Completely Bounded Quantification is Decidable

Dinesh Katiyar Sriram Sankar*
Stanford University
California, USA

Abstract

This paper proves the decidability of subtyping for $F_{\leq}$ when the bounds on polymorphic types do not contain *Top* (*i.e.*, in all types of the form $\forall\alpha{<:}\tau_1.\tau_2$, τ_1 does not contain *Top*). This general restriction is subsequently relaxed to allow unbounded quantification.

1 Introduction

$F_{\leq}$ [CW85,CG] is a typed λ-calculus with subtyping and bounded second-order polymorphism. The importance of $F_{\leq}$ in programming language design is that it provides a simple context for studying the typing problems that arise when subtyping and bounded quantification are added to polymorphic languages such as ML.

Curien and Ghelli [CG] recently developed a subtyping algorithm for $F_{\leq}$ and proved its partial correctness. Subsequently, Ghelli [Ghe90] presented a termination proof for this algorithm. A mistake was discovered in this termination proof, following which Pierce [Pie92] presented a proof showing that the subtyping problem for general $F_{\leq}$ types is undecidable.

This paper shows how one can make the subtyping problem decidable by imposing some restrictions on $F_{\leq}$ types. We first prove the termination of Curien and Ghelli's algorithm when the bounds on all polymorphic types involved do not contain *Top*. *i.e.*, In all types of the form $\forall\alpha <: \tau_1.\tau_2$, τ_1 does not contain *Top*. Such a bound completely determines the structure of α, hence we refer to this as "completely bounded". We later show that this restriction can be relaxed to allow unbounded quantification. Adding records and unions to our system causes it to become undecidable. We are currently working on defining subset restrictions for records and unions similar to those presented in this paper to make subtype checking decidable in the presence of these types.

We are in the process of designing a type system based on $F_{\leq}$ for a prototyping language called *Rapide* [BL90,MMM91], and are implementing a subtyping algorithm for this type system. Given the undecidability of subtyping for general $F_{\leq}$ types, we need to restrict our type system so as to

*ERL449, Computer Systems Laboratory, Stanford University, Stanford, California - 94305. *phone:* (415)723-1835. *email:* sankar@cs.stanford.edu.

make subtyping decidable. Results such as those presented in this paper will aid in determining the necessary restrictions.

In Section 2, we present Curien and Ghelli's algorithm. We prove the termination of this algorithm for completely bounded quantification in Section 3. Section 4 shows how we can relax our restrictions to allow unbounded quantification. Section 5 presents examples of records and unions that cannot be handled by simple extensions of Curien and Ghelli's algorithm. Section 6 concludes the paper by describing plans for future work.

2 Curien and Ghelli's algorithm

A $F_{\leq}$ type τ is either a simple type (such as *Int*), a variable, a function type $(\tau_1 \to \tau_2)$, or a polymorphic type $(\forall\alpha{<:}\tau_1.\tau_2)$. Given a list of assumptions Γ and two types σ and τ, the subtyping problem is to determine whether or not $\Gamma \vdash \sigma <: \tau$ — *i.e.*, whether or not σ is a subtype of τ given the assumptions in Γ. The assumptions in Γ will all be of the form $\alpha <: \tau$, where α is a type variable and τ is a type. (This convention — that α's refer to type variables and σ's and τ's refer to types — is used for the rest of the paper. We shall also use α to refer to simple types — the context will make it clear whether a particular α is a variable or a simple type.) τ in this case is called the *bound* of α and is referred to as $\Gamma(\alpha)$. Free variables in τ may only be *bounded* in other assumptions in Γ to the left (earlier in the list) of the assumption containing τ.

Curien and Ghelli's algorithm is presented as a list of axiom schemas and inference rules. These schemas and rules contain templates of subtyping problems. The algorithm proceeds by applying the inference rules backwards to the subtyping problem. If the subtyping problem matches the template below the line of an inference rule, it reduces to subtyping problems that can be derived from the templates above the line of the inference rule. If the subtyping problem matches the template of an axiom schema, the algorithm reports success. In all other cases, the algorithm reports failure.

The axiom schemas and inference rules are listed below:

(NTop) $$\Gamma \vdash \sigma <: \mathit{Top}$$

(NRefl) $$\Gamma \vdash \alpha <: \alpha$$

(NVar) $$\frac{\Gamma \vdash \Gamma(\alpha) <: \tau}{\Gamma \vdash \alpha <: \tau}$$

(NArrow) $$\frac{\Gamma \vdash \tau_1 <: \sigma_1 \qquad \Gamma \vdash \sigma_2 <: \tau_2}{\Gamma \vdash \sigma_1 \to \sigma_2 <: \tau_1 \to \tau_2}$$

(NAll) $$\frac{\Gamma \vdash \tau_1 <: \sigma_1 \qquad \Gamma, \alpha{<:}\tau_1 \vdash \sigma_2 <: \tau_2}{\Gamma \vdash \forall\alpha{<:}\sigma_1.\sigma_2 <: \forall\alpha{<:}\tau_1.\tau_2}$$

3 Proof of termination

We prove termination by defining a complexity metric which is finite and positive for each subtyping problem, and show that the application of inference rules causes the complexity of the new subtyping problems generated to decrease. Since the complexity cannot decrease indefinitely, the algorithm will have to terminate.

In 3.1, we define the complexity metric. In 3.2, we show how the complexity metric is affected by applying the various inference rules. We conclude with a condition (Theorem 1) that, if satisfied, will guarantee the termination of the subtyping algorithm. Finally, in 3.3, we show how our restrictions on bounds of polymorphic types satisfies the condition of Theorem 1.

3.1 The complexity metric

We define the *size* of a type τ with respect to a list of assumptions Γ, and refer to this as $size(\tau)_\Gamma$. The *complexity* of a subtyping problem $\Gamma \vdash \sigma <: \tau$ is defined as:

$$complexity(\Gamma \vdash \sigma <: \tau) \stackrel{\text{def}}{=} size(\sigma)_\Gamma + size(\tau)_\Gamma$$

$size(\tau)_\Gamma$ is determined by recursively replacing variables in τ with their respective bounds and then computing the textual size of the resulting expression. $size(\tau)_\Gamma$ is formally defined as:

$$\begin{array}{rcl} size(Top)_\Gamma & = & 1 \\ size(\alpha)_\Gamma & = & \begin{cases} size(\Gamma(\alpha))_\Gamma & \text{if } \Gamma(\alpha) \text{ is defined} \\ 1 & \text{otherwise} \end{cases} \\ size(\tau_1 \to \tau_2)_\Gamma & = & size(\tau_1)_\Gamma + size(\tau_2)_\Gamma \\ size(\forall \alpha{<:}\tau_1.\tau_2)_\Gamma & = & size(\tau_1)_\Gamma + size(\tau_2)_{\Gamma,\alpha<:\tau_1} \end{array}$$

Examples:

1. $size(Top \to \alpha)_{\alpha <: Int \to Int}$
$= size(Top)_{\alpha <: Int \to Int} + size(\alpha)_{\alpha <: Int \to Int}$
$= 1 + size(Int \to Int)_{\alpha <: Int \to Int}$
$= 1 + size(Int)_{\alpha <: Int \to Int} + size(Int)_{\alpha <: Int \to Int}$
$= 1 + 1 + 1 = 3.$

2. $size(\forall \alpha_1 <: (\alpha_2 \to Int).\alpha_1)_{\alpha_2 <: Int \to Int}$
$= size(\alpha_2 \to Int)_{\alpha_2 <: Int \to Int} + size(\alpha_1)_{\alpha_2 <: Int \to Int, \alpha_1 <: \alpha_2 \to Int}$
$= size(\alpha_2)_{\alpha_2 <: Int \to Int} + size(Int)_{\dots} + size(\alpha_2 \to Int)_{\alpha_2 <: Int \to Int, \alpha_1 <: \alpha_2 \to Int}$
$= size(Int \to Int)_{\dots} + 1 + size(\alpha_2)_{\alpha_2 <: Int \to Int, \alpha_1 <: \alpha_2 \to Int} + size(Int)_{\dots}$
$= 2 + 1 + size(Int \to Int)_{\dots} + 1$
$= 2 + 1 + 2 + 1 = 6.$

Lemma 1 *$size(\tau)_\Gamma$ is always finite and positive.*

Proof. It is obvious from the definition of *size* that it has to be positive. We prove that it is finite by showing that evaluation of *size* terminates for all τ and Γ. A complexity metric similar to that used in Ghelli's flawed termination proof actually works in this case.

The complexity metric to prove the termination of the evaluation of $size(\tau)_\Gamma$ is obtained by first ordering all the variables that occur in τ and Γ such that the following property is satisfied: If α_i is defined in the bound of α_j, then α_i occurs to the left of α_j in the ordering. It is possible to obtain such an ordering given the structure of $F_\leq$ (there may be multiple orderings that satisfy this condition in which case, one of them is chosen arbitrarily). The *depth* of each variable is then defined as the number of variables that occur to the left of it in this ordering.

The complexity of any subproblem $size(\tau')_{\Gamma'}$ that arises during the evaluation of $size(\tau)_\Gamma$ is the tuple $\langle D, S\rangle$, where D is the maximum depth over all the variables that occur in τ' and S is the textual length of τ'. There may be variables in τ' and Γ' that do not occur in either of τ or Γ. These variables are created when an existing variable is duplicated in the reduction process, thus requiring one of the uses to be renamed. The depth of the renamed variable is defined to be the same as its unrenamed counterpart. The key to this proof is that defining the depth of renamed variables in this manner maintains the condition based on which the initial ordering was created.

It is easy to see that the complexity decreases (the ordering between $\langle D, S\rangle$ tuples is lexicographic) during the evaluation of $size(\tau)_\Gamma$. For all reductions other than $size(\alpha)_\Gamma = size(\Gamma(\alpha))_\Gamma$, the D component of the complexity metric either remains the same or decreases while the S component decreases; whereas in the abovementioned reduction, the D component decreases.

Since this complexity metric cannot decrease indefinitely, the evaluation of *size* for any τ and Γ has to terminate. $\square$

3.2 The effect of inference rule application on complexity

NVar

$$complexity(\Gamma \vdash \alpha <: \tau) = size(\alpha)_\Gamma + size(\tau)_\Gamma = size(\Gamma(\alpha))_\Gamma + size(\tau)_\Gamma = complexity(\Gamma \vdash \Gamma(\alpha) <: \tau)$$

i.e., The complexity metric remains the same after application of the inference rule NVar. However, NVar may be applied continuously at most as many times as there are variables in Γ before one of the other rules has to be applied. The complexity metric reduces when any of the other rules are applied, so there is no problem.

NArrow

It is quite obvious that $complexity(\Gamma \vdash \tau_1 <: \sigma_1)$ and $complexity(\Gamma \vdash \sigma_2 <: \tau_2)$ are both less than $complexity(\Gamma \vdash \sigma_1 \to \sigma_2 <: \tau_1 \to \tau_2)$.

NALL

It is quite obvious that $complexity(\Gamma \vdash \tau_1 <: \sigma_1)$ is less than $complexity(\Gamma \vdash \forall\alpha{<:}\sigma_1.\sigma_2 <: \forall\alpha{<:}\tau_1.\tau_2)$. We shall now simplify $complexity(\Gamma \vdash \forall\alpha{<:}\sigma_1.\sigma_2 <: \forall\alpha{<:}\tau_1.\tau_2) - complexity(\Gamma, \alpha{<:}\tau_1 \vdash \sigma_2 <: \tau_2)$ to determine the conditions under which the complexity metric decreases when reducing to the second rule above the line in NALL. This expression, which must be positive for the complexity metric to decrease, simplifies as follows:

$$
\begin{aligned}
&complexity(\Gamma \vdash \forall\alpha{<:}\sigma_1.\sigma_2 <: \forall\alpha{<:}\tau_1.\tau_2) - complexity(\Gamma, \alpha{<:}\tau_1 \vdash \sigma_2 <: \tau_2)\\
&= (size(\forall\alpha <: \sigma_1.\sigma_2)_\Gamma + size(\forall\alpha <: \tau_1.\tau_2)_\Gamma) - (size(\sigma_2)_{\Gamma,\alpha <: \tau_1} + size(\tau_2)_{\Gamma,\alpha <: \tau_1})\\
&= (size(\sigma_1)_\Gamma + size(\sigma_2)_{\Gamma,\alpha <: \sigma_1} + size(\tau_1)_\Gamma + size(\tau_2)_{\Gamma,\alpha <: \tau_1}) - (size(\sigma_2)_{\Gamma,\alpha <: \tau_1} + size(\tau_2)_{\Gamma,\alpha <: \tau_1})\\
&= size(\sigma_1)_\Gamma + size(\tau_1)_\Gamma + (size(\sigma_2)_{\Gamma,\alpha <: \sigma_1} - size(\sigma_2)_{\Gamma,\alpha <: \tau_1})
\end{aligned}
$$

We concentrate on the part $size(\sigma_2)_{\Gamma,\alpha <: \sigma_1} - size(\sigma_2)_{\Gamma,\alpha <: \tau_1}$ from the last line above. If this expression is non-negative, then the overall expression will be positive, and therefore the complexity metric will decrease on the application of NALL. We need to present the following lemma before we can proceed further.

Lemma 2 *For any types σ, τ, and for any list of assumptions Γ, such that the variable α is not bounded in any of them, and also does not occur anywhere in Γ, $size(\sigma)_{\Gamma,\alpha <: \tau} = size(\sigma)_\Gamma + n(size(\tau)_\Gamma - 1)$, where $n \geq 0$ and depends only on σ.*

The evaluation of $size(\sigma)_{\Gamma'}$ (for any Γ') will reduce to zero or more evaluations of the form $size(\alpha)_{\Gamma''}$ in addition to other reductions. Reductions to the form $size(\alpha)_{\Gamma''}$ may be either due to occurrences of α in σ, or occurrences of α in bounds in Γ' which are used to replace the variables they bound. If Γ' does not contain any bound that contains α, then the number of times $size(\sigma)_{\Gamma'}$ reduces to the form $size(\alpha)_{\Gamma''}$ depends only on σ. Suppose this number is n.

Therefore, $size(\sigma)_\Gamma$ reduces to n evaluations of the form $size(\alpha)_{\Gamma''}$, each of which evaluates to 1 since α is not bounded anywhere. So we can write $size(\sigma)_\Gamma$ as $m + n$ where m is the result of the evaluation of the remainder of the reductions.

Similarly, $size(\sigma)_{\Gamma,\alpha <: \tau}$ reduces to n evaluations of the form $size(\alpha)_{\Gamma''}$, while the remainder of the reductions evaluate to m. Each reduction to $size(\alpha)_{\Gamma''}$ further reduces to $size(\tau)_{\Gamma''}$. Since Γ'' is of the form $\Gamma, \alpha <: \tau, \ldots,$ τ does not depend on the portion of Γ'' to the right of Γ. Therefore, $size(\tau)_{\Gamma''} = size(\tau)_\Gamma$. So we can write $size(\sigma)_{\Gamma,\alpha <: \tau}$ as $m + n(size(\tau)_\Gamma)$, which is the same as $size(\sigma)_\Gamma + n(size(\tau)_\Gamma - 1)$. $\square$

We now simplify $size(\sigma_2)_{\Gamma,\alpha <: \sigma_1} - size(\sigma_2)_{\Gamma,\alpha <: \tau_1}$.

$$
\begin{aligned}
&size(\sigma_2)_{\Gamma,\alpha <: \sigma_1} - size(\sigma_2)_{\Gamma,\alpha <: \tau_1}\\
&= (size(\sigma_2)_\Gamma + n(size(\sigma_1)_\Gamma - 1)) - (size(\sigma_2)_\Gamma + n(size(\tau_1)_\Gamma - 1)) \qquad \text{(for some } n \geq 0\text{)}\\
&= n(size(\sigma_1)_\Gamma - size(\tau_1)_\Gamma)
\end{aligned}
$$

We are now ready to present the first of our main results.

Theorem 1 *During the execution of Curien and Ghelli's algorithm, if* $size(\sigma_1)_\Gamma \geq size(\tau_1)_\Gamma$ *(where* σ_1*,* τ_1*, and* Γ *are as defined in* NALL*) is true every time* NALL *is used to reduce a subtyping problem to the subtyping problem derived from the right template above the line* $(\Gamma, \alpha{<:}\tau_1 \vdash \sigma_2 <: \tau_2)$*, then the algorithm will terminate.*

Proof. Obvious from the results of this section. □

3.3 Subset restrictions to guarantee termination

When applying NALL on a subtyping problem, we shall require that we first consider the subtyping problem derived from $\Gamma \vdash \tau_1 <: \sigma_1$ (the left template above the line in NALL). Only if the algorithm terminates successfully on this problem do we consider the subtyping problem derived from $\Gamma, \alpha{<:}\tau_1 \vdash \sigma_2 <: \tau_2$ (the right template above the line in NALL). Hence, we can assume that $\Gamma \vdash \tau_1 <: \sigma_1$ when the algorithm is applied on the subtyping problem derived from $\Gamma, \alpha{<:}\tau_1 \vdash \sigma_2 <: \tau_2$. Assuming this, we have the following corollary to Theorem 1.

Corollary 1 *For every* τ*,* σ *that are bounds of polymorphic types and for every list of assumptions* Γ*, if* $\Gamma \vdash \tau <: \sigma \Rightarrow size(\tau)_\Gamma \leq size(\sigma)_\Gamma$*, then Curien and Ghelli's algorithm will terminate.*

Lemma 3 *If* $size(\tau_1)_\Gamma = size(\tau_2)_\Gamma$*, then, for any* σ*,* $size(\sigma)_{\Gamma,\alpha <: \tau_1} = size(\sigma)_{\Gamma,\alpha <: \tau_2}$*.*

Proof. This follows trivially from Lemma 2. $size(\sigma)_{\Gamma,\alpha <: \tau_1} = size(\sigma)_\Gamma + n(size(\tau_1)_\Gamma - 1) = size(\sigma)_\Gamma + n(size(\tau_2)_\Gamma - 1) = size(\sigma)_{\Gamma,\alpha <: \tau_2}$. □

We are now ready to present another key result, Theorem 2, that if the subset restrictions mentioned in Section 1 are satisfied, then the condition of Corollary 1 will be satisfied. A straightforward consequence of this is that if these subset restrictions are met, then Curien and Ghelli's algorithm will terminate, and hence completely bounded quantification is decidable.

Theorem 2 *For all types* τ*,* σ *that do not contain Top, and for any list of assumptions* Γ*, if* $\Gamma \vdash \tau <: \sigma$*, then* $size(\tau)_\Gamma = size(\sigma)_\Gamma$*.*

Proof. If $\Gamma \vdash \tau <: \sigma$, then there must be a proof $\Gamma^1 \vdash \tau^1 <: \sigma^1, \Gamma^2 \vdash \tau^2 <: \sigma^2, \ldots, \Gamma^n \vdash \tau^n <: \sigma^n$ where $\Gamma^n = \Gamma$, $\tau^n = \tau$, and $\sigma^n = \sigma$, and each $\Gamma^i \vdash \tau^i <: \sigma^i$ is either of the form of NREFL, or obtained from earlier steps in the proof using one of the rules NVAR, NARROW, or NALL. Note that NTOP will not be used in such a proof since *Top* does not occur in σ and τ.

We prove by induction that for all i $(1 \leq i \leq n)$, $size(\tau^i)_{\Gamma^i} = size(\sigma^i)_{\Gamma^i}$. The induction hypothesis is that for all j $(1 \leq j < i)$, $size(\tau^j)_{\Gamma^j} = size(\sigma^j)_{\Gamma^j}$. Assuming the induction hypothesis, we prove $size(\tau^i)_{\Gamma^i} = size(\sigma^i)_{\Gamma^i}$. There are four cases to consider:

1. $\Gamma^i \vdash \tau^i <: \sigma^i$ is of the form of NREFL. *i.e.*, $\tau^i = \sigma^i$. Therefore, $size(\tau^i)_{\Gamma^i} = size(\sigma^i)_{\Gamma^i}$.

2. $\Gamma^i \vdash \tau^i <: \sigma^i$ is derived using NVar from an earlier step of the form $\Gamma^i \vdash \Gamma^i(\tau^i) <: \sigma^i$. In this case, τ^i is a variable bounded in Γ^i. Therefore, $size(\tau^i)_{\Gamma^i} = size(\Gamma^i(\tau^i))_{\Gamma^i} = size(\sigma^i)_{\Gamma^i}$.

3. $\Gamma^i \vdash \tau^i <: \sigma^i$ is derived using NArrow from earlier steps $\Gamma^k \vdash \tau^k <: \sigma^k$ and $\Gamma^l \vdash \tau^l <: \sigma^l$. Then $\Gamma^i = \Gamma^k = \Gamma^l$, $\tau^i = \sigma^k \to \tau^l$, and $\sigma^i = \tau^k \to \sigma^l$. Therefore, $size(\tau^i)_{\Gamma^i} = size(\sigma^k)_{\Gamma^i} + size(\tau^l)_{\Gamma^i} = size(\sigma^k)_{\Gamma^k} + size(\tau^l)_{\Gamma^l} = size(\tau^k)_{\Gamma^k} + size(\sigma^l)_{\Gamma^l} = size(\tau^k)_{\Gamma^i} + size(\sigma^l)_{\Gamma^i} = size(\sigma^i)_{\Gamma^i}$.

4. $\Gamma^i \vdash \tau^i <: \sigma^i$ is derived using NAll from earlier steps $\Gamma^k \vdash \tau^k <: \sigma^k$ and $\Gamma^l \vdash \tau^l <: \sigma^l$. Then $\Gamma^i = \Gamma^k$, $\Gamma^l = \Gamma^i, \alpha{<:}\tau^k$, $\tau^i = \forall\alpha{<:}\sigma^k.\tau^l$, and $\sigma^i = \forall\alpha{<:}\tau^k.\sigma^l$ for some variable α. Therefore, $size(\tau^i)_{\Gamma^i} = size(\sigma^k)_{\Gamma^i} + size(\tau^l)_{\Gamma^i,\alpha<:\sigma^k} = size(\sigma^k)_{\Gamma^k} + size(\tau^l)_{\Gamma^i,\alpha<:\tau^k} = size(\sigma^k)_{\Gamma^k} + size(\tau^l)_{\Gamma^l} = size(\tau^k)_{\Gamma^k} + size(\sigma^l)_{\Gamma^l} = size(\tau^k)_{\Gamma^i} + size(\sigma^l)_{\Gamma^i,\alpha<:\tau^k} = size(\sigma^i)_{\Gamma^i}$.
□

4 Allowing unbounded quantification

We can relax our restriction on the use of *Top* to allow unbounded quantification and still retain decidability of subtype checking. Furthermore, any variable bounded by *Top* (directly or indirectly) may be used as a bound for another variable. We refer to these variables as "*Top*-bounded".

With this relaxation, there are two kinds of bounds that we can write: (1) Types that do not contain either *Top* or *Top*-bounded variables; and (2) Types that are either *Top* or a *Top*-bounded variable.

To show that this will not cause problems, we redefine $size(Top)$ for all *Top*'s that occur as bounds of variables to be a number L that is larger than the *size* of any bounds of the first kind mentioned above. With this redefinition of *size*, it is quite easy to see that Corollary 1 continues to hold even when the bounds τ and σ are from this relaxed domain.

There are four cases to consider where τ and σ may each be either of the two kinds mentioned above. We consider each case separately:

1. τ and σ are both types not containing *Top* or *Top*-bounded variables: This case has been handled in Theorem 2.

2. τ and σ are both types that are either *Top* or a *Top*-bounded variable: Then $size(\tau) = size(\sigma) = L$.

3. τ is a type not containing *Top* or *Top*-bounded variables while σ is either *Top* or a *Top*-bounded variable: Then $size(\sigma) = L > size(\tau)$.

4. τ is either *Top* or a *Top*-bounded variable while σ is a type not containing *Top* or *Top*-bounded variables: In this case, τ cannot be a subtype of σ, so we need not consider it any further.

5 Record and union types

We have shown that one can achieve a decidable type system for a fairly unconstricted subset of $F_{\leq}$. An immediate extension that we started working on was the addition of record and union types. However, it turns out that adding either record or union types to the type system (along with their associated inference rule) makes the subtyping problem undecidable. Finding the right restrictions to allow the addition of these types is the subject of current research.

5.1 Record types

We denote the record type with fields $l_1 \ldots l_n$ having types $\tau_1 \ldots \tau_n$ respectively as $\{l_1\colon \tau_1, \ldots, l_n\colon \tau_n\}$. The inference rule for records is :

$$(\textsc{NRec}) \qquad \frac{\Gamma \vdash \sigma_1 <: \tau_1 \qquad \ldots \qquad \Gamma \vdash \sigma_n <: \tau_n}{\Gamma \vdash \{l_1\colon \sigma_1, \ldots, l_n\colon \sigma_n, l_{n+1}\colon \sigma_{n+1}, \ldots\} <: \{l_1\colon \tau_1, \ldots, l_n\colon \tau_n\}}$$

Essentially one can either add extra fields to a record type or specialize the types of existing fields to get a subtype.

As we saw earlier, the problem with the original unrestricted system was that one could have subtypes that were structurally much more complicated than the supertype and one could exploit this in creating subtyping subproblems that grew infinitely in their complexity. We prevented this finally by restricting the use of *Top*, which is what allowed a subtype to be more complicated than the supertype in the first place. However, record types provide another way of allowing a subtype to be more complex than the supertype, namely by adding extra fields. So the empty record type provides an entity conceptually similar to *Top*. We now show that one can reproduce the non-terminating example mentioned in Pierce [Pie92], in spite of the restrictions on *Top* with the use of records.

Let $\neg\tau = \tau \to b$ where b is some simple type. Note that $\neg\sigma <: \neg\tau$ iff $\tau <: \sigma$.

Let

$$\theta = \{a\colon \forall \alpha <: \{\}. \neg\{a\colon \forall \beta <: \alpha. \neg\beta\}\}$$

Now consider the subtyping problem

$$\alpha_0 <: \theta \vdash \alpha_0 <: \{a\colon \forall \alpha_1 <: \alpha_0. \neg\alpha_1\}$$

This causes the following sequence of subproblems to be generated infinitely:

$$\begin{array}{rclcl}
\alpha_0 <: \theta & \vdash & \alpha_0 & <: & \{a\colon \forall\alpha_1 <: \alpha_0.\neg\alpha_1\} \\
\alpha_0 <: \theta & \vdash & \{a\colon \forall\alpha_1 <: \{\}.\neg\{a\colon \forall\alpha_2 <: \alpha_1.\neg\alpha_2\}\} & <: & \{a\colon \forall\alpha_1 <: \alpha_0.\neg\alpha_1\} \\
\alpha_0 <: \theta, \alpha_1 <: \alpha_0 & \vdash & \neg\{a\colon \forall\alpha_2 <: \alpha_1.\neg\alpha_2\} & <: & \neg\alpha_1 \\
\alpha_0 <: \theta, \alpha_1 <: \alpha_0 & \vdash & \alpha_1 & <: & \{a\colon \forall\alpha_2 <: \alpha_1.\neg\alpha_2\} \\
\alpha_0 <: \theta, \alpha_1 <: \alpha_0 & \vdash & \alpha_0 & <: & \{a\colon \forall\alpha_2 <: \alpha_1.\neg\alpha_2\} \\
\alpha_0 <: \theta, \alpha_1 <: \alpha_0 & \vdash & \{a\colon \forall\alpha_2 <: \{\}.\neg\{a\colon \forall\alpha_3 <: \alpha_2.\neg\alpha_3\}\} & <: & \{a\colon \forall\alpha_2 <: \alpha_1.\neg\alpha_2\} \\
\alpha_0 <: \theta, \alpha_1 <: \alpha_0, \alpha_2 <: \alpha_1 & \vdash & \neg\{a\colon \forall\alpha_3 <: \alpha_2.\neg\alpha_3\} & <: & \neg\alpha_2
\end{array}$$

and so on.

This pattern of non-termination is practically identical to the one displayed in [Pie92].

5.2 Union types

Union types pose problems similar to that of record types.

Union types are written as $\{l_1 : \tau_1 + \ldots + l_n : \tau_n\}$, with the usual meaning. The inference rule for union types is fairly straightforward.

$$(\textsc{NUnion}) \qquad \frac{\Gamma \vdash \sigma_1 <: \tau_1 \qquad \ldots \qquad \Gamma \vdash \sigma_n <: \tau_n}{\Gamma \vdash \{l_1 : \sigma_1 + \ldots + l_n : \sigma_n\} <: \{l_1 : \tau_1 + \ldots + l_n : \tau_n + \ldots\}}$$

The subtype is allowed to be a union over a subset of specializations of the types in the supertype.

Since the subtype can be a union of a fewer types than the supertype, one can have types with unions in contravariant positions thereby resulting in subtypes that have more complex structures than the supertype. Again, one can exploit this to generate an example of a subtyping problem that doesn't terminate. Behavior identical to that displayed in the previous example on record types arises for the subtyping problem

$$\alpha_0 <: \theta \vdash \alpha_0 <: \neg\{a : \neg(\forall \alpha_1 <: \alpha_0 . \neg\alpha_1)\}$$

where

$$\theta = \neg\{a : \neg(\forall \alpha <: \neg\{\}.\{a : \neg(\forall \beta <: \alpha . \neg\beta)\})\}$$

6 Future work

We are currently working on various possible subset restrictions on records and unions to make subtype checking decidable in the presence of these types. Following this, we intend to extend our system with recursive types. This might involve techniques similar to those employed by Amadio and Cardelli [AC91] to study the interaction of recursive types with subtyping. A further extension would be the addition of the so-called F-bounds [CCH*89], which essentially allow the bounds of polymorphic functions to be recursive.

Acknowledgements

We would like to thank John Mitchell for many insightful discussions. We are also grateful to David Luckham, Neel Madhav, Sigurd Meldal and other members of the Programming and Verification Group at Stanford for many useful discussions on the type system for *Rapide*. The authors were supported by DARPA grant ONR N00014-90-J1232 (Sriram) and NSF grant CCR-8814921 (Dinesh).

References

[AC91] R. Amadio and L. Cardelli. Subtyping recursive types. In *Proc. 18th ACM Symp. on Principles of Programming Languages*, pages 104–118, 1991.

[BL90] F. Belz and D. C. Luckham. A new approach to prototyping Ada–based hardware/software systems. In *Proc. ACM Tri-Ada Conference*, pages 141–155, 1990.

[CCH*89] P. Canning, W. Cook, W. Hill, J. C. Mitchell, and W. Olthoff. F-bounded quantification for object-oriented programming. In *Functional Prog. and Computer Architecture*, pages 273–280, 1989.

[CG] P.-L. Curien and G. Ghelli. Coherence of subsumption. (to appear).

[CW85] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *Computing Surveys*, 17(4):471–522, 1985.

[Ghe90] G. Ghelli. *Proof Theoretic Studies about a Minimal Type System Integrating Inclusion and Parametric Polymorphism.* PhD thesis, University of Pisa, 1990.

[MMM91] J. C. Mitchell, S. Meldal, and N. Madhav. An extension of standard ML modules with subtyping and inheritance. In *Proc. 18th ACM Symp. on Principles of Programming Languages*, pages 270–278, 1991.

[Pie92] B. Pierce. Bounded quantification is undecidable. In *Proc. 19th ACM Symp. on Principles of Programming Languages*, pages 305–315, 1992.