Hence, the problem $TIS_{atom}$ is also* PSPACE*-hard.*

**Proof** By results in [Tiu92], there exists a poset $\mathcal{P}$ for which $\mathcal{P}$-$\mathrm{SSI}_{restr}$ is PSPACE-hard. Lemma 4.3 translates this result to $\mathrm{TIS}_{atom}$. ■

### 4.2 SSI reduced to TIS

We now generalize the reduction described in Section 4.1 to show how satisfiability of a set of arbitrary inequalities can be simulated by typability of a term. To simulate inequalities that are not necessarily of a restricted form, we no longer have the requirement that the signature give only atomic types to constants.

Let $\mathcal{P} = \langle P, \leq_P \rangle$ be any poset. We define the signature $\Sigma_{\mathcal{P}} = \langle P, \leq_P, T \rangle$ where $T = \{\langle c_p^l, p \rightarrow p \rangle \mid p \in P\} \cup \{\langle c_p^u, p \rangle \mid p \in P\}$. Note that the signature $\Sigma_{\mathcal{P}}$ is not atomic. The following lemma generalizes the reduction of $\mathcal{P}$-SSI$_{restr}$ to $\Sigma_{\mathcal{P}}^{atom}$-TIS and shows how arbitrary inequalities can be simulated by terms in the signature $\Sigma_{\mathcal{P}}$ using the constants $c_p^l$'s of functional type.

**Lemma 4.6**

*(i) The satisfiability problem for subtype inequalities over a poset $\mathcal{P}$ is polynomial time reducible to the typability problem over a signature $\Sigma_{\mathcal{P}}$, i.e., $\mathcal{P}$-SSI $\leq_m$ $\Sigma_{\mathcal{P}}$-TIS.*

*(ii) The satisfiability of subtype inequalities problem is polynomial time reducible to the typability problem, i.e., SSI $\leq_m$ TIS.*

**Proof**

(i) For any type $\sigma$ over $P$, we define a term $M_\sigma$ and a context $C_\sigma[\,]$ with one hole simultaneously by induction $\sigma$. Intuitively, $M_\sigma$ is defined so that $\sigma$ is a subtype of any type of $M_\sigma$, and $C_\sigma[\,]$ is defined so that for any term $N$, a typing of $C_\sigma[N]$ constrains the type of $N$ to be a subtype of $\sigma$. If $\sigma$ is a type over variable set $V = \{t_1, \ldots, t_n\}$ then $M_\sigma$ and $C_\sigma[\,]$ have free variables $u_1, v_1, \ldots, u_n, v_n$.

- $M_{t_i} = v_i\, u_i$
  $C_{t_i}[\,] = v_i[\,]$
- $M_p = c_p^u$
  $C_p[\,] = c_p^l[\,]$
- $M_{\sigma \rightarrow \tau} = \lambda x.\, K\, M_\tau\, C_\sigma[x]$
  $C_{\sigma \rightarrow \tau}[\,] = C_\tau[[\,]\, M_\sigma]$

where $K$ is the combinator $(\lambda x.(\lambda y.\, x))$. Given an arbitrary inequality $\sigma_1 \leq \sigma_2$, we can now define the term $[\sigma_1 \leq \sigma_2]$ as follows:

$$[\sigma_1 \leq \sigma_2] \quad = \quad C_{\sigma_2}[M_{\sigma_1}]$$

As in lemma 4.3, for any system $\mathcal{I}$ of inequalities, we produce the following term

$$\begin{array}{rcl} M & = & \lambda u_1 \ldots \lambda u_n. \\ & & (\lambda v_1 \ldots \lambda v_n. \\ & & (\lambda x.\,[i_1])((\lambda x.\,[i_2])(\cdots((\lambda x.\,[i_{m-1}])[i_m])\cdots)) \\ & & )\, \underbrace{(\lambda x.\, x) \cdots (\lambda x.\, x)}_{n \text{ times}} \end{array}$$

where $\mathcal{I} = \{i_1, \ldots, i_m\}$

(ii) The signature $\Sigma_{\mathcal{P}}$ can be produced from the partial order $\mathcal{P}$ in polynomial time.

■

From Corollaries 3.2, 3.3, 4.4, and Lemma 4.6, we have the following theorem.

**Theorem 4.7**

*(i) The satisfiability problem for restricted system of subtype inequalities (SSI$_{restr}$) and the typability problem over a signature in which all constants have atomic types (TIS$_{atom}$) are polynomial time equivalent.*

*(ii) The satisfiability of subtype inequalities (SSI) and the typability problems (TIS) are polynomial time equivalent.*

## 5 Most General Typings

In this section we investigate a general theory of instances and most-general typing judgements for the type system $ST_{\preceq}$. While the typability problems we considered before were with respect to the given fixed subtype ordering on base types, we will now study most general tyings for typing judgements with additional subtype assumptions. One reason for allowing additional subtype hypotheses is that, in general, this is the only way to obtain most general types. A second justification is that from the most general typing for an expression which includes additional subtype assumptions, it is possible to decide whether the expression is typable with respect to any fixed subtype ordering.

Intuitively, a most general typing judgement for a term is one which characterizes all possible typings for it. In more detail, if $C, \Gamma \triangleright M\!:\sigma$ is a most general typing for $M$, then all other derivable typings $C', \Gamma' \triangleright M\!:\sigma'$ can be obtained from it. The precise manner in which $C', \Gamma' \triangleright M\!:\sigma'$ is obtained from the most general typing $C, \Gamma \triangleright M\!:\sigma$ is via a definition of *instance*, a binary relation on typing judgements. Then a most general typing for a term $M$ is a derivable typing judgement such that all other derivable typings for $M$ are instances of it. Thus, most general typings are crucially related to the notion of instance that one uses. In [Mit91b], the following definition of instance is given:

**Definition 5.1** *A typing statement $C', \Gamma' \triangleright M\!:\sigma'$ is an instance of $C, \Gamma \triangleright M\!:\sigma$ if there exists a substitution $S$ such that:*

*(i)* $C' \vdash S\,C$,

*(ii)* $\sigma' = S\,\sigma$,

*(iii)* $\forall x \in Dom(\Gamma), \Gamma'(x) = S\,\Gamma(x)$.

[Mit91b] also gives an algorithm GA which infers a most general typing for any term with respect to the stated definition of instance. Unfortunately, the size of the coercion set in the most general typing produced by GA can become exponential in the size of the term. Since an important aspect of any type-inference algorithm is the size and readability of its output, there has been previous work on optimizing the type inference algorithm to produce the shortest most general typings [FM89, FM90]. However, there is an intrinsic limit to the shortest most general typings that can be produced if we adhere to Definition 5.1 of instance. To give a relatively simple example, there are typings for terms which cannot be instances of any typing with empty coercion set.

**Example 5.2** *Let* $M = \lambda f.\,\lambda x.\,\lambda y.\,K\,(f\,x)\,(f\,y)$. *Using type* $s$ *for* $x$ *and type* $t$ *for* $y$, *a derivable typing for* $M$ *is:*

$$\{s \preceq u, t \preceq u, v \preceq w\},\ \phi \triangleright M\colon (u \rightarrow v) \rightarrow s \rightarrow t \rightarrow w \qquad (1)$$

*A typing for the term* $M$ *with empty coercion set is:*

$$\phi, \phi \triangleright M\colon (s \rightarrow t) \rightarrow s \rightarrow s \rightarrow t \qquad (2)$$

*However, the typing (2) with empty coercion set cannot have the typing (1) as an instance, using Definition 5.1. For, there can be no substitution* $S$ *such that* $S\sigma = \sigma'$, *where* $\sigma = (s \rightarrow t) \rightarrow s \rightarrow s \rightarrow t$ *and* $\sigma' = (u \rightarrow v) \rightarrow s \rightarrow t \rightarrow w$.

Since the problem in Example 5.2 arises from the definition of instance that one has chosen, a natural object of interest is an alternative formulation of the notion of instance which would permit most general typings with succinct set of subtype assumptions. It would be particularly good, if for example, one can obtain most general typings in which the subtype assumption set is always empty. The following more general definition of instance allows the typing (1) to be an instance of (2), and thus allows a most general typing of $M$ from Example 5.2 with an empty coercion set.

**Definition 5.3** *A typing statement* $C', \Gamma' \triangleright M\colon \sigma'$ *is an instance of* $C, \Gamma \triangleright M\colon \sigma$ *if there exists a substitution* $S$ *such that:*

*(i)* $C' \vdash S\,C$,

*(ii)* $C' \vdash S\,\sigma \preceq \sigma'$,

*(iii)* $\forall x \in Dom(\Gamma), C' \vdash \Gamma'(x) \preceq S\,\Gamma(x)$.

We can see that the typing (2) has (1) as an instance, by Definition 5.3, using the the substitution $S$ defined by $S(s) = u$ and $S(t) = v$. Unfortunately, even this more powerful notion of instance does not allow typings with empty coercion sets to be more general than all other derivable typings for arbitrary terms.

**Example 5.4** *Take* $M = f(f\,x)$. *Consider any derivable typing for* $M$, $\phi, \Gamma \triangleright M\colon \sigma$, *with empty coercion set. Then we must have* $\Gamma(x) = \sigma, \Gamma(f) = \sigma \rightarrow \sigma$. *We can see that this typing with empty coercion set cannot have the following typing for* $M$ *as an instance.*

$$\{s \preceq t, u \preceq v, u \preceq t\},\ \{f\colon t \rightarrow u, x\colon s\} \triangleright f(f\,x)\colon v$$

*For consider any substitution* $S$. *By condition (ii) of Definition 5.3, we must have* $C' \vdash S(\sigma) \preceq v$, *where* $C' = \{s \preceq t, u \preceq v, u \preceq t\}$. *By inspection of* $C'$, *this implies that* $S(\sigma) = u$ *or* $S(\sigma) = v$. *In either case, we cannot satisfy condition (iii) for the variable* $x$, *since* $C' \not\vdash s \preceq u$ *and* $C' \not\vdash s \preceq v$.

Investigation of other plausible notions of instance by the authors also failed to allow typings with empty coercion sets to be most general. It was therefore a natural question whether there could be *any* definition of instance that would permit such elegant most general typings. In this section, we show that this is impossible. The main result of this section is that there is no suitable definition of instance for which there is a bound on the size of the coercion set in most general typings of terms. Thus we need to allow arbitrarily many subtype assumptions in giving the most general types of terms, in general.

We begin with a formalization of the properties of any suitable definition of instance. An instance relation , $\succ$, is a binary relation on typing judgements. Intuitively, $C, \Gamma \triangleright M\colon \sigma \succ C, \Gamma' \triangleright M\colon \sigma'$ means that the typing judgement $C, \Gamma \triangleright M\colon \sigma$ is more general than $C', \Gamma' \triangleright M\colon \sigma'$. A definition of instance usually does not depend on the term $M$ appearing in the judgements. Further, a basic property that one requires of any definition of instance, is the closure of derivability under instantiation. These two properties are captured in the following definition. Since all the properties in this section are proved for pure $\lambda$-terms, *i.e.*, without any term constants, we omit mentioning any signature in the following definition and the rest of this section.

**Definition 5.5** *Let* $\succ$ *be a definition of instance on typing judgements. Then* $\succ$ *is a* **sound** *definition of instance if:*

*(i)* $C, \Gamma \triangleright M\colon \sigma \succ C', \Gamma' \triangleright M\colon \sigma'$ *iff* $C, \Gamma \triangleright N\colon \sigma \succ C', \Gamma' \triangleright N\colon \sigma'$, *for all terms* $N$.

*(ii) If* $C, \Gamma \triangleright M\colon \sigma \succ C', \Gamma' \triangleright M\colon \sigma'$ *and* $\vdash_{ST_\preceq} C, \Gamma \triangleright M\colon \sigma$ *then* $\vdash_{ST_\preceq} C', \Gamma' \triangleright M\colon \sigma'$.

Given a definition of instance, $\succ$, a typing judgement $C, \Gamma \triangleright M\colon \sigma$ is a most general typing for term $M$ if

- The typing judgement $C, \Gamma \triangleright M\colon \sigma$ is derivable.
- For all derivable typing judgements $C, \Gamma' \triangleright M\colon \sigma'$, we have $C, \Gamma \triangleright M\colon \sigma \succ C, \Gamma' \triangleright M\colon \sigma'$.

We can now show that the term $f(f\,x)$ cannot have a most general typing, under any sound definition of instance, in which the coercion set is empty. This is a special case of a more general theorem that we will prove later, but its proof is illustrative in understanding the proof of the more general theorem.

**Lemma 5.6** *Let* $M = f(f\,x)$ *and* $\succ$ *be any sound definition of instance. Then* $M$ *does not have a most general typing in which the coercion set is empty.*

**Proof** Consider any derivable typing $\phi, \Gamma \triangleright M\colon \sigma$ with empty coercion set. Then we must have $x\colon \sigma, f\colon \sigma \rightarrow \sigma \in \Gamma$. We will show that $\phi, \Gamma \triangleright M\colon \sigma$ cannot be a most general typing for $M$.

Let $C' = \{s \preceq t, u \preceq t, u \preceq v\}, \Gamma' = \{x\colon s, f\colon t \rightarrow u\}, \sigma' = v$. Clearly, $\vdash_{ST_\preceq} C', \Gamma' \triangleright M\colon \sigma'$. Consider the term $N = x$. Since $x\colon \sigma \in \Gamma$, we have $\vdash_{ST_\preceq} \phi, \Gamma \triangleright N\colon \sigma$. But since $C' \not\vdash s \preceq v$, we have $\not\vdash_{ST_\preceq} C', \Gamma' \triangleright N\colon \sigma'$. By property (ii) of a sound definition of instance,

$$\phi, \Gamma \triangleright N\colon \sigma \not\succ C', \Gamma' \triangleright N\colon \sigma'$$

Hence, by property (i) of a sound definition of instance,

$$\phi, \Gamma \triangleright M\colon \sigma \not\succ C', \Gamma' \triangleright M\colon \sigma'$$

∎

We will now generalize Lemma 5.6 to show that no bound on the size of the coercion set will suffice to express most general typings of all terms. For any atomic coercion set $C$ and set of type variables $T$, define the coercion set $C\restriction_T$ to be the set

$$\{s \preceq t \in C \mid s \in T \text{ or } t \in T\}$$

We say that an atomic coercion set $C$ is *closed* if for any non-trivial atomic subtyping assertion, $s \preceq t$ with $s \neq t$,

$C \vdash s \preceq t$ iff $s \preceq t \in C$. For any atomic coercion set $C$, define its *closure* as the atomic coercion set defined by

$$Clos(C) = \{s \preceq t \mid s \neq t, C \vdash s \preceq t\}$$

We use $FTV(C)$ to denote the type variables appearing in the coercion set $C$, and use $FTV(\sigma), FTV(\Gamma)$ for a type $\sigma$ and context $\Gamma$, similarly. We now state without proofs some technical lemmas involving free type variables in coercion sets which will be useful in the proof of our main theorem.

**Lemma 5.7** *For any atomic coercion set $C$,*

*(i) $Clos(C)$ is closed.*

*(ii) $FTV(Clos(C)) \subseteq FTV(C)$.*

**Lemma 5.8** *Let $C$ be an atomic coercion set.*

*(i) If $C \vdash \sigma \preceq \tau$ then $Clos(C) \vdash \sigma \preceq \tau$.*

*(ii) If $\vdash_{ST_{\preceq}} C, \Gamma \triangleright M{:}\sigma$ then $\vdash_{ST_{\preceq}} Clos(C), \Gamma \triangleright M{:}\sigma$.*

**Lemma 5.9** *Suppose $C$ is a closed atomic coercion set.*

*(i) If $C \vdash \sigma \preceq \tau$ and $FTV(\sigma) \subseteq T$ or $FTV(\tau) \subseteq T$ then $C\restriction_T \vdash \sigma \preceq \tau$.*

*(ii) If $\vdash_{ST_{\preceq}} C, \Gamma \triangleright M{:}\sigma$ and $M$ has no $\lambda$-abstractions then $\vdash_{\preceq} C\restriction_{FTV(\Gamma)}, \Gamma \triangleright M{:}\sigma$.*

We are now ready to show that we cannot bound the size of the coercion set of most general typings. As in Lemma 5.6, we will use terms of the form $f(f\,x)$ in proving our result. More precisely, we show that the term

$$M = z\ f_1(f_1\,x_1) \cdots f_{k+1}(f_{k+1}\,x_{k+1})$$

cannot have a most general typing which uses at most $k$ subtype assumptions. Roughly, the argument is as follows. If there are no subtype assumptions involving the types in one of the subterms $f_i(f_i\,x_i)$ then this subterm has been typed using an empty coercion set and hence by an argument similar to Lemma 5.6, this typing cannot be most general. Otherwise, a typing which uses at most $k$ subtype assumptions must relate the types used in two subterms $f_i(f_i\,x_i)$ and $f_j(f_j\,x_j)$. Since this is a constraint that does not have to be enforced in typing $M$, such a typing must not be more general than a typing in which the type assumptions used in typing the subterms $f_i(f_i\,x_i)$ and $f_j(f_j\,x_j)$ are completely unrelated. All this is made more precise, using Lemmas 5.7, 5.8, 5.9, in the proof of the following theorem which is given in the Appendix.

**Theorem 5.10** *Let $\succ$ be any sound definition of instance. For every $k$ there is a term $M_k$ with $|M_k| = O(k)$ such that the most general typing of $M_k$ requires a coercion set with at least $k+1$ elements, i.e., the size of the coercion set grows at least linearly in the size of the term.*

## 6 Conclusion

This paper contains two results on type inference with subtyping. The first shows that typability of a term in the presence of a fixed subtype ordering is equivalent to satisfiability of subtype inequalities over the same subtype ordering. A natural special case of typability arises when all program constants only have atomic types. This special case is equivalent to satisfiability of a certain restricted form of subtype inequalities. Since this restricted satisfiability problem has been shown previously to be PSPACE-hard, our equivalence gives a PSPACE lower bound on the algorithmic problem of typability with either general or atomic types. The second main result is concerned with most general typings, which are the typical output of a type-inference algorithm. Instead of working with a particular notion of instance, we have given a general lower bound that is independent of the notion of instance one may chose. Our lower bound is that for any sound notion of instance, the number of subtype assumptions in the most general typing grows linearly in the length of the term.

One important investigation is a precise characterization of the structural complexity of the problem of typability. The equivalences shown here demonstrate the importance of the problem of subtype inequality satisfaction and the importance of inequalities of the restricted form. One open problem is to find the complexity-theoretic relationship between satisfiability with arbitrary subtype inequalities and satisfiability with inequalities of the restricted form. Another problem is to give matching upper bounds and lower bounds for each problem.

The lower bounds in this paper suggest the importance of finding more tractable subproblems. This would have important consequences for programming language design. Both the algorithmic complexity of typability and the lack of succinct most general typings appear to arise due to particular partial orders that may not arise in practice. It is therefore important to identify classes of subtype orderings that lead to reasonably flexible programming languages and, at the same time, yield tractable type inference problems.

## References


[ASU86] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers: principles, techniques, tools.* Addison-Wesley, 1986.

[Ben94] M. Benke. Efficient type reconstruction in the presence of inheritance. Manuscript, 1994.

[FM89] Y. Fuh and P. Mishra. Polymorphic subtype inference: closing the theory-practice gap. In *Proc. Theory and Practice of Software Development*, pages 167–184, March 1989.

[FM90] Y. Fuh and P. Mishra. Type inference with subtypes. *Theor. Computer Science*, 73, 1990.

[GMW79] M.J. Gordon, R. Milner, and C.P. Wadsworth. *Edinburgh LCF.* Springer LNCS 78, Berlin, 1979.

[H+92] P. Hudak et al. Report on the programming language Haskell. *SIGPLAN Notices*, 27(5):Section R, 1992.

[HF92] P. Hudak and J. Fasel. A gentle introduction to Haskell. *SIGPLAN Notices*, 27(5):Section T, 1992.

[LM92] P.D. Lincoln and J.C. Mitchell. Algorithmic aspects of type inference with subtypes. In *Proc. 19th ACM Symp. on Principles of Programming Languages*, pages 293–304, January 1992.

[Mil85] R. Milner. The Standard ML core language. *Polymorphism*, 2(2), 1985. 28 pages. An earlier version appeared in Proc. 1984 ACM Symp. on Lisp and Functional Programming.

[Mit84] J.C. Mitchell. Coercion and type inference (summary). In *Proc. 11th ACM Symp. on Principles of Programming Languages*, pages 175–185, January 1984.

[Mit91a] J.C. Mitchell. A simple reduction of unification to typability. Message to `types@theory.lcs.mit.edu`, June 28 1991.

[Mit91b] J.C. Mitchell. Type inference with simple subtypes. *J. Functional Programming*, 1(3):245–286, 1991.

[PT91] V. Pratt and J. Tiuryn. Satisfiability of inequations in a poset. Manuscript, October 1991.

[Tiu91] J. Tiuryn. Solving term inequalities is PSPACE-hard. Manuscript, October 1991.

[Tiu92] J. Tiuryn. Subtype inequalities. In *Proc. IEEE Symp. on Logic in Computer Science*, pages 308–315, 1992.

[Tur85] D.A. Turner. Miranda: a non-strict functional language with polymorphic types. In *IFIP Int'l Conf. on Functional Programming and Computer Architecture, Nancy*, Berlin, 1985. Springer LNCS 201.

[Tys88] J. Tyszkiewicz. Complexity of type inference in finitely typed lambda calculus. Master's thesis, University of Warsaw, 1988.

[Wan87] M. Wand. A simple algorithm and proof for type inference. *Fundamenta Informaticae*, 10:115–122, 1987.

[WO89] M. Wand and P. O'Keefe. On the complexity of type inference with coercion. In *Proc. ACM Conf. Functional Programming and Computer Architecture*, pages 293–298, 1989.


## A Appendix

**Proof of Lemma 4.3** Suppose $\mathcal{P} = \langle P, \leq_P \rangle$. Let $\mathcal{I}$ be a restricted system of inequalities over $\mathcal{P}$, with type variables from a finite set $V = \{t_1, \ldots, t_n\}$. For any inequality $i \in \mathcal{I}$, we will define a term $[i]$ over $\Sigma_{\mathcal{P}}^{atom}$ with free variables $u_1, v_1, \ldots, u_n, v_n$. In defining the terms $[i]$, the variables $v_i$ should be understood as having function types $\sigma_i \to \tau_i$ with $\sigma_i$ serving as a lower bound on any satisfying substitution for the type variable $t_i$, and $\tau_i$ serving as an upper bound. Thus, the application of $v_i$ to $u_i$ yields a term whose type has to be a supertype of the substitution for $t_i$, while the application of $v_i$ to a term $N$, forces the type of $N$ to be a subtype of the substitution of $t_i$. Using these two ideas, the terms $[i]$ are then so defined as to enforce the subtype relations implied by the inequality $i$.

$$\begin{array}{lcl}
[p \leq t_i] & = & v_i\, c_p \\
[t_i \leq t_j] & = & v_j\, (v_i\, u_i) \\
[t_i \leq t_j \to t_k] & = & v_k\, ((v_i\, u_i)\, (v_j\, u_j)) \\
[t_j \to t_k \leq t_i] & = & v_i\, (\lambda x.\, K (v_k\, u_k)\, (v_j\, x))
\end{array}$$

where $K$ is the combinator $(\lambda x.\, (\lambda y.\, x))$.

Finally, we produce the following term

$$\begin{array}{lcl}
M & = & \lambda u_1 \ldots \lambda u_n. \\
 & & (\lambda v_1 \ldots \lambda v_n. \\
 & & (\lambda x.\, [i_1])((\lambda x.\, [i_2])(\cdots ((\lambda x.\, [i_{m-1}])[i_m]) \cdots)) \\
 & & )\, \underbrace{(\lambda x.\, x) \cdots (\lambda x.\, x)}_{n \text{ times}}
\end{array}$$

Abstracting the variables $v_i$ and applying to the identity functions $\lambda x.\, x$ forces each variable $v_i$ to be of the desired function type that was assumed in the construction of the terms $[i]$.

We now prove that $M$ is typable iff $\mathcal{I}$ is satisfiable.

If $\mathcal{I}$ is satisfiable by substitution $\varphi_{\mathcal{I}}$, it can be easily seen that $M$ can be typed by using type $\varphi_{\mathcal{I}}(t_j)$ for $u_j$ and $\varphi_{\mathcal{I}}(t_j) \to \varphi_{\mathcal{I}}(t_j)$ for $v_j$.

For the converse, assume that $M$ is typable. By our previous comments, consider a derivation $\mathcal{D}$ of $M$ in which all type occurrences are ground types and in which *(subtp)* is used only after *(var)* and *(const)*.

Note that if $\vdash_{ST_{\preceq}} \Gamma \triangleright \lambda x.\, x : \tau$ then $\tau = \tau_l \to \tau_r$ where $\tau_l \preceq_P \tau_r$. By the following derivation fragment in which there are no *(subtp)* occurrences after *(app)* or *(abs)*

$$\dfrac{\dfrac{\cdots, v : \tau \triangleright (\qquad) : \sigma}{\cdots \triangleright \lambda v.\, (\qquad) : \tau \to \sigma} \qquad \cdots \triangleright \lambda x.\, x : \tau}{\cdots \triangleright (\lambda v.\, (\qquad))(\lambda x.\, x) : \sigma}$$

it follows by constructing $\mathcal{D}$ backwards that $\mathcal{D}$ must have a line of the form

$$\Gamma_{uv} \triangleright (\lambda x.\, [i_1])((\lambda x.\, [i_2]) \cdots) : \sigma$$

where $\Gamma_{uv} = \{u_1 : \rho_1, \ldots, u_n : \rho_n, v_1 : \sigma_1 \to \tau_1, \ldots, v_n : \sigma_n \to \tau_n\}$ for some types $\rho_1, \ldots, \rho_n, \sigma_1, \ldots, \sigma_n, \tau_1, \ldots, \tau_n$ with $\sigma_i \preceq_P \tau_i$ and $\sigma_i, \tau_i \in T_P$.
Take $\varphi$ to be any substitution with $\sigma_i \preceq_P \varphi(x_i) \preceq_P \tau_i$, such a $\varphi$ exists since $\sigma_i, \tau_i \in T_P$. We prove that $\varphi$ is a satisfying substitution for $\mathcal{I}$. By constructing $\mathcal{D}$ backwards we also see that $\Gamma_{uv} \triangleright [i_l] : \sigma'_l$ for some $\sigma'_l$. We prove that $\varphi$ as defined above will then satisfy $i_l$. We only show two representative cases:

**Case** $i_l \equiv t_i \leq t_j \to t_k$**:** Then $[i_l] = (v_k\, ((v_i\, u_i)\, (v_j\, u_j)))$. Since $\Gamma_{uv} \triangleright [i_l] : \sigma'_l$, we must have occurring in $\mathcal{D}$

$$\Gamma_{uv} \triangleright v_k : \delta_2 \to \sigma'_l \tag{3}$$

$$\Gamma_{uv} \triangleright ((v_i\, u_i)(v_j\, u_j)) : \delta_2 \tag{4}$$

from (4) we must have that

$$\Gamma_{uv} \triangleright (v_i\, u_i) : \delta_1 \to \delta_2 \tag{5}$$

$$\Gamma_{uv} \triangleright (v_j\, u_j) : \delta_1 \tag{6}$$

By Lemma 4.2, $\tau_i \preceq_P \delta_1 \to \delta_2$, $\tau_j \preceq_P \delta_1$. Thus $\tau_i = s_i \to w_i$ and $\delta_1 \preceq_P s_i$ and $w_i \preceq_P \delta_2$. Since $\Gamma(v_k) = \sigma_k \to \tau_k$ it follows that $\sigma_k \to \tau_k \preceq_P \delta_2 \to \sigma'_l$, *i.e.*, $\delta_2 \preceq_P \sigma_k$.
Thus $\tau_i = s_i \to w_i$ with $\tau_j \preceq_P s_i$ and $w_i \preceq_P \sigma_k$, *i.e.*, $\varphi(t_i) \preceq_P \tau_i \preceq_P \tau_j \to \sigma_k \preceq_P \varphi(t_j) \to \varphi(t_k)$. Hence, $\varphi(t_i) \preceq_P \varphi(t_j \to t_k)$.

**Case** $i_l \equiv t_j \to t_k \le t_i$**:** Then $[i_l] = v_i N$, where $N = (\lambda x.\, K(v_k\, u_k)\,(v_j\, x))$.
Then we must have occurring in $\mathcal{D}$

$$\Gamma_{uv}, x{:}\,\sigma_x \triangleright v_k\, u_k{:}\,\delta_1 \tag{7}$$
$$\Gamma_{uv}, x{:}\,\sigma_x \triangleright v_j\, x{:}\,\delta_2 \tag{8}$$
$$\Gamma_{uv} \triangleright N{:}\,\sigma_x \to \delta_3 \tag{9}$$

where $\delta_1 \preceq_P \delta_3$, since if $\vdash_{ST_\preceq} \Gamma \triangleright K{:}\,\tau$ then $\tau = \tau_l \to \tau_l' \to \tau_r$ with $\tau_l \preceq_P \tau_r$.

By Lemma 4.2, from (8) we get that $\sigma_x \preceq_P \sigma_j$ and from (7) we get that $\tau_k \preceq_P \delta_1$. Then by similar reasoning as before since $\vdash_{ST_\preceq} \Gamma_{uv} \triangleright v_i N{:}\cdots$ we get that $\sigma_x \to \delta_3 \preceq_P \sigma_i$, *i.e.*,

$$\begin{array}{rcl} \varphi(t_j \to t_k) & = & \varphi(t_j) \to \varphi(t_j) \\ & \preceq_P & \sigma_j \to \tau_k \\ & \preceq_P & \sigma_x \to \delta_1 \\ & \preceq_P & \sigma_x \to \delta_3 \\ & \preceq_P & \sigma_i \\ & \preceq_P & \varphi(t_i) \end{array}$$

■

**Proof of Theorem 5.10** Consider any $k \ge 0$. Take

$$M \;=\; z\, f_1(f_1\, x_1) \cdots f_{k+1}(f_{k+1}\, x_{k+1})$$

We will prove that $M$ cannot have a most general typing whose coercion set has at most $k$ elements. Consider any provable typing $C, \Gamma \triangleright M{:}\,\sigma$ with $|C| \le k$. We will exhibit a provable typing $C', \Gamma' \triangleright M{:}\,\sigma'$ such that $C, \Gamma \triangleright M{:}\,\sigma \not\succ C', \Gamma' \triangleright M{:}\,\sigma'$.

Let $\rho_i$ and $\rho_i'$ be the typing assumptions for $x_i$ and $f_i$ respectively, in $\Gamma$, *i.e.*, let $\rho_i = \Gamma(x_i)$, $\rho_i' = \Gamma(f_i)$ for $i = 1, \ldots, k+1$. Then, define

$$\begin{array}{rcll} \Gamma_i & = & \{x_i{:}\,\rho_i,\, f_i{:}\,\rho_i'\}, & i = 1, \ldots, k+1 \\ S_i & = & FTV(C) \cap FTV(\Gamma_i), & i = 1, \ldots, k+1 \end{array}$$

**Case I:** For some $j$, $1 \le j \le k+1$, $S_j = \phi$. Take

$$\begin{array}{rcl} C' & = & \displaystyle\bigcup_{1 \le i \le k+1} \{s_i \preceq t_i, u_i \preceq t_i, u_i \preceq v_i\} \\ \Gamma' & = & \{z{:}\, v_1 \to v_2 \to \cdots \to v_{k+1} \to w\} \;\cup \\ & & \displaystyle\bigcup_{1 \le i \le k+1} \{x_i{:}\, s_i,\, f_i{:}\, t_i \to u_i\} \\ \sigma' & = & w \end{array}$$

Since $\vdash_{ST_\preceq} C, \Gamma \triangleright M{:}\,\sigma$ and $C$ is atomic, using Lemma 2.2 there is a proof $\mathcal{D}$ as in Table 1.

Suppose that $S_j = \phi$. From the proof $\mathcal{D}$, we have that $\vdash_{ST_\preceq} C, \Gamma \triangleright f_j(f_j\, x_j){:}\,\sigma_j$. Using Lemma 2.1, we have that $\vdash_{ST_\preceq} C, \Gamma_j \triangleright f_j(f_j\, x_j){:}\,\sigma_j$. By Lemma 5.8(ii), $\vdash_{ST_\preceq} \mathit{Clos}(C), \Gamma_j \triangleright f_j(f_j\, x_j){:}\,\sigma_j$. By Lemma 5.7(i) and Lemma 5.9(ii),

$$\vdash_{ST_\preceq} \mathit{Clos}(C)\lceil_{FTV(\Gamma_j)}, \Gamma_j \triangleright f_j(f_j\, x_j){:}\,\sigma_j.$$

Now

$$\mathit{Clos}(C)\lceil_{FTV(\Gamma_j)} = \mathit{Clos}(C)\lceil_{FTV(\Gamma_j) \cap FTV(\mathit{Clos}(C))}$$

By Lemma 5.7(ii),

$$FTV(\Gamma_j) \cap FTV(\mathit{Clos}(C)) = \phi.$$

Hence,

$$\mathit{Clos}(C)\lceil_{FTV(\Gamma_j)} = \phi.$$

Thus,

$$\vdash_{ST_\preceq} \phi, \Gamma_j \triangleright f_j(f_j\, x_j){:}\,\sigma_j$$

Hence, it must be that

$$\rho_j = \sigma_j, \quad \rho_j' = \sigma_j \to \sigma_j \tag{10}$$

Consider the term

$$\begin{array}{rcl} N & = & z\,(f_1(f_1\, x_1)) \cdots \\ & & (f_{j-1}(f_{j-1}\, x_{j-1}))\; x_j \;\cdots \\ & & (f_{k+1}(f_{k+1}\, x_{k+1})) \end{array}$$

From (10), we know that $\vdash_{ST_\preceq} C, \Gamma \triangleright x_j{:}\,\sigma_j$. Plugging this proof instead of $C, \Gamma \triangleright f_j(f_j\, x_j){:}\,\sigma_j$ in the derivation $\mathcal{D}$, we get that $\vdash_{ST_\preceq} C, \Gamma \triangleright N{:}\,\sigma$. But, we cannot have that $\vdash_{ST_\preceq} C', \Gamma' \triangleright N{:}\,\sigma'$ since $C' \not\vdash s_j \preceq v_j$ and thus $N$ is not typable under assumptions $C', \Gamma'$. Thus, $\not\vdash_{ST_\preceq} C', \Gamma' \triangleright N{:}\,\sigma'$. Since $\succ$ is sound,

$$\begin{array}{rl} & C, \Gamma \triangleright N{:}\,\sigma \not\succ C', \Gamma' \triangleright N{:}\,\sigma' \\ \Rightarrow & C, \Gamma \triangleright M{:}\,\sigma \not\succ C', \Gamma' \triangleright M{:}\,\sigma' \end{array}$$

**Case II:** $S_j \ne \phi$, for $j = 1, \ldots, k+1$. Consider

$$\begin{array}{rcl} C' & = & \phi \\ \Gamma' & = & \{z{:}\, p_1 \to p_2 \to \cdots \to p_{k+1} \to w\} \;\cup \\ & & \displaystyle\bigcup_{1 \le i \le k+1} \{x_i{:}\, p_i,\, f_i{:}\, p_i \to p_i\} \\ \sigma' & = & w \end{array}$$

Consider the sets $C\lceil_{S_1}, \ldots, C\lceil_{S_{k+1}}$. Then they cannot all be pairwise disjoint. For, if they were, we have

$$\begin{array}{rcll} |C\lceil_{S_1} \cup \cdots \cup C\lceil_{S_{k+1}}| & = & |C\lceil_{S_1}| + \cdots + |C\lceil_{S_{k+1}}| & \\ & \ge & 1 + \cdots + 1 & (S_i \ne \phi) \\ & = & k+1 & \end{array}$$

But, $C\lceil_{S_1} \cup \cdots \cup C\lceil_{S_{k+1}} \subseteq C$, $|C| \le k$. Hence, there exist $i, j$ such that $C\lceil_{S_i} \cap C\lceil_{S_j} \ne \phi, i \ne j$.

**Claim A.1** *There exist type variables $s_i, s_j, u$ such that $s_i \in FTV(\Gamma_i), s_j \in FTV(\Gamma_j)$ and $C \vdash s_i \preceq u, C \vdash s_j \preceq u$.*

**Proof** Since $C\lceil_{S_i} \cap C\lceil_{S_j} \ne \phi$, let $s \preceq t \in C\lceil_{S_i} \cap C\lceil_{S_j}$. Then $s$ or $t$ is in $S_i$ and $s$ or $t$ is in $S_j$.

- $s \in S_i, t \in S_j$. Take $s_i = s, s_j = t, u = t$.
- $t \in S_i, s \in S_j$. Take $s_i = t, s_j = s, u = t$.
- $s \in S_i, s \in S_j$. Take $s_i = s_j = u = s$.
- $t \in S_i, t \in S_j$. Take $s_i = s_j = u = t$.

■

Let $\rho_l = \alpha_l, \rho_l' = \beta_l \to \gamma_l$ for $l = i, j$. For $\delta_l = \alpha_l, \beta_l$, or $\gamma_l$, define a term $F(\delta_l)$ as follows:

$$\begin{array}{rcl} F(\alpha_l) & = & EQ\, x_l\, y_l \\ F(\beta_l) & = & f_l\, y_l \\ F(\gamma_l) & = & EQ\,(f_l\, x_l)\, y_l \end{array}$$

$$
\begin{array}{ccccc}
 & \vdots & & \vdots & \vdots \\
C, \Gamma \triangleright z\colon \rho_z & C, \Gamma \triangleright f_1 (f_1\, x_1)\colon \sigma_1 & & \vdots & \vdots \\
\hline
\multicolumn{2}{c}{C, \Gamma \triangleright z\, (f_1 (f_1\, x_1))\colon \cdots} & & C, \Gamma \triangleright f_2 (f_2\, f_2)\colon \sigma_2 & \vdots \\
\hline
 & & \ddots & & \vdots \\
 & & & \ddots & \vdots \\
 & & & & C, \Gamma \triangleright f_{k+1}(f_{k+1}\, x_{k+1})\colon \sigma_{k+1} \\
\hline
 & & & \multicolumn{2}{c}{C, \Gamma \triangleright M\colon \sigma}
\end{array}
$$

Table 1: Derivation $\mathcal{D}$ of Theorem 5.10

The term $EQ$ is defined as

$$EQ = \lambda x.\, \lambda y.\, \lambda w.\, K\, w\, (\lambda z.\, \lambda w.\, w\, (z\, x)\, (z\, y))$$

which ensures that $EQ\, M\, N$ is typable iff $M$ and $N$ can be given the same type. Then we can make the following observations:

(i) $\vdash_{ST_{\preceq}} C, \Gamma \cup \{y_l\colon \delta_l\} \triangleright F(\delta_l)\colon \tau$ for some type $\tau$, *i.e.*, $F(\delta_l)$ is typable under the assumptions $C, \Gamma \cup \{y_l\colon \delta_l\}$.

(ii) Suppose that $\vdash_{ST_{\preceq}} C', \Gamma'' \triangleright F(\delta_l)\colon \tau$ and $\Gamma'' \supset \Gamma'$. Then $\Gamma''(y_l) = p_l$.

**Case IIa:** One of $\alpha_i, \alpha_j, \beta_i, \beta_j, \gamma_i, \gamma_j$ is not a type variable. Then let $\delta_l$ be the one that is not. Consider the term

$$N = K\, M\, (\lambda y_l.\, \lambda w.\, w\, F(\delta_l)\, (\lambda z.\, (y_l\, z)))$$

From Observations (i) and (ii), we get that

$$\vdash_{ST_{\preceq}} C, \Gamma \triangleright N\colon \sigma \text{ and } \not\vdash_{ST_{\preceq}} C', \Gamma' \triangleright N\colon \sigma'$$

**Case IIb:** All of $\alpha_l, \beta_l, \gamma_l$ are type variables. By Claim A.1, we have variables $s_i, s_j, u$ satisfying the required properties. We also know that $s_l = \alpha_l, \beta_l$ or $\gamma_l$ for $l = i$ or $j$ and

$$\vdash_{ST_{\preceq}} C, \Gamma \cup \{y_i\colon s_i, y_j\colon s_j\} \triangleright EQ\, y_i\, y_j\colon \tau$$

From Observations (i) and (ii), taking

$$N = K\, M\, (\lambda y_i.\, \lambda y_j.\, \lambda w.\, w\, F(s_i)\, F(s_j)\, (EQ\, y_i\, y_j))$$

we have that

$$\vdash_{ST_{\preceq}} C, \Gamma \triangleright N\colon \sigma \text{ but } \not\vdash_{ST_{\preceq}} C', \Gamma' \triangleright N\colon \sigma'$$

Hence, in either case

$$C, \Gamma \triangleright M\colon \sigma \not\preceq C', \Gamma' \triangleright M\colon \sigma'$$

Taking the term $M_k = z\, f_1(f_1\, x_1) \cdots f_{k+1}(f_{k+1}\, x_{k+1})$, we see that $|M_k| = O(k)$. ∎