A Decision Procedure for the Subtype Relation on Intersection Types with Bounded Variables

Benjamin C. Pierce

10 August 1989

CMU-CS-89-1693

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We introduce an extension of the intersection type discipline in which types may contain variables with upper and lower bounds, present an algorithm for deciding the subtype relation in the extended system, and prove that the algorithm is correct.

This work was sponsored in part by the Office of Naval Research, and the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976, monitored by the xxxxxxx under Contract No. N00014-84-K-0415.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Office of Naval Research, the Defense Advanced Research Projects Agency or the U.S. Government.

A Decision Procedure for the Subtype Relation on Intersection Types with Bounded Variables

Benjamin C. Pierce
CMU-CS-89-169
August 8, 1989

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

We introduce an extension of the intersection type discipline in which types may contain variables with upper and lower bounds, present an algorithm for deciding the subtype relation in the extended system, and prove that the algorithm is correct.

This work was supported in part by the Office of Naval Research and the Defense Advanced Research Projects Agency (DOD) under contract number N00014-84-K-0415. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of DARPA or the U.S. Government.

1 Introduction

The *intersection type discipline* [CDV81] is a simple, yet powerful language for describing the behavior of programs. Beyond its intrinsic theoretical interest [BCD83,CDV80], it appears to be an attractive foundation for the type systems of practical programming languages [Rey88,Rey89].

Although the usual formulations do not allow quantification over types—there is no $\forall$ type constructor—intersection types do provide a form of polymorphism. For example, the identity function

$$\lambda x.\, x$$

has among others the intersection type

$$(\mathbf{int}\to\mathbf{int})\ \&\ (\mathbf{real}\to\mathbf{real})\ \&\ (\mathbf{bool}\to\mathbf{bool}).$$

In a theoretical sense, this kind of polymorphism is extremely powerful. For example, it can be shown that every strongly normalizing λ-term has an intersection type of a particular form. However, from a practical standpoint this "finitary" polymorphism is somewhat cumbersome. Type inference for intersection types is undecidable [RdRV84], so to obtain a typechecking algorithm programs must be annotated with some explicit type information. In Forsythe [Rey88], for example, function definitions must be annotated with an exhaustive list of the types of the arguments to which the function will be applied:

$$\lambda x : \mathbf{int}, \mathbf{real}, \mathbf{bool}.\, x$$

It is natural to ask whether a type system based on the intersection type discipline can be extended to include infinitary (parametric) polymorphism as well:

$$\forall\alpha.\, \lambda x : \alpha\to\alpha.\, x$$

In fact, since intersection types are given a *subtype ordering*, it makes sense to consider *bounded quantification* [CW85], where the type parameter to a polymorphic function is constrained to lie between specified upper and lower bounds:

$$\forall(\mathbf{int} \le \alpha \le \mathbf{complex}).\, \lambda i : \alpha.\, i + 5$$

(In Cardelli and Wegner's formulation, bounded quantifiers are given only an upper bound; we give both upper and lower bounds.)

As a step toward this extension, we consider the somewhat simpler problem of adding *bounded variables* to a system of types based on the intersection types discipline. That is, we do not allow type quantification, but types may contain free variables, which are given upper and lower bounds by an auxiliary *constraint list*:

$$\lambda i : \alpha.\, i + 5 \qquad [\mathbf{int} \le \alpha \le \mathbf{complex}]$$

(Adding variables but not explicit quantifiers is reminiscent of the form of polymorphism found in ML [Mil78], where all type variables in a type expression are implicitly universally quantified at the outside.)

In designing a typechecking algorithm for a programming language with types of this form, one major difficulty lies in deciding whether a type σ is a subtype of τ—intuitively,

whether an instance of σ may be safely used wherever an instance of τ is expected. The standard formulation of intersection types provides a perfectly unambiguous definition of the subtype relation, and it is not hard to see how the definition should be extended to types with bounded variables. But the most convenient definition of this relation, as the set of all judgements of the form $\sigma \leq \tau$ that can be proved from a certain set of inference rules, is hopelessly non-effective. Although a straightforward algorithm exists for the case without bounded variables (see Section 5.1.3), it does not appear to generalize. The presence of both intersections and variables complicates the situation significantly.

This paper presents an algorithm for deciding the subtype relation on intersection types with bounded variables. Section 2 describes the type system and the subtype relation in more detail. Section 3 introduces some notational conventions and presents the formal definitions of $\leq_Q$ (the subtype relation as a set of inference rules) and le_Q (the algorithm). Section 4 gives a detailed proof of the equivalence of $\leq_Q$ and le_Q. Section 5 discusses the algorithm and proof, sketches some alternative approaches, and points out directions in which our work should be extended.

2 Types and Subtypes

This section describes the intersection type discipline and its extension with bounded variables in more detail. (See [Rey88] for a more thorough presentation.)

2.1 The Intersection Type Discipline

Intersection types are built from primitive types like **int**, **real**, and **bool**; the $\rightarrow$ (arrow) type constructor, the $\&$ ("intersection," or "conjunction") constructor; and the special type **ns** ("nonsense," "top," or ω).

The exact set of primitive types chosen does not affect any properties of the type discipline. All that matters is that there be some relation $\leq_{pr}$ defined on the primitive types and that this relation be a preorder (i.e., that it be transitive and reflexive). For example, we might choose the set {**int**, **real**, **complex**, **bool**}, with **int** $\leq_{pr}$ **real** $\leq_{pr}$ **complex** and **bool** unrelated to the other three.

The type $\sigma_1 \rightarrow \sigma_2$ describes functions from σ_1 to σ_2. The rule for subtyping of arrow types is the usual *contravariant* one:

$$\sigma_1 \rightarrow \sigma_2 \leq \tau_1 \rightarrow \tau_2 \quad \text{if} \quad \tau_1 \leq \sigma_1 \text{ and } \sigma_2 \leq \tau_2.$$

For example, **real**$\rightarrow$**int** $\leq$ **real**$\rightarrow$**real** and **complex**$\rightarrow$**real** $\leq$ **real**$\rightarrow$**real**, while **int**$\rightarrow$**int** and **real**$\rightarrow$**real** are not related by $\leq$.

The type $\sigma\&\tau$ describes values that have *both* type σ and type τ. The term "intersection types" is generally preferred over "conjunctive types," though the latter is also used. In the $\leq$ preordering, $\sigma\&\tau$ is axiomatized as a greatest lower bound of σ and τ. This leads to three rules:

$$\sigma\&\tau \leq \sigma$$
$$\sigma\&\tau \leq \tau$$
$$\sigma \leq \tau_1\&\tau_2 \quad \text{iff} \quad \sigma \leq \tau_1 \text{ and } \sigma \leq \tau_2.$$

(The left side of the third rule implies the right side by the first two rules and the transitivity of $\leq$. The other direction must be stipulated separately.)

The interaction between the subtyping relations for arrow types and those for intersection types is captured by the rule:

$$(\tau_1 \rightarrow \tau_2) \& (\tau_1 \rightarrow \tau_3) \leq_Q \tau_1 \rightarrow (\tau_2 \& \tau_3).$$

(Note that the converse of this rule follows from the rules already given.)

Finally, **ns** is a type of any expression whatsoever, including "untypeable" ones. It is a maximal element in the subtype ordering. For every type τ,

$$\tau \leq \mathbf{ns}.$$

Also, we do not distinguish between untypeable values and functions whose result is untypeable:

$$\mathbf{ns} \leq \tau \rightarrow \mathbf{ns}.$$

2.2 Bounded Variables

Adding bounded variables to the system just described is actually quite easy. We simply assume the existence of a countable set of *type variables* $\{\alpha, \beta, \ldots\}$. Whenever we write down a type involving variables, we do so in the context of a list of upper and lower bounds (constraints) for the variables:

$$\alpha \rightarrow (\mathbf{int} \& \beta) \qquad [\mathbf{int} \leq \alpha \leq \mathbf{complex},\ (\mathbf{int} \rightarrow \mathbf{int}) \leq \beta \leq (\mathbf{int} \rightarrow \mathbf{ns})].$$

Such a type represents a *set* of types (generally infinite) in which the variables are instantiated with all types satisfying the given constraints. To ensure that this makes sense, we stipulate that the bounds on a variable α must not mention α (directly or indirectly).

We extend the subtype relation to a family of relations indexed by constraint lists, writing "σ is a subtype of τ under constraints Q" as $\sigma \leq_Q \tau$.

The subtyping rules for type variables are the expected ones: under a given set of constraints in which a variable α is bound, α is a supertype of its lower bound and a subtype of its upper bound. (Of course, like any type, α is also a subtype of itself.)

Extending a constraint list with bindings for fresh variables should not alter the subtype relation for types involving only existing variables. In particular, if $\sigma \leq \tau$ does not hold under constraints Q, it should never be the case that $\sigma \leq \tau$ under constraints $Q' \supseteq Q$. To prevent this, we restrict our attention to "well-formed" constraint lists, in which each variable's lower bound is a subtype of its upper bound (under the portion of the constraint list in question giving bindings for the variables mentioned in the upper and lower bounds).

A less important restriction is that constraint lists are only allowed to list a given variable once. To constrain a variable α to be less than both σ_1 and σ_2, we use the constraint $\alpha \leq \sigma_1 \& \sigma_2$. Similarly, although there is no explicit "least upper bound" type constructor, it can be shown that any two types *have* a least upper bound, which can be used in place of two separate lower bounds for a variable.

3 Definitions

3.1 Notational Conventions

Finite lists are written inside square brackets and are normally denoted by variables with bars, e.g., $\overline{\phi} = [\mathbf{int}, \mathbf{real}, \mathbf{real}\rightarrow\mathbf{real}]$. Since we do not use nested lists, we identify an element ϕ with the singleton list $[\phi]$ and extend the square bracket notation to list extension and concatenation, e.g., $[\phi, \overline{\phi'}]$, $[\overline{\phi}, \phi']$, $[\overline{\phi}, \overline{\phi'}]$, etc. Also, since list concatenation is associative, we normally drop nested brackets, writing both $[[\phi, \phi'], \phi'']$ and $[\phi, [\phi', \phi'']]$ as $[\phi, \phi', \phi'']$.

We will occasionally treat a list as if it were a finite set, writing, for example, $\mathbf{int} \in [\mathbf{bool}, \mathbf{int}]$.

The length of a list $\overline{\phi}$ is written $\mathit{length}(\overline{\phi})$.

We write $A \equiv B$ to mean that A and B are textually identical. Often, the expression B will contain one or more metavariables that are "unbound" in the current context. In this case, $A \equiv B$ is read as "A has the form B," and in the following text the new metavariables are understood to be bound to the corresponding subphrases of A. For example, "$\tau \equiv \mathbf{ns}\rightarrow\sigma$ and $\sigma \leq_Q \mathbf{real}$" is a true statement when τ is $\mathbf{ns}\rightarrow\mathbf{int}$.

$\mathcal{P}_{\mathrm{fin}}(S)$ is the set of finite subsets of a set S.

3.2 Types and Constraint Lists

We begin by defining type expressions (hereafter called simply "types") and the lists of constraints that specify bounds for type variables, as well as some auxiliary notions (like the *size* of a type) that are used in the proofs.

Definition 1: $\mathbf{P}$ is the set of *primitive types*. The metavariable ρ ranges over elements of $\mathbf{P}$. (When we need to talk about more then one primitive type at once, we use ρ with primes and/or subscripts.)

Definition 2: $\mathbf{V}$ is a countable set of *type variables*, ranged over by the metavariable α.

Definition 3: $\mathbf{T}$ is the set of *types*, defined by the following grammar:

$$\tau \ ::= \ \mathbf{ns} \mid \tau\&\tau \mid \tau\rightarrow\tau \mid \rho \mid \alpha$$

The metavariables τ, σ, θ, ϕ, and ψ range over elements of $\mathbf{T}$.

The set of type variables occuring in a type τ is denoted by $\mathit{fv}(\tau)$. If V is a set of type variables, then $\mathbf{T}[V] = \{\tau \in \mathbf{T} \mid \mathit{fv}(\tau) \subseteq V\}$.

If $\overline{\tau}$ is the list $[\tau_1, \tau_2, \ldots, \tau_n]$, then $\overline{\tau}\rightarrow\tau$ abbreviates $\tau_1\rightarrow(\tau_2\rightarrow(\ldots(\tau_n\rightarrow\tau)))$.

Definition 4: A *constraint* q consists of a type variable α, a *lower bound* ϕ, and an *upper bound* ψ, written $\phi \leq \alpha \leq \psi$.

Definition 5: A *constraint list* Q is a list of constraints $[q_1, \ldots, q_n]$ in which:

1. the constrained variables $\alpha_1, \ldots, \alpha_n$ are pairwise distinct;
2. every type variable occurring in ϕ_i or ψ_i is the constrained variable of q_k for some $k < i$.

Each α_i is said to be *defined* in Q; its *rank*, written $rank_Q(\alpha_i)$, is i; its lower and upper bounds, written $lb_Q(\alpha_i)$ and $ub_Q(\alpha_i)$, are ϕ_i and ψ_i.

The *domain* of Q, written $dom(Q)$, is the set of variables defined in Q.

Definition 6: Let $Q = [q_1, \ldots, q_n]$ be a constraint list and $\alpha \in dom(Q)$. Then $Q|_\alpha$ is the prefix $[q_1, \ldots, q_{i-1}]$ of Q, where $i = rank_Q(\alpha)$.

Definition 7: Let Q be a constraint list. The *size* of a type $\tau \in \mathbf{T}[dom(Q)]$, written $size_Q(\tau)$, is defined inductively as follows:

$$\begin{array}{lcl} size_Q(\mathbf{ns}) & = & 0 \\ size_Q(\tau_1 \& \tau_2) & = & size_Q(\tau_1) + size_Q(\tau_2) + 1 \\ size_Q(\tau_1 \rightarrow \tau_2) & = & size_Q(\tau_1) + size_Q(\tau_2) + 1 \\ size_Q(\rho) & = & 0 \\ size_Q(\alpha) & = & \omega \cdot rank_Q(\alpha) \end{array}$$

The size of a list of types, $size_Q([\tau_1, \ldots, \tau_n])$, is $\sum_{1 \le i \le n} size_Q(\tau_i)$.

3.3 The Subtype Relation

We now define the conditions under which one type is a subtype of another.

Definition 8: We assume that a reflexive, transitive relation $\le_{\mathbf{pr}}$ is defined on the elements of $\mathbf{P}$.

Definition 9: Let Q be a constraint list. The *subtype relation* $\le_Q$ is the least relation closed under the rules in Figure 1.

Formally, $\le_Q$ is the least fixed point of a monotone function—the function f that, given a relation $\le_Q{}^i$, adds the immediate consequences of rules A–UB to produce $\le_Q{}^{i+1}$. In fact,

$$\le_Q = \bigcup_{0 \le i \le \infty} (\le_Q{}^i).$$

This means we can think of rules A–UB as a proof system, with rules A, B, D, E, G, H, REFL, LB, and UB as axioms, and C, F, and TRANS as rules of inference. If $\sigma \le_Q \tau$, there is a finite proof tree whose conclusion is $\sigma \le_Q \tau$. In arguments involving $\le_Q$, we often reason by induction on the structure of such proof trees.

Definition 10: The empty constraint list $[\,]$ is *well formed*. If Q is well formed, $\alpha \notin dom(Q)$, and $\phi \le_{Q|_\alpha} \psi$, then $[Q, \phi \le \alpha \le \psi]$ is well formed.

The predicate $wf(Q)$ is true iff Q is well formed.

3.4 The Decision Procedure

Next, we define an algorithm, le_Q, for testing whether a type σ is a subtype of τ. Intuitively, we intend "$le_Q(\sigma, \overline{\tau}, \tau)$" to be equivalent to "$\sigma \le_Q \overline{\tau} \rightarrow \tau$" when Q is well formed.

(A)	$\tau_1 \& \tau_2 \leq_Q \tau_1$
(B)	$\tau_1 \& \tau_2 \leq_Q \tau_2$
(C)	$\dfrac{\sigma \leq_Q \tau_1 \quad \sigma \leq_Q \tau_2}{\sigma \leq_Q \tau_1 \& \tau_2}$
(D)	$\tau \leq_Q \mathbf{ns}$
(E)	$\dfrac{\rho \leq_{\mathrm{pr}} \rho'}{\rho \leq_Q \rho'}$
(F)	$\dfrac{\tau_1 \leq_Q \sigma_1 \quad \sigma_2 \leq_Q \tau_2}{\sigma_1 \to \sigma_2 \leq_Q \tau_1 \to \tau_2}$
(G)	$(\tau_1 \to \tau_2) \& (\tau_1 \to \tau_3) \leq_Q \tau_1 \to (\tau_2 \& \tau_3)$
(H)	$\mathbf{ns} \leq_Q \tau \to \mathbf{ns}$
(REFL)	$\tau \leq_Q \tau$
(TRANS)	$\dfrac{\sigma \leq_Q \theta \quad \theta \leq_Q \tau}{\sigma \leq_Q \tau}$
(LB)	$\dfrac{(\phi \leq \alpha \leq \psi) \in Q}{\phi \leq_Q \alpha}$
(UB)	$\dfrac{(\phi \leq \alpha \leq \psi) \in Q}{\alpha \leq_Q \psi}$

Figure 1: Rules for $\leq_Q$

Definition 11: Let Q be a constraint list. The predicate

$$le_Q \in (\mathbf{T}[dom(Q|_\alpha)] \times \mathbf{List}(\mathbf{T}[dom(Q|_\alpha)]) \times \mathbf{T}[dom(Q|_\alpha)]) \to \mathbf{Bool}$$

is defined inductively by the rules in Figure 2.

It is important that le_Q perform a complete analysis of τ—descending into both sides of &'s, into the right sides of →'s, and into the lower bounds of variables, until it reaches a primitive—before looking at σ at all. This is the reason for le_Q's middle argument: the left sides of →'s are pushed onto a queue as they are encountered and popped off again later, outermost to innermost, as →'s are encountered in σ.

The treatment of variables in rules 9–13b is probably the least transparent aspect of the algorithm. Intuitively, $\sigma \leq_Q \alpha$ can hold either because σ is a subtype of the lower bound of α, or because σ is identically equal to α. So when le_Q encounters a variable α as its third argument, it both descends into the lower bound of α *and* begins to analyze the first argument, leaving the variable itself as the third argument.

To see that le_Q is well defined, note that when the value of $le_Q(\sigma', \overline{\tau}', \tau')$ depends recursively on $le_Q(\sigma, \overline{\tau}, \tau)$, it is always the case that $size_Q(\sigma) + size_Q(\overline{\tau}) + size_Q(\tau)$ is strictly less than $size_Q(\sigma') + size_Q(\overline{\tau}') + size_Q(\tau')$. Because the set of sizes is well-ordered, no infinite sequence of recursive calls is possible.

Also, since le_Q is deterministic, enlarging Q does not affect its results:

Lemma 12: Let Q be a constraint list, $\alpha \in dom(Q)$, $\sigma, \tau \in \mathbf{T}[dom(Q)]$, and $\overline{\tau} \in \mathbf{List}(\mathbf{T}[dom(Q)])$. Then

$$le_{Q|_\alpha}(\sigma, \overline{\tau}, \tau) \;=\; le_Q(\sigma, \overline{\tau}, \tau).$$

4 Correctness of the Decision Procedure

We show the implications

$$\sigma \leq_Q \tau \;\Rightarrow\; le_Q(\sigma, [\,], \tau)$$

and

$$le_Q(\sigma, [\,], \tau) \;\Rightarrow\; \sigma \leq_Q \tau$$

separately. First, we must go to some trouble to show that le_Q considered as a relation (that is, the restriction of le_Q to empty middle arguments) is both reflexive (Corollary 18) and transitive (Corollary 23). With these out of the way, it is easy to show the first implication (Theorem 24). The other implication is much more straightforward (Theorem 28).

4.1 Reflexivity

The proof that the relation computed by le_Q is reflexive, though much shorter than the proof that it is transitive, is not entirely trivial. The difficulty comes from the fact that le_Q is asymmetric: it does a complete analysis of its third argument before ever looking at its first argument. Thus, a straightforward argument using $le_Q(\tau, [\,], \tau)$ as the induction hypothesis breaks down immediately. We need to strengthen the induction hypothesis by introducing the notion of a set, $u_Q(\sigma)$, of "reflexive supertypes" of σ.

(1)	$le_Q(\sigma, \overline{\tau}, \mathbf{ns})$	$=$		true
(2)	$le_Q(\sigma, \overline{\tau}, \tau_1 \& \tau_2)$	$=$		$le_Q(\sigma, \overline{\tau}, \tau_1)$
			$\wedge$	$le_Q(\sigma, \overline{\tau}, \tau_2)$
(3)	$le_Q(\sigma, \overline{\tau}, \tau_1 \rightarrow \tau_2)$	$=$		$le_Q(\sigma, [\overline{\tau}, \tau_1], \tau_2)$
(4)	$le_Q(\mathbf{ns}, \overline{\tau}, \rho)$	$=$		false
(5)	$le_Q(\sigma_1 \& \sigma_2, \overline{\tau}, \rho)$	$=$		$le_Q(\sigma_1, \overline{\tau}, \rho)$
			$\vee$	$le_Q(\sigma_2, \overline{\tau}, \rho)$
(6a)	$le_Q(\sigma_1 \rightarrow \sigma_2, [\tau_a, \overline{\tau}_b], \rho)$	$=$		$le_Q(\tau_a, [\,], \sigma_1)$
			$\wedge$	$le_Q(\sigma_2, \overline{\tau}_b, \rho)$
(6b)	$le_Q(\sigma_1 \rightarrow \sigma_2, [\,], \rho)$	$=$		false
(7a)	$le_Q(\rho, [\tau_a, \overline{\tau}_b], \rho')$	$=$		false
(7b)	$le_Q(\rho, [\,], \rho')$	$=$		$\rho \leq_{\mathrm{pr}} \rho'$
(8)	$le_Q(\alpha, \overline{\tau}, \rho)$	$=$		$le_Q(ub_Q(\alpha), \overline{\tau}, \rho)$
(9)	$le_Q(\mathbf{ns}, \overline{\tau}, \alpha)$	$=$		$le_Q(\mathbf{ns}, \overline{\tau}, lb_Q(\alpha))$
(10)	$le_Q(\sigma_1 \& \sigma_2, \overline{\tau}, \alpha)$	$=$		$le_Q(\sigma_1, \overline{\tau}, \alpha)$
			$\vee$	$le_Q(\sigma_2, \overline{\tau}, \alpha)$
			$\vee$	$le_Q(\sigma_1 \& \sigma_2, \overline{\tau}, lb_Q(\alpha))$
(11a)	$le_Q(\sigma_1 \rightarrow \sigma_2, [\tau_a, \overline{\tau}_b], \alpha)$	$=$		$(le_Q(\tau_a, [\,], \sigma_1) \wedge le_Q(\sigma_2, \overline{\tau}_b, \alpha))$
			$\vee$	$le_Q(\sigma_1 \rightarrow \sigma_2, [\tau_a, \overline{\tau}_b], lb_Q(\alpha))$
(11b)	$le_Q(\sigma_1 \rightarrow \sigma_2, [\,], \alpha)$	$=$		$le_Q(\sigma_1 \rightarrow \sigma_2, [\,], lb_Q(\alpha))$
(12)	$le_Q(\rho, \overline{\tau}, \alpha)$	$=$		$le_Q(\rho, \overline{\tau}, lb_Q(\alpha))$
(13a)	$le_Q(\alpha, [\,], \alpha')$	$=$		$(\alpha \equiv \alpha')$
			$\vee$	$le_Q(ub_Q(\alpha), [\,], \alpha')$
			$\vee$	$le_Q(\alpha, [\,], lb_Q(\alpha'))$
(13b)	$le_Q(\alpha, [\tau_a, \overline{\tau}_b], \alpha')$	$=$		$le_Q(ub_Q(\alpha), [\tau_a, \overline{\tau}_b], \alpha')$
			$\vee$	$le_Q(\alpha, [\tau_a, \overline{\tau}_b], lb_Q(\alpha'))$

Figure 2: Rules for le_Q

Definition 13: Let Q be a constraint list. Define

$$u_Q \in \mathbf{T}[dom(Q)] \rightarrow \mathcal{P}_{\mathrm{fin}}(\mathbf{T}[dom(Q)])$$

as follows:

$$\begin{array}{lcl} u_Q(\mathbf{ns}) & = & \{\mathbf{ns}\} \\ u_Q(\tau_1 \& \tau_2) & = & u_Q(\tau_1) \cup u_Q(\tau_2) \cup \{\tau_1 \& \tau_2\} \\ u_Q(\tau_1 \rightarrow \tau_2) & = & \{\tau_1 \rightarrow \theta_2 \mid \theta_2 \in u_Q(\tau_2)\} \\ u_Q(\rho) & = & \{\rho\} \\ u_Q(\alpha) & = & u_Q(ub_Q(\alpha)) \cup \{\alpha\}. \end{array}$$

We need to establish a few technical lemmas about properties of u_Q.

Lemma 14: $\sigma \in u_Q(\sigma)$.

Proof: By induction on the structure of σ. If $\sigma \equiv \mathbf{ns}$, $\sigma_1 \& \sigma_2$, ρ, or α, the result is immediate from Definition 13. If $\sigma \equiv \sigma_1 \rightarrow \sigma_2$, then by the induction hypothesis, $\sigma_2 \in u_Q(\sigma_2)$, hence $\sigma_1 \rightarrow \sigma_2 \in \{\sigma_1 \rightarrow \theta_2 \mid \theta_2 \in u_Q(\sigma_2)\}$. *(End of Proof)*

Lemma 15: $\sigma \in u_Q(\theta) \Rightarrow u_Q(\sigma) \subseteq u_Q(\theta)$.

Proof: By induction on the structure of σ. *(End of Proof)*

Lemma 16: $\overline{\tau} \rightarrow \tau_i \in u_Q(\overline{\tau} \rightarrow (\tau_1 \& \tau_2))$ for $i = 1, 2$.

Proof: By induction on $length(\overline{\tau})$.

Base step: $\overline{\tau} \equiv [\,]$
Immediate from Definition 13 and Lemma 14.

Induction step: $\overline{\tau} \equiv [\tau_a, \overline{\tau}_b] \quad \overline{\tau}_b \rightarrow \tau_i \in u_Q(\overline{\tau}_b \rightarrow (\tau_1 \& \tau_2))$
By Lemma 15,

$$\begin{array}{ll} & \tau_a \rightarrow (\overline{\tau}_b \rightarrow (\tau_1 \& \tau_2)) \in u_Q(\overline{\tau} \rightarrow (\tau_1 \& \tau_2)) \\ \Rightarrow & u_Q(\tau_a \rightarrow (\overline{\tau}_b \rightarrow (\tau_1 \& \tau_2))) \subseteq u_Q(\overline{\tau} \rightarrow (\tau_1 \& \tau_2)) \\ \Rightarrow & \forall \theta_2 \in u_Q(\overline{\tau}_b \rightarrow (\tau_1 \& \tau_2))).\ \tau_a \rightarrow \theta_2 \in u_Q(\overline{\tau} \rightarrow (\tau_1 \& \tau_2)) \\ \Rightarrow & \tau_a \rightarrow (\overline{\tau}_b \rightarrow \tau_i) \in u_Q(\overline{\tau} \rightarrow (\tau_1 \& \tau_2)). \end{array}$$

(End of Proof)

Now we are ready for the main proof, from which it follows immediately (Corollary 18) that le_Q is reflexive.

Proposition 17: $\overline{\tau} \rightarrow \tau \in u_Q(\sigma) \Rightarrow le_Q(\sigma, \overline{\tau}, \tau)$.

Proof: By complete induction on

$$size_Q(\sigma) + size_Q(\overline{\tau}) + size_Q(\tau).$$

Let

$$\begin{array}{lcl} A & \equiv & \overline{\tau} \rightarrow \tau \in u_Q(\sigma) \\ B & \equiv & le_Q(\sigma, \overline{\tau}, \tau). \end{array}$$

Case i: $\tau \equiv \mathbf{ns}$
By rule 1 of Definition 11, B always holds, and so the implication is immediate.

Case ii: $\tau \equiv \tau_1 \& \tau_2$

$$\begin{array}{lll} A & \Rightarrow & \overline{\tau} \rightarrow \tau_1 \in u_Q(\sigma) \wedge \overline{\tau} \rightarrow \tau_2 \in u_Q(\sigma) \quad \text{(Lemma 16)} \\ & \Rightarrow & le_Q(\sigma, \overline{\tau}, \tau_1) \wedge le_Q(\sigma, \overline{\tau}, \tau_2) \quad \text{(IH)} \\ & \Rightarrow & B. \quad \text{(rule 2)} \end{array}$$

Case iii: $\tau \equiv \tau_1 \rightarrow \tau_2$

$$\begin{array}{lll} A & = & [\overline{\tau}, \tau_1] \rightarrow \tau_2 \in u_Q(\sigma) \\ & \Rightarrow & le_Q(\sigma, [\overline{\tau}, \tau_1], \tau_2) \quad \text{(IH)} \\ & \Rightarrow & B. \quad \text{(rule 3)} \end{array}$$

Case iv: $\tau \equiv \rho \quad \sigma \equiv \mathbf{ns}$

By Definition 13, $\overline{\tau} \rightarrow \rho \notin u_Q(\mathbf{ns})$, so A cannot hold and the implication is trivially true.

Case v: $\tau \equiv \rho \quad \sigma \equiv \sigma_1 \& \sigma_2$

$$\begin{array}{lll} A & \Rightarrow & \overline{\tau} \rightarrow \rho \in u_Q(\sigma_1) \vee \overline{\tau} \rightarrow \rho \in u_Q(\sigma_2) \quad \text{(Definition 13)} \\ & \Rightarrow & le_Q(\sigma_1, \overline{\tau}, \rho) \vee le_Q(\sigma_2, \overline{\tau}, \rho) \quad \text{(IH)} \\ & \Rightarrow & B. \quad \text{(rule 5)} \end{array}$$

Case vi: $\tau \equiv \rho \quad \sigma \equiv \sigma_1 \rightarrow \sigma_2 \quad \overline{\tau} \equiv [\tau_a, \overline{\tau}_b]$

$$\begin{array}{lll} A & \Rightarrow & (\sigma_1 \equiv \tau_a) \wedge \overline{\tau}_b \rightarrow \rho \in u_Q(\sigma_2) \quad \text{(Definition 13)} \\ & \Rightarrow & \sigma_1 \in u_Q(\tau_a) \ \wedge \ \overline{\tau}_b \rightarrow \rho \in u_Q(\sigma_2) \quad \text{(Lemma 14)} \\ & \Rightarrow & le_Q(\tau_a, [\,], \sigma_1) \wedge le_Q(\sigma_2, \overline{\tau}_b, \rho) \quad \text{(IH)} \\ & = & B. \quad \text{(rule 6a)} \end{array}$$

Case vii: $\tau \equiv \rho \quad \sigma \equiv \sigma_1 \rightarrow \sigma_2 \quad \overline{\tau} \equiv [\,]$

By Definition 13, A cannot hold and the implication is trivially true.

Case viii: $\tau \equiv \rho' \quad \sigma \equiv \rho \quad \overline{\tau} \equiv [\,]$

$$\begin{array}{lll} A & \Rightarrow & \rho \equiv \rho' \quad \text{(Definition 13)} \\ & \Rightarrow & \rho \leq_{\mathrm{pr}} \rho' \quad \text{(reflexivity of } \leq_{\mathrm{pr}}\text{)} \\ & \Rightarrow & B. \quad \text{(rule 7b)} \end{array}$$

Case ix: $\tau \equiv \rho' \quad \sigma \equiv \rho \quad \overline{\tau} \not\equiv [\,]$

By Definition 13, A cannot hold.

Case x: $\tau \equiv \rho \quad \sigma \equiv \alpha$

$$\begin{array}{lll} A & \Rightarrow & \overline{\tau} \rightarrow \rho \in u_Q(ub_Q(\alpha)) \quad \text{(Definition 13)} \\ & \Rightarrow & le_Q(ub_Q(\alpha)), \overline{\tau}, \rho) \quad \text{(IH)} \\ & \Rightarrow & B. \quad \text{(rule 8)} \end{array}$$

Case xi: $\tau \equiv \alpha \quad \sigma \equiv \mathbf{ns}$

By Definition 13, A cannot hold.

Case xii: $\tau \equiv \alpha \quad \sigma \equiv \sigma_1 \& \sigma_2$

$$\begin{array}{lll} A & \Rightarrow & \overline{\tau} \rightarrow \alpha \in u_Q(\sigma_1) \vee \overline{\tau} \rightarrow \alpha \in u_Q(\sigma_2) \quad \text{(Definition 13)} \\ & \Rightarrow & le_Q(\sigma_1, \overline{\tau}, \alpha) \vee le_Q(\sigma_2, \overline{\tau}, \alpha) \quad \text{(IH)} \\ & \Rightarrow & B. \quad \text{(rule 10)} \end{array}$$

Case xiii: $\tau \equiv \alpha \quad \sigma \equiv \sigma_1 \rightarrow \sigma_2 \quad \overline{\tau} \equiv [\tau_a, \overline{\tau}_b]$

$$\begin{array}{lll} A & \Rightarrow & (\sigma_1 \equiv \tau_a) \wedge \overline{\tau}_b \rightarrow \alpha \in u_Q(\sigma_2) \quad \text{(Definition 13)} \\ & \Rightarrow & \sigma_1 \in u_Q(\tau_a) \ \wedge \ \overline{\tau}_b \rightarrow \alpha \in u_Q(\sigma_2) \quad \text{(Lemma 14)} \\ & \Rightarrow & le_Q(\tau_a, [\,], \sigma_1) \wedge le_Q(\sigma_2, \overline{\tau}_b, \alpha) \quad \text{(IH)} \\ & \Rightarrow & B. \quad \text{(rule 11a)} \end{array}$$

Case xiv: $\tau \equiv \alpha \quad \sigma \equiv \sigma_1 \rightarrow \sigma_2 \quad \overline{\tau} \equiv [\,]$
By Definition 13, A cannot hold.

Case xv: $\tau \equiv \alpha \quad \sigma \equiv \rho$
By Definition 13, A cannot hold.

Case xvi: $\tau \equiv \alpha' \quad \sigma \equiv \alpha \quad \overline{\tau} \equiv [\,]$

$$\begin{array}{lll} A & \Rightarrow & (\alpha \equiv \alpha') \vee \alpha' \in u_Q(ub_Q(\alpha)) \quad \text{(Definition 13)} \\ & \Rightarrow & le_Q(\alpha, [\,], \alpha') \quad \text{(rule 13a)} \\ & & \vee \ le_Q(ub_Q(\alpha), [\,], \alpha') \quad \text{(IH)} \\ & \Rightarrow & le_Q(\alpha, [\,], \alpha') \vee le_Q(\alpha, [\,], \alpha') \quad \text{(rule 13a)} \\ & = & B. \end{array}$$

Case xvii: $\tau \equiv \alpha' \quad \sigma \equiv \alpha \quad \overline{\tau} \not\equiv [\,]$

$$\begin{array}{lll} A & \Rightarrow & \overline{\tau} \rightarrow \alpha' \in u_Q(ub_Q(\alpha)) \quad \text{(Definition 13)} \\ & \Rightarrow & le_Q(ub_Q(\alpha), \overline{\tau}, \alpha') \quad \text{(IH)} \\ & \Rightarrow & B. \quad \text{(rule 13b)} \end{array}$$

(*End of Proof*)

From Lemma 14 and Proposition 17 it immediately follows that le_Q, considered as a relation, is reflexive.

Corollary 18: $le_Q(\sigma, [\,], \sigma)$.

4.2 Transitivity

In proofs of correctness of decision procedures for transitive relations, the proof of transitivity is the hardest single piece. Here, the main induction hypothesis is quite tricky. Moreover, the fact that the statement of Proposition 22 involves *three* types, the analysis in the proof must consider a very large number of cases.

First, it is convenient to introduce a way of abbreviating an indeterminate number of similar invocations of le_Q.

Definition 19: Let

$$\Psi_Q([\tau_1, \ldots, \tau_n], [\sigma_1, \ldots, \sigma_n]) \ \equiv \ le_Q(\tau_1, [\,], \sigma_1) \wedge \ldots \wedge le_Q(\tau_n, [\,], \sigma_n).$$

Next, for the proof of transitivity to go through, we need to be able to talk about well-formedness (which was defined in terms of $\leq_Q$) in terms of le_Q. If le_Q is correct, then the two alternatives are equivalent. Our proof of correctness therefore proceeds by induction

on Q, proving the transitivity of le_Q (and hence its correctness) from the assumption that $le_{Q'}$ is correct when Q' is a prefix of Q.

Definition 20: Let Q be a constraint list. We say that "property S holds for Q" if for all $\sigma, \tau \in \mathbf{T}[dom(Q)]$,

$$\sigma \leq_Q \tau \quad \Leftrightarrow \quad le_Q(\sigma, [\,], \tau).$$

One lemma takes care of a common case analysis:

Lemma 21: $le_Q(\sigma, \overline{\tau}, lb_Q(\alpha)) \Rightarrow le_Q(\sigma, \overline{\tau}, \alpha)$.

Proof: Immediate from rules 9–13. (*End of Proof*)

Now we are ready for the main theorem, from which the transitivity of le_Q follows directly (Corollary 23).

Proposition 22: Let Q be a well-formed constraint list. Assume that for each $\alpha \in \mathbf{T}[dom(Q)]$, property S holds of $Q|_\alpha$. Then

$$le_Q(\tau, \overline{\tau}', \tau') \wedge \Psi_Q(\overline{\tau}''', \overline{\tau}') \wedge le_Q(\tau', \overline{\tau}'', \tau'')$$

implies

$$le_Q(\tau, [\overline{\tau}''', \overline{\tau}''], \tau'').$$

Proof: By complete induction on

$$size_Q(\tau) + size_Q(\overline{\tau}') + size_Q(\tau') + size_Q(\overline{\tau}'') + size_Q(\tau'').$$

Let

$$\begin{aligned} A &\equiv le_Q(\tau, \overline{\tau}', \tau') \\ B &\equiv \Psi_Q(\overline{\tau}''', \overline{\tau}') \\ C &\equiv le_Q(\tau', \overline{\tau}'', \tau'') \\ D &\equiv le_Q(\tau, [\overline{\tau}''', \overline{\tau}''], \tau''). \end{aligned}$$

Since $\overline{\tau}'$ and $\overline{\tau}'''$ always have the same length, we assume throughout the proof that whenever $\overline{\tau}' \equiv [\,]$, $\overline{\tau}''' \equiv [\,]$, and whenever $\overline{\tau}' \equiv [\tau'_a, \overline{\tau}'_b]$, $\overline{\tau}''' \equiv [\tau'''_a, \overline{\tau}'''_b]$.

Case i: $\tau'' \equiv \mathbf{ns}$

By rule 1 of Definition 11, D is always true, so the implication $A \wedge B \wedge C \Rightarrow D$ is immediate.

Case ii: $\tau'' \equiv \tau''_1 \& \tau''_2$

$$\begin{array}{rcll} A \wedge B \wedge C & = & (A \wedge B \wedge le_Q(\tau', \overline{\tau}'', \tau''_1)) & \\ & \wedge & (A \wedge B \wedge le_Q(\tau', \overline{\tau}'', \tau''_2)) & \text{(rule 2)} \\ & \Rightarrow & le_Q(\tau, [\overline{\tau}''', \overline{\tau}''], \tau''_1) & \\ & \wedge & le_Q(\tau, [\overline{\tau}''', \overline{\tau}''], \tau''_2) & \text{(IH (twice))} \\ & = & D. & \text{(rule 2)} \end{array}$$

Case iii: $\tau'' \equiv \tau_1'' \to \tau_2''$

$$\begin{array}{rcll} A \wedge B \wedge C & = & A \wedge B \wedge le_Q(\tau', [\overline{\tau}'', \tau_1''], \tau_2'') & \text{(rule 3)} \\ & \Rightarrow & le_Q(\tau, [\overline{\tau}''', \overline{\tau}'', \tau_1''], \tau_2'') & \text{(IH)} \\ & = & D. & \text{(rule 3)} \end{array}$$

Case iv: $\tau'' \equiv \rho''$ $\tau' \equiv \mathbf{ns}$

By rule 4, C cannot hold, so the implication is trivially true.

Case v: $\tau'' \equiv \rho''$ $\tau' \equiv \tau_1' \,\&\, \tau_2'$

$$\begin{array}{rcll} A \wedge B \wedge C & = & le_Q(\tau, \overline{\tau}', \tau_1') \wedge le_Q(\tau, \overline{\tau}', \tau_2') & \text{(rule 2)} \\ & & \wedge \; B & \\ & & \wedge \; (le_Q(\tau_1', \overline{\tau}'', \rho'') \vee le_Q(\tau_2', \overline{\tau}'', \rho'')) & \text{(rule 5)} \\ & \Rightarrow & (le_Q(\tau, \overline{\tau}', \tau_1') \wedge B \wedge le_Q(\tau_1', \overline{\tau}'', \tau'')) & \\ & & \vee \; (le_Q(\tau, \overline{\tau}', \tau_2') \wedge B \wedge le_Q(\tau_2', \overline{\tau}'', \tau'')) & \\ & \Rightarrow & D \vee D. & \text{(IH)} \end{array}$$

Case vi: $\tau'' \equiv \rho''$ $\tau' \equiv \tau_1' \to \tau_2'$ $\overline{\tau}'' \equiv [\tau_a'', \overline{\tau}_b'']$

$$\begin{array}{rcll} A \wedge B \wedge C & = & le_Q(\tau, [\overline{\tau}', \tau_1'], \tau_2') & \text{(rule 3)} \\ & & \wedge \; B & \\ & & \wedge \; le_Q(\tau_a'', [\,], \tau_1') \wedge le_Q(\tau_2', \overline{\tau}_b'', \rho'') & \text{(rule 6a)} \\ & = & le_Q(\tau, [\overline{\tau}', \tau_1'], \tau_2') & \\ & & \wedge \; \Psi_Q([\overline{\tau}''', \tau_a''], [\overline{\tau}', \tau_1']) \wedge le_Q(\tau_2', \overline{\tau}_b'', \rho'') & \\ & \Rightarrow & le_Q(\tau, [\overline{\tau}''', \tau_a'', \tau_b''], \rho'') & \text{(IH)} \\ & = & D. & \end{array}$$

Case vii: $\tau'' \equiv \rho''$ $\tau' \equiv \tau_1' \to \tau_2'$ $\overline{\tau}'' \equiv [\,]$

By rule 6b, C is always false, so the implication holds trivially.

Case viii: $\tau'' \equiv \rho''$ $\tau' \equiv \rho'$ $\tau \equiv \mathbf{ns}$

By rule 4, A never holds, so the implication is trivially true.

Case ix: $\tau'' \equiv \rho''$ $\tau' \equiv \rho'$ $\tau \equiv \tau_1 \& \tau_2$

$$\begin{array}{rcll} A \wedge B \wedge C & = & (le_Q(\tau_1, \overline{\tau}', \rho') \wedge B \wedge C) & \\ & & \vee \; (le_Q(\tau_2, \overline{\tau}', \rho') \wedge B \wedge C) & \text{(rule 5)} \\ & \Rightarrow & le_Q(\tau_1, [\overline{\tau}''', \overline{\tau}''], \rho'') & \\ & & \vee \; le_Q(\tau_2, [\overline{\tau}''', \overline{\tau}''], \rho'') & \text{(IH)} \\ & = & D. & \text{(rule 5)} \end{array}$$

Case x: $\tau'' \equiv \rho''$ $\tau' \equiv \rho'$ $\tau \equiv \tau_1 \to \tau_2$ $\overline{\tau}' \equiv [\tau_a', \overline{\tau}_b']$

$$\begin{array}{rcll} A \wedge B \wedge C & = & le_Q(\tau_a', [\,], \tau_1) \wedge le_Q(\tau_2, \overline{\tau}_b', \rho') & \\ & & \wedge \; le_Q(\tau_a''', [\,], \tau_a') \wedge \Psi_Q(\overline{\tau}_b''', \overline{\tau}_b') & \\ & & \wedge \; C & \text{(rule 6a)} \\ & = & le_Q(\tau_a', [\,], \tau_1) \wedge le_Q(\tau_a''', [\,], \tau_a') & \\ & & \wedge \; le_Q(\tau_2, \overline{\tau}_b', \rho') \wedge \Psi_Q(\overline{\tau}_b''', \overline{\tau}_b') \wedge C & \\ & \Rightarrow & le_Q(\tau_a''', [\,], \tau_1) & \\ & & \wedge \; le_Q(\tau_2, [\overline{\tau}_b''', \overline{\tau}''], \rho'') & \text{(IH)} \\ & = & D. & \text{(rule 6a)} \end{array}$$

Case xi: $\tau'' \equiv \rho''$ $\tau' \equiv \rho'$ $\tau \equiv \tau_1 \rightarrow \tau_2$ $\overline{\tau}' \equiv [\,]$

By rule 6b, A never holds, so the implication is trivially true.

Case xii: $\tau'' \equiv \rho''$ $\tau' \equiv \rho'$ $\tau \equiv \rho$ $\overline{\tau}' \equiv [\,]$ $\overline{\tau}'' \equiv [\,]$

$$\begin{array}{rcll} A \wedge B \wedge C & = & (\rho \leq_{\mathrm{pr}} \rho') \wedge (\rho' \leq_{\mathrm{pr}} \rho'') & \text{(rule 7a)} \\ & \Rightarrow & \rho \leq_{\mathrm{pr}} \rho'' & \text{(transitivity of } \leq_{\mathrm{pr}}\text{)} \\ & = & D. & \text{(rule 7a)} \end{array}$$

Case xiii: $\tau'' \equiv \rho''$ $\tau' \equiv \rho'$ $\tau \equiv \rho$ $\overline{\tau}' \not\equiv [\,] \vee \overline{\tau}'' \not\equiv [\,]$

By rule 7b, either A or C is false, so the implication holds trivially.

Case xiv: $\tau'' \equiv \rho''$ $\tau' \equiv \rho'$ $\tau \equiv \alpha$

$$\begin{array}{rcll} A \wedge B \wedge C & = & le_Q(ub_Q(\alpha), \overline{\tau}', \rho') \wedge B \wedge C & \text{(rule 8)} \\ & \Rightarrow & le_Q(ub_Q(\alpha), [\overline{\tau}''', \overline{\tau}''], \rho'') & \text{(IH)} \\ & = & D. & \text{(rule 8)} \end{array}$$

Case xv: $\tau'' \equiv \rho''$ $\tau' \equiv \alpha'$ $\tau \equiv \mathbf{ns}$

$$\begin{array}{rcll} A \wedge B \wedge C & = & le_Q(\mathbf{ns}, \overline{\tau}', lb_Q(\alpha')) & \text{(rule 9)} \\ & \wedge & B & \\ & \wedge & le_Q(ub_Q(\alpha'), \overline{\tau}'', \rho''). & \text{(rule 8)} \end{array}$$

From the well-formedness of Q, we have

$$lb_Q(\alpha') \leq_{Q|_{\alpha'}} ub_Q(\alpha').$$

Property **S** gives us

$$le_{Q|_{\alpha'}}(lb_Q(\alpha'), [\,], ub_Q(\alpha')).$$

Lemma 12 then gives

$$le_Q(lb_Q(\alpha'), [\,], ub_Q(\alpha')).$$

Now, since $size_Q(\mathbf{ns}) + size_Q(\alpha') > size_Q(lb_Q(\alpha')) + size_Q(ub_Q(\alpha'))$, the IH applies, giving

$$le_Q(\mathbf{ns}, \overline{\tau}', lb_Q(\alpha')) \wedge B \wedge le_Q(lb_Q(\alpha'), \overline{\tau}'', \rho'').$$

Since $size_Q(\alpha') > size_Q(lb_Q(\alpha'))$, the IH applies again, giving D.

This pattern of reasoning arises in most of the proof cases involving variables, and since the applicability of the induction hypothesis does not depend on the form of τ, we need not repeat the argument when we come to it again. Let

$$E \equiv le_Q(\tau, \overline{\tau}', lb_Q(\alpha')) \wedge B \wedge le_Q(ub_Q(\alpha'), \overline{\tau}'', \rho'').$$

To indicate that $E \Rightarrow D$ by the well-formedness of Q and two applications of the induction hypothesis, we write simply

$$E \Rightarrow D. \qquad (wf(Q) + \mathrm{IH} \times 2)$$

It will also be convenient to define temporary abbreviations for expressions that appear several times. Our convention is that these temporary abbreviations are assigned letters in the middle of the alphabet, while abbreviations whose scope is the entire proof are assigned letters from the beginning of the alphabet.

Case xvi: $\tau'' \equiv \rho''$ $\tau' \equiv \alpha'$ $\tau \equiv \tau_1 \& \tau_2$

$$A \wedge B \wedge C = M_1 \vee M_2 \vee E, \quad \text{(rules 10 and 8)}$$

where

$$M_1 \equiv le_Q(\tau_1, \overline{\tau}', \alpha') \wedge B \wedge C$$
$$M_2 \equiv le_Q(\tau_2, \overline{\tau}', \alpha') \wedge B \wedge C.$$

Now,

$$\begin{array}{lll} M_1 \vee M_2 \Rightarrow & le_Q(\tau_1, [\overline{\tau}''', \overline{\tau}''], \rho'') & \\ \quad \vee & le_Q(\tau_2, [\overline{\tau}''', \overline{\tau}''], \rho'') & \text{(IH)} \\ \Rightarrow & D. & \text{(rule 5)} \\ E \Rightarrow & D. & (wf(Q) + \text{IH} \times 2) \end{array}$$

Case xvii: $\tau'' \equiv \rho''$ $\tau' \equiv \alpha'$ $\tau \equiv \tau_1 \rightarrow \tau_2$ $\overline{\tau}' \equiv [\,]$

$$\begin{array}{lll} A \wedge B \wedge C = & E & \text{(rules 11b and 8)} \\ \Rightarrow & D. & (wf(Q) + \text{IH} \times 2) \end{array}$$

Case xviii: $\tau'' \equiv \rho''$ $\tau' \equiv \alpha'$ $\tau \equiv \tau_1 \rightarrow \tau_2$ $\overline{\tau}' \equiv [\tau'_a, \overline{\tau}'_b]$

$$A \wedge B \wedge C = M \vee E, \quad \text{(rules 11a and 8)}$$

where

$$\begin{array}{lll} M \equiv & le_Q(\tau'_a, [\,], \tau_1) \wedge le_Q(\tau_2, \overline{\tau}'_b, \alpha') & \\ \quad \wedge & le_Q(\tau'''_a, [\,], \tau'_a) \wedge \Psi_Q(\overline{\tau}'''_b, \overline{\tau}'_b) & \\ \quad \wedge & C. & \end{array}$$

Now,

$$\begin{array}{lll} M \Rightarrow & le_Q(\tau'''_a, [\,], \tau_1) & \\ \quad \wedge & le_Q(\tau_2, [\overline{\tau}'''_b, \overline{\tau}''], \rho'') & \text{(IH)} \\ = & D. & \text{(rule 6)} \\ E \Rightarrow & D. & (wf(Q) + \text{IH} \times 2) \end{array}$$

Case xix: $\tau'' \equiv \rho''$ $\tau' \equiv \alpha'$ $\tau \equiv \rho$

$$\begin{array}{lll} A \wedge B \wedge C = & E & \text{(rules 12 and 8)} \\ \Rightarrow & D. & (wf(Q) + \text{IH} \times 2) \end{array}$$

Case xx: $\tau'' \equiv \rho''$ $\tau' \equiv \alpha'$ $\tau \equiv \alpha$ $\overline{\tau}' \equiv [\,]$

$$A \wedge B \wedge C = L \vee M \vee E, \quad \text{(rules 13a and 8)}$$

where

$$L \equiv (\alpha \equiv \alpha') \wedge B \wedge le_Q(ub_Q(\alpha'), \overline{\tau}'', \rho'')$$
$$M \equiv le_Q(ub_Q(\alpha), \overline{\tau}', \alpha') \wedge B \wedge C.$$

Now,

$$\begin{array}{lll} L \Rightarrow & D. & \text{(rule 8)} \\ M \Rightarrow & le_Q(ub_Q(\alpha), [\overline{\tau}''', \overline{\tau}''], \rho'') & \text{(IH)} \\ = & D. & \text{(rule 8)} \\ E \Rightarrow & D. & (wf(Q) + \text{IH} \times 2) \end{array}$$

Case xxi: $\tau'' \equiv \rho''$ $\tau' \equiv \alpha'$ $\tau \equiv \alpha$ $\overline{\tau}' \not\equiv [\,]$

$$\begin{array}{lcll} A \wedge B \wedge C & = & M \vee E, & \text{(rules 13b and 8)} \\ & \Rightarrow & D. & \text{(as in case xx)} \end{array}$$

Case xxii: $\tau'' \equiv \alpha''$ $\tau' \equiv \mathbf{ns}$

$$\begin{array}{lcll} A \wedge B \wedge C & = & A \wedge B \wedge le_Q(\mathbf{ns}, \overline{\tau}'', lb_Q(\alpha'')) & \text{(rule 9)} \\ & \Rightarrow & D. & \text{(IH and rule 9)} \end{array}$$

Case xxiii: $\tau'' \equiv \alpha''$ $\tau' \equiv \tau_1' \& \tau_2'$

$$A \wedge B \wedge C = L \vee M \vee N, \quad \text{(rule 10)}$$

where

$$\begin{array}{lcl} L & \equiv & A \wedge B \wedge le_Q(\tau_1', \overline{\tau}'', \alpha'') \\ M & \equiv & A \wedge B \wedge le_Q(\tau_2', \overline{\tau}'', \alpha'') \\ N & \equiv & A \wedge B \wedge le_Q(\tau_1' \& \tau_2', \overline{\tau}'', lb_Q(\alpha'')). \end{array}$$

Now,

$$\begin{array}{lcll} L & \Rightarrow & le_Q(\tau, \overline{\tau}', \tau_1') \wedge B \wedge le_Q(\tau_1', \overline{\tau}'', \alpha'') & \text{(rule 2)} \\ & \Rightarrow & D. & \text{(IH)} \\ M & \Rightarrow & D. & \text{(similarly)} \\ N & \Rightarrow & le_Q(\tau, [\overline{\tau}''', \overline{\tau}''], lb_Q(\alpha'')) & \text{(IH)} \\ & \Rightarrow & D. & \text{(Lemma 21)} \end{array}$$

Again, the pattern of reasoning by which $N \Rightarrow D$ appears several times in the remainder of the proof. So we let

$$F \equiv A \wedge B \wedge le_Q(\tau', \overline{\tau}'', lb_Q(\alpha''))$$

and write simply

$$F \Rightarrow D. \quad \text{(IH + Lemma 21)}$$

Case xxiv: $\tau'' \equiv \alpha''$ $\tau' \equiv \tau_1' \rightarrow \tau_2'$ $\overline{\tau}'' \equiv [\tau_a'', \overline{\tau}_b'']$

$$A \wedge B \wedge C = L \vee F, \quad \text{(rules 11a and 3)}$$

where

$$L \equiv le_Q(\tau, [\overline{\tau}', \tau_1'], \tau_2') \wedge B \wedge le_Q(\tau_a'', [\,], \tau_1') \wedge le_Q(\tau_2', \overline{\tau}_b'', \alpha'').$$

Now,

$$\begin{array}{lcll} L & \Rightarrow & le_Q(\tau, [\overline{\tau}''', \overline{\tau}''], \alpha'') & \text{(IH)} \\ & = & D. & \\ F & \Rightarrow & D. & \text{(IH + Lemma 21)} \end{array}$$

Case xxv: $\tau'' \equiv \alpha''$ $\tau' \equiv \tau_1' \rightarrow \tau_2'$ $\overline{\tau}'' \equiv [\,]$

$$\begin{array}{lcll} A \wedge B \wedge C & = & F & \text{(rule 11b)} \\ & \Rightarrow & D. & \text{(IH + Lemma 21)} \end{array}$$

Case xxvi: $\tau'' \equiv \alpha''$ $\tau' \equiv \rho'$

$$\begin{array}{rcll} A \wedge B \wedge C & = & F & \text{(rule 12)} \\ & \Rightarrow & D. & \text{(IH + Lemma 21)} \end{array}$$

Case xxvii: $\tau'' \equiv \alpha''$ $\tau' \equiv \alpha'$ $\tau \equiv \mathbf{ns}$ $\overline{\tau}'' \equiv [\,]$

$$A \wedge B \wedge C = L \vee E \vee F, \quad \text{(rules 13a and 9)}$$

where

$$L \equiv A \wedge B \wedge (\alpha' \equiv \alpha'').$$

Now,

$$\begin{array}{rcll} L & \Rightarrow & le_Q(\mathbf{ns}, \overline{\tau}', \alpha'') \wedge B \wedge le_Q(\alpha'', [\,], \alpha'') & \text{(Corollary 18)} \\ & \Rightarrow & le_Q(\mathbf{ns}, \overline{\tau}''', \alpha'') & \text{(IH)} \\ & = & D. & \\ E & \Rightarrow & D. & (wf(Q) + \text{IH} \times 2) \\ F & \Rightarrow & D. & \text{(IH + Lemma 21)} \end{array}$$

Case xxviii: $\tau'' \equiv \alpha''$ $\tau' \equiv \alpha'$ $\tau \equiv \mathbf{ns}$ $\overline{\tau}'' \not\equiv [\,]$

$$\begin{array}{rcll} A \wedge B \wedge C & = & E \vee F & \text{(rules 13b and 9)} \\ & \Rightarrow & D. & \end{array}$$

Case xxix: $\tau'' \equiv \alpha''$ $\tau' \equiv \alpha'$ $\tau \equiv \tau_1 \& \tau_2$ $\overline{\tau}'' \equiv [\,]$

$$A \wedge B \wedge C = L \vee M_1 \vee M_2 \vee E \vee F, \quad \text{(rules 13a and 10)}$$

where L is the same as in case *xxvii*, and

$$\begin{array}{rcl} M_1 & \equiv & le_Q(\tau_1, \overline{\tau}', \alpha') \wedge B \wedge C \\ M_2 & \equiv & le_Q(\tau_2, \overline{\tau}', \alpha') \wedge B \wedge C \end{array}$$

Now,

$$\begin{array}{rcll} L & \Rightarrow & D. & \text{(as in case } xxvii\text{)} \\ M_1 & \Rightarrow & le_Q(\tau_1, [\overline{\tau}''', \overline{\tau}''], \alpha'') & \text{(IH)} \\ & \Rightarrow & D. & \text{(rule 10)} \\ M_2 & \Rightarrow & D. & \text{(similarly)} \\ E & \Rightarrow & D. & (wf(Q) + \text{IH} \times 2) \\ F & \Rightarrow & D. & \text{(IH + Lemma 21)} \end{array}$$

Case xxx: $\tau'' \equiv \alpha''$ $\tau' \equiv \alpha'$ $\tau \equiv \tau_1 \& \tau_2$ $\overline{\tau}'' \not\equiv [\,]$

$$\begin{array}{rcll} A \wedge B \wedge C & = & M_1 \vee M_2 \vee E \vee F & \text{(rules 13b and 10)} \\ & \Rightarrow & D. & \text{(as in case } xxix\text{)} \end{array}$$

where M_1 and M_2 are the same as in case *xxix*.

Case xxxi: $\tau'' \equiv \alpha''$ $\tau' \equiv \alpha'$ $\tau \equiv \tau_1 \rightarrow \tau_2$ $\overline{\tau}'' \equiv [\,]$ $\overline{\tau}' \equiv [\tau'_a, \overline{\tau}'_b]$

$$A \wedge B \wedge C \Rightarrow L \vee M \vee E \vee F, \qquad \text{(rules 13a and 11a)}$$

where L is the same as in case *xxvii*, and

$$M \equiv le_Q(\tau'_a, [\,], \tau_1) \wedge le_Q(\tau_2, \overline{\tau}'_b, \alpha') \wedge B \wedge C$$

Now,

$$\begin{array}{lll} L \Rightarrow & D. & \text{(as in case } xxvii\text{)} \\ M \Rightarrow & le_Q(\tau'''_a, [\,], \tau_1) & \\ \quad \wedge & le_Q(\tau_2, [\overline{\tau}'''_b, \overline{\tau}''], \alpha'') & \text{(IH)} \\ \Rightarrow & D. & \text{(rule 11)} \\ E \Rightarrow & D. & (wf(Q) + \text{IH} \times 2) \\ F \Rightarrow & D. & \text{(IH + Lemma 21)} \end{array}$$

Case xxxii: $\tau'' \equiv \alpha''$ $\tau' \equiv \alpha'$ $\tau \equiv \tau_1 \rightarrow \tau_2$ $\overline{\tau}'' \not\equiv [\,]$ $\overline{\tau}' \equiv [\tau'_a, \overline{\tau}'_b]$

$$\begin{array}{lll} A \wedge B \wedge C = & M \vee E \vee F & \text{(rules 13b and 11a)} \\ \Rightarrow & D, & \text{(as in case } xxxi\text{)} \end{array}$$

where M is the same as in case *xxxi*.

Case xxxiii: $\tau'' \equiv \alpha''$ $\tau' \equiv \alpha'$ $\tau \equiv \tau_1 \rightarrow \tau_2$ $\overline{\tau}'' \equiv [\,]$ $\overline{\tau}' \equiv [\,]$

$$\begin{array}{lll} A \wedge B \wedge C = & L \vee E \vee F & \text{(rules 13a and 11b)} \\ \Rightarrow & D, & \text{(as in case } xxxi\text{)} \end{array}$$

where L is the same as in case *xxxi*.

Case xxxiv: $\tau'' \equiv \alpha''$ $\tau' \equiv \alpha'$ $\tau \equiv \tau_1 \rightarrow \tau_2$ $\overline{\tau}'' \not\equiv [\,]$ $\overline{\tau}' \equiv [\,]$

$$\begin{array}{lll} A \wedge B \wedge C = & E \vee F & \text{(rules 13b and 11b)} \\ \Rightarrow & D. & \end{array}$$

Case xxxv: $\tau'' \equiv \alpha''$ $\tau' \equiv \alpha'$ $\tau \equiv \rho$ $\overline{\tau}'' \equiv [\,]$

$$\begin{array}{lll} A \wedge B \wedge C = & L \vee E \vee F & \text{(rules 13a and 12)} \\ \Rightarrow & D. & \text{(as in case } xxvii\text{)} \end{array}$$

where L is the same as in case *xxvii*.

Case xxxvi: $\tau'' \equiv \alpha''$ $\tau' \equiv \alpha'$ $\tau \equiv \rho$ $\overline{\tau}'' \not\equiv [\,]$

$$\begin{array}{lll} A \wedge B \wedge C = & E \vee F & \text{(rules 13b and 12)} \\ \Rightarrow & D. & \end{array}$$

Case xxxvii: $\tau'' \equiv \alpha''$ $\tau' \equiv \alpha'$ $\tau \equiv \alpha$ $\overline{\tau}'' \equiv [\,]$ $\overline{\tau}' \equiv [\,]$

$$A \wedge B \wedge C \Rightarrow L \vee M \vee N \vee E \vee F, \qquad \text{(rule 13b)}$$

where L is the same as in case *xxvii*, and

$$M \equiv (\alpha \equiv \alpha') \wedge B \wedge le_Q(ub_Q(\alpha'), \overline{\tau}'', \alpha'')$$
$$N \equiv le_Q(ub_Q(\alpha), \overline{\tau}', \alpha') \wedge B \wedge C.$$

Now,

L	$\Rightarrow$	$D.$	(as in case *xxvii*)
M	$=$	$le_Q(ub_Q(\alpha), \overline{\tau}'', \alpha'')$	(IH)
	$\Rightarrow$	$D.$	(rule 13)
N	$\Rightarrow$	$le_Q(ub_Q(\alpha), [\overline{\tau}''', \overline{\tau}''], \alpha'')$	(IH)
	$\Rightarrow$	$D.$	(rule 13)
E	$\Rightarrow$	$D.$	($wf(Q)$ + IH $\times$ 2)
F	$\Rightarrow$	$D.$	(IH + Lemma 21)

Case xxxviii: $\tau'' \equiv \alpha''$ $\tau' \equiv \alpha'$ $\tau \equiv \alpha$ $\overline{\tau}'' \not\equiv [\,]$ $\overline{\tau}' \equiv [\,]$

$A \wedge B \wedge C$	$=$	$M \vee N \vee E \vee F,$	(rules 13b and 13a)
	$\Rightarrow$	$D,$	(as in case *xxxvii*)

where M and N are the same as in case *xxxvii*.

Case xxxix: $\tau'' \equiv \alpha''$ $\tau' \equiv \alpha'$ $\tau \equiv \alpha$ $\overline{\tau}'' \equiv [\,]$ $\overline{\tau}' \not\equiv [\,]$

$A \wedge B \wedge C$	$=$	$L \vee N \vee E \vee F,$	(rules 13a and 13b)
	$\Rightarrow$	$D,$	(as in case *xxxvii*)

where L and N are the same as in case *xxxvii*.

Case xl: $\tau'' \equiv \alpha''$ $\tau' \equiv \alpha'$ $\tau \equiv \alpha$ $\overline{\tau}'' \not\equiv [\,]$ $\overline{\tau}' \not\equiv [\,]$

$A \wedge B \wedge C$	$=$	$N \vee E \vee F,$	(rule 13a)
	$\Rightarrow$	$D,$	(as in case *xxxvii*)

where N is the same as in case *xxxvii*. (*End of Proof*)

From Proposition 22, it now follows directly that the relation computed by le_Q is transitive:

Corollary 23: $le_Q(\tau, [\,], \tau') \wedge le_Q(\tau', [\,], \tau'') \Rightarrow le_Q(\tau, [\,], \tau'')$.

4.3 Main Proof of Correctness

With transitivity and reflexivity now in hand, it is a simple matter to complete the proof that the relation computed by le_Q is contained in the relation $\leq_Q$ whenever Q is well formed.

Theorem 24: Let Q be a well-formed constraint list. Assume that for each $\alpha \in \mathbf{T}[dom(Q)]$, property **S** holds of $Q|_\alpha$. Then

$$\sigma \leq_Q \tau \Rightarrow le_Q(\sigma, [\,], \tau).$$

Proof: By the comments in Section 3.3, it suffices to show that if there exists a proof of $\sigma \leq_Q \tau$ from rules A–UB, then $le_Q(\sigma, [\,], \tau) =$ true. We may therefore argue by induction on the structure of this proof. We proceed by cases on the last rule used.

Case A: $\tau_1 \& \tau_2 \leq_Q \tau_1$

$$\begin{array}{lll} & \tau_1 \in u_Q(\tau_1) & \text{(Lemma 14)} \\ \Rightarrow & \tau_1 \in u_Q(\tau_1 \& \tau_2) & \text{(Definition 13)} \\ \Rightarrow & le_Q(\tau_1 \& \tau_2, [\,], \tau_1). & \text{(Proposition 17)} \end{array}$$

Case B: $\tau_1 \& \tau_2 \leq_Q \tau_2$
Similar.

Case C: $\dfrac{\sigma \leq_Q \tau_1 \quad \sigma \leq_Q \tau_2}{\sigma \leq_Q \tau_1 \& \tau_2}$

$$\begin{array}{lll} & le_Q(\sigma, [\,], \tau_1) \wedge le_Q(\sigma, [\,], \tau_2) & \text{(IH)} \\ \Rightarrow & le_Q(\sigma, [\,], \tau_1 \& \tau_2). & \text{(rule 2)} \end{array}$$

Case D: $\tau \leq_Q \mathbf{ns}$
By rule 1, $le_Q(\tau, [\,], \mathbf{ns})$ holds for all τ.

Case E: $\dfrac{\rho \leq_{\mathrm{pr}} \rho'}{\rho \leq_Q \rho'}$
By rule 7b, $\rho \leq_{\mathrm{pr}} \rho' \;\Rightarrow\; le_Q(\rho, [\,], \rho')$.

Case F: $\dfrac{\tau_1 \leq_Q \sigma_1 \quad \sigma_2 \leq_Q \tau_2}{\sigma_1 \rightarrow \sigma_2 \leq_Q \tau_1 \rightarrow \tau_2}$

$$\begin{array}{lll} & le_Q(\tau_1, [\,], \sigma_1) \wedge le_Q(\sigma_2, [\,], \tau_2) & \text{(IH)} \\ \Rightarrow & le_Q(\sigma_1 \rightarrow \sigma_2, [\,], \tau_1 \rightarrow \tau_2). & \text{(rules 6a and 3)} \end{array}$$

Case G: $(\tau_1 \rightarrow \tau_2) \& (\tau_1 \rightarrow \tau_3) \leq_Q \tau_1 \rightarrow (\tau_2 \& \tau_3)$
Let $T \equiv (\tau_1 \rightarrow \tau_2) \& (\tau_1 \rightarrow \tau_3)$. Then

$$\begin{array}{lll} & \tau_1 \rightarrow \tau_2 \in u_Q(\tau_1 \rightarrow \tau_2) \wedge \tau_1 \rightarrow \tau_3 \in u_Q(\tau_1 \rightarrow \tau_3) & \text{(Lemma 14)} \\ \Rightarrow & \tau_1 \rightarrow \tau_2 \in u_Q(T) \wedge \tau_1 \rightarrow \tau_3 \in u_Q(T) & \text{(Definition 13)} \\ \Rightarrow & le_Q(T, [\tau_1], \tau_2) \wedge le_Q(T, [\tau_1], \tau_3) & \text{(Proposition 17)} \\ \Rightarrow & le_Q(T, [\tau_1], \tau_2 \& \tau_3) & \text{(rule 2)} \\ \Rightarrow & le_Q(T, [\,], \tau_1 \rightarrow (\tau_2 \& \tau_3)). & \text{(rule 3)} \end{array}$$

Case H: $\mathbf{ns} \leq_Q \tau \rightarrow \mathbf{ns}$
By rule 1, $le_Q(\mathbf{ns}, [\tau], \mathbf{ns})$. By rule 3, $le_Q(\mathbf{ns}, [\tau], \tau \rightarrow \mathbf{ns})$.

Case REFL: $\tau \leq_Q \tau$
By Corollary 18, $le_Q(\tau, [\,], \tau)$.

Case TRANS: $\dfrac{\sigma \leq_Q \theta \quad \theta \leq_Q \tau}{\sigma \leq_Q \tau}$

$$\begin{array}{lll} & le_Q(\sigma, [\,], \theta) \wedge le_Q(\theta, [\,], \tau) & \text{(IH)} \\ \Rightarrow & le_Q(\sigma, [\,], \tau). & \text{(Corollary 23)} \end{array}$$

Case LB: $\dfrac{(\phi \leq \alpha \leq \psi) \in Q}{\phi \leq_Q \alpha}$

	$le_Q(lb_Q(\alpha), [\,], lb_Q(\alpha))$	(Corollary 18)
$\Rightarrow$	$le_Q(lb_Q(\alpha), [\,], \alpha)$.	(Lemma 21)

Case UB: $\dfrac{(\phi \leq \alpha \leq \psi) \in Q}{\alpha \leq_Q \psi}$

	$ub_Q(\alpha) \in u_Q(ub_Q(\alpha))$	(Lemma 14)
$\Rightarrow$	$ub_Q(\alpha) \in u_Q(\alpha)$	(Definition 13)
$\Rightarrow$	$le_Q(\alpha, [\,], ub_Q(\alpha))$.	(Proposition 17)

(*End of Proof*)

Our last task is to prove that the relation computed by le_Q contains $\leq_Q$. It is convenient to begin by establishing some simple properties of $\leq_Q$.

Lemma 25: $\mathbf{ns} \leq_Q \overline{\tau} \rightarrow \mathbf{ns}$.

Proof: By induction on *length*$(\overline{\tau})$.

Base step: $\overline{\tau} \equiv [\,]$
By rule REFL, $\mathbf{ns} \leq_Q \mathbf{ns}$.

Induction step: $\overline{\tau} \equiv [\tau_a, \overline{\tau}_b]$ $\quad \mathbf{ns} \leq_Q \overline{\tau}_b \rightarrow \mathbf{ns}$

$\tau_a \rightarrow \mathbf{ns} \leq_Q \tau_a \rightarrow (\overline{\tau}_b \rightarrow \mathbf{ns})$	(IH and rule F)
$\mathbf{ns} \leq_Q \tau_a \rightarrow \mathbf{ns}$	(rule H)
$\mathbf{ns} \leq_Q \tau_a \rightarrow (\overline{\tau}_b \rightarrow \mathbf{ns})$	(rule TRANS)

(*End of Proof*)

Lemma 26: $(\overline{\tau} \rightarrow \tau_1) \& (\overline{\tau} \rightarrow \tau_2) \leq_Q \overline{\tau} \rightarrow (\tau_1 \& \tau_2)$.

Proof: By induction on *length*$(\overline{\tau})$.

Base step: $\overline{\tau} \equiv [\,]$
By rule REFL, $\tau_1 \& \tau_2 \leq_Q \tau_1 \& \tau_2$.

Induction step: $\overline{\tau} \equiv [\tau_a, \overline{\tau}_b]$ $\quad (\overline{\tau}_b \rightarrow \tau_1) \& (\overline{\tau}_b \rightarrow \tau_2) \leq_Q \overline{\tau}_b \rightarrow (\tau_1 \& \tau_2)$

$\tau_a \rightarrow ((\overline{\tau}_b \rightarrow \tau_1) \& (\overline{\tau}_b \rightarrow \tau_2))$ $\leq_Q \tau_a \rightarrow (\overline{\tau}_b \rightarrow (\tau_1 \& \tau_2))$.	(rules REFL and F)
$(\tau_a \rightarrow (\overline{\tau}_b \rightarrow \tau_1)) \& (\tau_a \rightarrow (\overline{\tau}_b \rightarrow \tau_2))$ $\leq_Q \tau_a \rightarrow ((\overline{\tau}_b \rightarrow \tau_1) \& (\overline{\tau}_b \rightarrow \tau_2))$.	(rule G)
$(\tau_a \rightarrow (\overline{\tau}_b \rightarrow \tau_1)) \& (\tau_a \rightarrow (\overline{\tau}_b \rightarrow \tau_2))$ $\leq_Q \tau_a \rightarrow (\overline{\tau}_b \rightarrow (\tau_1 \& \tau_2))$.	(rule TRANS)

(*End of Proof*)

Lemma 27: $\overline{\tau} \rightarrow lb_Q(\alpha) \leq_Q \overline{\tau} \rightarrow \alpha$.

Proof: By induction on *length*$(\overline{\tau})$.

Base step: $\overline{\tau} \equiv [\,]$
By rule LB, $lb_Q(\alpha) \leq_Q \alpha$.

Induction step: $\overline{\tau} \equiv [\tau_a, \overline{\tau}_b] \quad \overline{\tau}_b \rightarrow lb_Q(\alpha) \leq_Q \overline{\tau}_b \rightarrow \alpha$

$$\begin{array}{lll} & \tau_a \leq_Q \tau_a & \text{(rule REFL)} \\ \Rightarrow & \tau_a \rightarrow (\overline{\tau}_b \rightarrow lb_Q(\alpha)) \leq_Q \tau_a \rightarrow (\overline{\tau}_b \rightarrow \alpha) & \text{(rule F)} \\ = & \overline{\tau} \rightarrow lb_Q(\alpha) \leq_Q \overline{\tau} \rightarrow \alpha. & \end{array}$$

(*End of Proof*)

Theorem 28: Let Q be a constraint list, $\sigma, \tau \in \mathbf{T}[dom(Q)]$, and $\overline{\tau} \in \mathbf{List}(\mathbf{T}[dom(Q)])$. Then

$$le_Q(\sigma, \overline{\tau}, \tau) \Rightarrow \sigma \leq_Q \overline{\tau} \rightarrow \tau.$$

Proof: By induction on the recursion depth of the computation of $le_Q(\sigma, \overline{\tau}, \tau)$. We argue by cases on the outermost invocation of le_Q.

Case 1:

$$\begin{array}{lll} & le_Q(\sigma, \overline{\tau}, \mathbf{ns}) & \\ \Rightarrow & \mathbf{ns} \leq_Q \overline{\tau} \rightarrow \mathbf{ns} & \text{(Lemma 25)} \\ \Rightarrow & \sigma \leq_Q \overline{\tau} \rightarrow \mathbf{ns}. & \text{(Rules D and TRANS)} \end{array}$$

Case 2:

$$\begin{array}{lll} & le_Q(\sigma, \overline{\tau}, \tau_1 \& \tau_2) & \\ = & le_Q(\sigma, \overline{\tau}, \tau_1) \ \wedge \ le_Q(\sigma, \overline{\tau}, \tau_2) & \\ \Rightarrow & \sigma \leq_Q \overline{\tau} \rightarrow \tau_1 \ \wedge \ \sigma \leq_Q \overline{\tau} \rightarrow \tau_2 & \text{(IH)} \\ \Rightarrow & \sigma \leq_Q (\overline{\tau} \rightarrow \tau_1) \& (\overline{\tau} \rightarrow \tau_2) & \text{(rule C)} \\ \Rightarrow & \sigma \leq_Q \overline{\tau} \rightarrow (\tau_1 \& \tau_2). & \text{(Lemma 26 and rule TRANS)} \end{array}$$

Case 3:

$$\begin{array}{lll} & le_Q(\sigma, \overline{\tau}, \tau_1 \rightarrow \tau_2) & \\ = & le_Q(\sigma, [\overline{\tau}, \tau_1], \tau_2) & \\ \Rightarrow & \sigma \leq_Q [\overline{\tau}, \tau_1] \rightarrow \tau_2 & \text{(IH)} \\ = & \sigma \leq_Q \overline{\tau} \rightarrow (\tau_1 \rightarrow \tau_2). & \end{array}$$

Case 4:

$$\begin{array}{lll} & le_Q(\mathbf{ns}, \overline{\tau}, \rho) & \\ = & \mathbf{false} & \\ \Rightarrow & \mathbf{ns} \leq_Q \overline{\tau} \rightarrow \rho. & \text{(trivially)} \end{array}$$

Case 5:

$$\begin{array}{lll} & le_Q(\sigma_1 \& \sigma_2, \overline{\tau}, \rho) & \\ = & le_Q(\sigma_1, \overline{\tau}, \rho) \ \vee \ le_Q(\sigma_2, \overline{\tau}, \rho) & \\ \Rightarrow & \sigma_1 \leq_Q \overline{\tau} \rightarrow \rho \ \vee \ \sigma_2 \leq_Q \overline{\tau} \rightarrow \rho & \text{(IH)} \\ \Rightarrow & \sigma_1 \& \sigma_2 \leq_Q \overline{\tau} \rightarrow \rho \ \vee \ \sigma_1 \& \sigma_2 \leq_Q \overline{\tau} \rightarrow \rho. & \text{(rules A, B, and TRANS)} \end{array}$$

Case 6a:

$$
\begin{array}{lll}
 & le_Q(\sigma_1 \to \sigma_2,\ [\tau_a, \overline{\tau}_b],\ \rho) & \\
= & le_Q(\tau_a,\ [\,],\ \sigma_1) \ \wedge\ le_Q(\sigma_2,\ \overline{\tau}_b,\ \rho) & \\
\Rightarrow & \tau_a \leq_Q \sigma_1 \ \wedge\ \sigma_2 \leq_Q \overline{\tau}_b \to \rho & \text{(IH)} \\
\Rightarrow & \sigma_1 \to \sigma_2 \leq_Q \tau_a \to (\overline{\tau}_b \to \rho). & \text{(rule F)}
\end{array}
$$

Case 6b:

$$
\begin{array}{lll}
 & le_Q(\sigma_1 \to \sigma_2,\ [\,],\ \rho) & \\
= & \text{false} & \\
\Rightarrow & \sigma_1 \to \sigma_2 \leq_Q \rho. & \text{(trivially)}
\end{array}
$$

Case 7a:

$$
\begin{array}{lll}
 & le_Q(\rho,\ [\tau_a, \overline{\tau}_b],\ \rho') & \\
= & \text{false} & \\
\Rightarrow & \rho \leq_Q [\tau_a, \overline{\tau}_b] \to \rho'. & \text{(trivially)}
\end{array}
$$

Case 7b:

$$
\begin{array}{lll}
 & le_Q(\rho,\ [\,],\ \rho') & \\
= & \rho \leq_{pr} \rho' & \\
\Rightarrow & \rho \leq_Q \rho'. & \text{(rule E)}
\end{array}
$$

Case 8:

$$
\begin{array}{lll}
 & le_Q(\alpha,\ \overline{\tau},\ \rho) & \\
= & le_Q(ub_Q(\alpha),\ \overline{\tau},\ \rho) & \\
\Rightarrow & ub_Q(\alpha) \leq_Q \overline{\tau} \to \rho & \text{(IH)} \\
\Rightarrow & \alpha \leq_Q \overline{\tau} \to \rho. & \text{(rules UB and TRANS)}
\end{array}
$$

Case 9:

$$
\begin{array}{lll}
 & le_Q(\mathbf{ns},\ \overline{\tau},\ \alpha) & \\
= & le_Q(\mathbf{ns},\ \overline{\tau},\ lb_Q(\alpha)) & \\
\Rightarrow & \mathbf{ns} \leq_Q \overline{\tau} \to lb_Q(\alpha) & \text{(IH)} \\
\Rightarrow & \mathbf{ns} \leq_Q \overline{\tau} \to \alpha. & \text{(Lemma 27 and rule TRANS)}
\end{array}
$$

Case 10:

$$
\begin{array}{lll}
 & le_Q(\sigma_1 \& \sigma_2,\ \overline{\tau},\ \alpha) & \\
= & le_Q(\sigma_1,\ \overline{\tau},\ \alpha) & \\
\vee & le_Q(\sigma_2,\ \overline{\tau},\ \alpha) & \\
\vee & le_Q(\sigma_1 \& \sigma_2,\ \overline{\tau},\ lb_Q(\alpha)) & \\
\Rightarrow & \sigma_1 \leq_Q \overline{\tau} \to \alpha & \\
\vee & \sigma_2 \leq_Q \overline{\tau} \to \alpha & \\
\vee & \sigma_1 \& \sigma_2 \leq_Q \overline{\tau} \to lb_Q(\alpha) & \text{(IH)} \\
\Rightarrow & \sigma_1 \& \sigma_2 \leq_Q \overline{\tau} \to \alpha & \text{(rule A)} \\
\vee & \sigma_1 \& \sigma_2 \leq_Q \overline{\tau} \to \alpha & \text{(rule B)} \\
\vee & \sigma_1 \& \sigma_2 \leq_Q \overline{\tau} \to \alpha. & \text{(Lemma 27 and rule TRANS)}
\end{array}
$$

Case 11a:

$$
\begin{array}{lll}
 & le_Q(\sigma_1 \to \sigma_2, [\tau_a, \overline{\tau}_b], \alpha) & \\
= & (le_Q(\tau_a, [\,], \sigma_1) \ \wedge \ le_Q(\sigma_2, \overline{\tau}_b, \alpha)) & \\
\vee & le_Q(\sigma_1 \to \sigma_2, [\tau_a, \overline{\tau}_b], lb_Q(\alpha)) & \\
\Rightarrow & (\tau_a \leq_Q \sigma_1 \ \wedge \ \sigma_2 \leq_Q \overline{\tau}_b \to \alpha) & \\
\vee & \sigma_1 \to \sigma_2 \leq_Q [\tau_a, \overline{\tau}_b] \to lb_Q(\alpha) & \text{(IH)} \\
\Rightarrow & \sigma_1 \to \sigma_2 \leq_Q [\tau_c, \overline{\tau}_b] \to \alpha & \text{(rule F)} \\
\vee & \sigma_1 \to \sigma_2 \leq_Q [\tau_a, \overline{\tau}_b] \to \alpha. & \text{(Lemma 27 and rule TRANS)}
\end{array}
$$

Case 11b:

$$
\begin{array}{lll}
 & le_Q(\sigma_1 \to \sigma_2, [\,], \alpha) & \\
= & le_Q(\sigma_1 \to \sigma_2, [\,], lb_Q(\alpha)) & \\
\Rightarrow & \sigma_1 \to \sigma_2 \leq_Q lb_Q(\alpha) & \text{(IH)} \\
\Rightarrow & \sigma_1 \to \sigma_2 \leq_Q \alpha. & \text{(Lemma 27 and rule TRANS)}
\end{array}
$$

Case 12:

$$
\begin{array}{lll}
 & le_Q(\rho, \overline{\tau}, \alpha) & \\
= & le_Q(\rho, \overline{\tau}, lb_Q(\alpha)) & \\
\Rightarrow & \rho \leq_Q \overline{\tau} \to lb_Q(\alpha) & \text{(IH)} \\
\Rightarrow & \rho \leq_Q \overline{\tau} \to \alpha. & \text{(Lemma 27 and rule TRANS)}
\end{array}
$$

Case 13a:

$$
\begin{array}{lll}
 & le_Q(\alpha, [\,], \alpha') & \\
= & (\alpha \equiv \alpha') & \\
\vee & le_Q(ub_Q(\alpha), [\,], \alpha') & \\
\vee & le_Q(\alpha, [\,], lb_Q(\alpha')) & \\
\Rightarrow & \alpha \leq_Q \alpha' & \text{(rule REFL)} \\
\vee & ub_Q(\alpha) \leq_Q \alpha' & \text{(IH)} \\
\vee & \alpha \leq_Q lb_Q(\alpha') & \text{(IH)} \\
\Rightarrow & \alpha \leq_Q \alpha' & \\
\vee & \alpha \leq_Q \alpha' & \text{(rules UB and TRANS)} \\
\vee & \alpha \leq_Q \alpha'. & \text{(Lemma 27 and rule TRANS)}
\end{array}
$$

Case 13b:

$$
\begin{array}{lll}
 & le_Q(\alpha, [\tau_a, \overline{\tau}_b], \alpha') & \\
= & le_Q(ub_Q(\alpha), [\tau_a, \overline{\tau}_b], \alpha') & \\
\vee & le_Q(\alpha, [\tau_a, \overline{\tau}_b], lb_Q(\alpha')) & \\
\Rightarrow & ub_Q(\alpha) \leq_Q [\tau_a, \overline{\tau}_b] \to \alpha' & \\
\vee & \alpha \leq_Q [\tau_a, \overline{\tau}_b] \to lb_Q(\alpha') & \text{(IH)} \\
\Rightarrow & \alpha \leq_Q [\tau_a, \overline{\tau}_b] \to \alpha' & \text{(rules UB and TRANS)} \\
\vee & \alpha \leq_Q [\tau_a, \overline{\tau}_b] \to \alpha'. & \text{(Lemma 27 and rule TRANS)}
\end{array}
$$

(*End of Proof*)

Together, Theorems 24 and 28 assure us that the algorithm le_Q does indeed compute the same relation as $\leq_Q$:

Corollary 29: Let Q be a well-formed constraint list and $\sigma, \tau \in \mathbf{T}[dom(Q)]$. Then

$$\sigma \leq_Q \tau \ \Leftrightarrow \ le_Q(\sigma, [\,], \tau).$$

Proof: By induction on the length of Q. (*End of Proof*)

5 Discussion

5.1 Alternative Decision Procedures

The algorithm in Definition 11 is the only one that we have been able to rigorously prove correct. But there are several others that seem intuitively plausible, and whose proofs of correctness, if they exist, might turn out to be considerably more tractable than the one in Section 4.

The most obvious difficulty with the formulation of le_Q is that the cases for variables and for primitives on the right are extremely similar—each variable case is simply the corresponding primitive case with an extra disjunct where the variable is replaced by its lower bound. Both algorithms presented in this section attempt a "tighter" factoring of the problem.

Another unfortunate property of our existing formulation of le_Q is its (lack of) efficiency. Not only must the left argument be completely analyzed for each primitive encountered on the right, potentially duplicating significant work (especially when $\overline{\tau}$ is nonempty), but worse yet, many nearly identical subcomputations will be generated when a variable appears on the right, because it will be replaced by its lower bound at every step of the analysis of the left argument. Again, both algorithms in this section attempt to reduce duplication of work. (The second one introduces some additional inefficiency of its own, however.)

All three algorithms (the one in Definition 11 and the two about to be discussed) are variants of an algorithm *le*, consisting just of rules 1–7b from Figure 2. It is not hard to show that *le* is a decision procedure for the relation $\leq$—the least relation closed under rules A–TRANS of Figure 1.

5.1.1 Frozen Variables

In the rules in Figure 1, there are three ways that a variable α can be a supertype of a type σ:

1. By rule TRANS or LB;
2. By rule A, B, or REFL;
3. By rule UB.

Let us assume that the first case will take care of itself. (We know that *le* is transitive. If we extend it to handle variables in such a way that the second and third cases work properly, then there should be nothing about the presence of variables that makes the first case any more problematic than it was to begin with. Of course, this reasoning is not strictly valid, since, for example, the middle term θ in rule TRANS can be a variable; this situation can actually be quite delicate.)

In the second case, α behaves exactly like a primitive type: the fact that $\sigma\, le_Q\, \alpha$ has nothing to do with the bounds of α.

In the third case, α behaves exactly like its lower bound.

These observations lead to the following idea: when a variable is encountered as the third argument, we try both replacing it with its lower bound, and replacing it with $freeze(\alpha)$, which is a supertype of α and nothing else. Figure 3 defines an algorithm le^{fr}_Q

(1)	$le^{fr}_Q(\sigma, \overline{\tau}, \mathbf{ns})$	$=$		true
(2)	$le^{fr}_Q(\sigma, \overline{\tau}, \tau_1 \& \tau_2)$	$=$		$le^{fr}_Q(\sigma, \overline{\tau}, \tau_1)$
			$\wedge$	$le^{fr}_Q(\sigma, \overline{\tau}, \tau_2)$
(3)	$le^{fr}_Q(\sigma, \overline{\tau}, \tau_1 \rightarrow \tau_2)$	$=$		$le^{fr}_Q(\sigma, [\overline{\tau}, \tau_1], \tau_2)$
(4)	$le^{fr}_Q(\sigma, \overline{\tau}, \alpha)$	$=$		$le^{fr}_Q(\sigma, \overline{\tau}, lb_Q(\alpha))$
			$\vee$	$le^{fr}_Q(\sigma, \overline{\tau}, \mathit{freeze}(\alpha))$
(5)	$le^{fr}_Q(\mathbf{ns}, \overline{\tau}, \delta)$	$=$		false
(6)	$le^{fr}_Q(\sigma_1 \& \sigma_2, \overline{\tau}, \delta)$	$=$		$le^{fr}_Q(\sigma_1, \overline{\tau}, \delta)$
			$\vee$	$le^{fr}_Q(\sigma_2, \overline{\tau}, \delta)$
(7a)	$le^{fr}_Q(\sigma_1 \rightarrow \sigma_2, [\tau_a, \overline{\tau}_b], \delta)$	$=$		$le^{fr}_Q(\tau_a, [\,], \sigma_1)$
			$\wedge$	$le^{fr}_Q(\sigma_2, \overline{\tau}_b, \delta)$
(7b)	$le^{fr}_Q(\sigma_1 \rightarrow \sigma_2, [\,], \delta)$	$=$		false
(8)	$le^{fr}_Q(\alpha, \overline{\tau}, \delta)$	$=$		$le^{fr}_Q(ub_Q(\alpha), \overline{\tau}, \delta)$
			$\vee$	$le^{fr}_Q(\mathit{freeze}(\alpha), \overline{\tau}, \delta)$
(9a)	$le^{fr}_Q(\delta, [\tau_a, \overline{\tau}_b], \delta')$	$=$		false
(9b)	$le^{fr}_Q(\delta, [\,], \delta')$	$=$		$(\delta \equiv \rho \wedge \delta' \equiv \rho' \wedge \rho \leq_{\mathrm{pr}} \rho')$
			$\vee$	$(\delta \equiv \mathit{freeze}(\alpha) \wedge \delta' \equiv \mathit{freeze}(\alpha))$

Figure 3: Rules for le^{fr}_Q

based on this idea. (The metavariable δ ranges over both primitive types and frozen variables.)

This algorithm is intuitively very plausible, but we have not been able to show that it is equivalent to $\leq_Q$. For every induction hypothesis that we have been able to come up with, the proof of transitivity breaks down at the case $\tau \leq_Q \tau'_1 \& \tau'_2 \leq_Q \rho''$. Indeed, for the most straightforward definition of "transitivity," where frozen variables are taken to belong to the set $\mathbf{P}$ of primitive types, le^{fr}_Q is *not* transitive. Here is the counterexample. Let

$$Q = [\mathbf{int} \leq \alpha \leq \mathbf{real}].$$

Then

$$le^{fr}_Q(\mathbf{int}, [\,], \alpha)$$

and

$$le^{fr}_Q(\alpha, [\,], \mathit{freeze}(\alpha)),$$

but

$$\neg\, le^{fr}_Q(\mathbf{int}, [\,], \mathit{freeze}(\alpha)).$$

5.1.2 Marked Variables

Another idea is to completely discard variables appearing as the third argument to le_Q, replacing them by their lower bounds and continuing until a primitive is reached on the

$$
\begin{array}{lllll}
(1) & le^m_Q(\sigma, \overline{\tau}, \overline{V}, \mathbf{ns}, V) & = & & \text{true} \\
(2) & le^m_Q(\sigma, \overline{\tau}, \overline{V}, \tau_1 \& \tau_2, V) & = & & le^m_Q(\sigma, \overline{\tau}, \overline{V}, \tau_1, V) \\
 & & & \wedge & le^m_Q(\sigma, \overline{\tau}, \overline{V}, \tau_2, V) \\
(3) & le^m_Q(\sigma, \overline{\tau}, \overline{V}, \tau_1 \rightarrow \tau_2, V) & = & & le^m_Q(\sigma, [\overline{\tau}, \tau_1], [\overline{V}, V], \tau_2, V) \\
(4) & le^m_Q(\sigma, \overline{\tau}, \overline{V}, \alpha, V) & = & & le^m_Q(\sigma, \overline{\tau}, \overline{V}, lb_Q(\alpha), V \cup \{\alpha\}) \\
(5) & le^m_Q(\mathbf{ns}, \overline{\tau}, \overline{V}, \rho, V) & = & & \text{false} \\
(6) & le^m_Q(\sigma_1 \& \sigma_2, \overline{\tau}, \overline{V}, \rho, V) & = & & le^m_Q(\sigma_1, \overline{\tau}, \overline{V}, \rho, V) \\
 & & & \vee & le^m_Q(\sigma_2, \overline{\tau}, \overline{V}, \rho, V) \\
(7a) & le^m_Q(\sigma_1 \rightarrow \sigma_2, [\tau_a, \overline{\tau}_b], [V_a, \overline{V}_b], \rho, V) & = & & le^m_Q(\tau_a, [\,], [\,], \sigma_1, \{\}) \\
 & & & \wedge & le^m_Q(\sigma_2, \overline{\tau}_b, \overline{V}_b, \rho, V - V_a) \\
(7b) & le^m_Q(\sigma_1 \rightarrow \sigma_2, [\,], [\,], \rho, V) & = & & \text{false} \\
(8) & le^m_Q(\alpha, \overline{\tau}, \overline{V}, \rho, V) & = & & le^m_Q(ub_Q(\alpha), \overline{\tau}, \overline{V}, \rho, V) \\
 & & & \vee & (\alpha \in V \ \wedge \ \forall U \in \overline{V}.\, \alpha \in U) \\
(9a) & le^m_Q(\rho, [\tau_a, \overline{\tau}_b], [V_a, \overline{V}_b], \rho', V) & = & & \text{false} \\
(9b) & le^m_Q(\rho, [\,], [\,], \rho', V) & = & & (\rho \equiv \rho \ \wedge \ \rho' \equiv \rho' \ \wedge \ \rho \leq_{\mathrm{pr}} \rho')
\end{array}
$$

Figure 4: Rules for le^m_Q

right without ever looking at the first argument. Later, when a variable is enountered on the left, we simply notice whether that variable was passed through during the analysis of the third argument, succeeding immediately if so. The extra parameters of le^m_Q in Figure 4 keep track of the variables that are marked as "already seen on the right." (The metavariable V ranges over finite sets of variables.)

Again, the intuition behind the algorithm is quite simple (though not as simple as the others), and its proof of correctness (if there is one!) ought to be shorter and more elegant than the one in Section 4. We have tried le^m_Q on several examples and even sketched a proof of reflexivity, but, as always, the proof of transitivity raises serious difficulties. The problematic case is the same as for le^{fr}_Q. The solution almost certainly involves finding a clever induction hypothesis.

5.1.3 Canonical sets of types

One other promising direction lies in trying to directly extend a proof technique that gives a very simple proof of the equivalence of $\leq$ and le. We define a notion of *(sets of) canonical types*, and show that the rule of subtyping for these is equivalent to both $\leq$ and le.

Let the function $(-)^*$ be defined as follows:

$$
\begin{array}{lcl}
\mathbf{ns}^* & = & \{\} \\
(\sigma_1 \& \sigma_2)^* & = & \sigma_1{}^* \cup \sigma_2{}^* \\
(\sigma_1 \rightarrow \sigma_2)^* & = & \{\sigma_1 \rightarrow \gamma_2 \mid \gamma_2 \in \sigma_2{}^*\} \\
\rho^* & = & \{\rho\}.
\end{array}
$$

Now define the relation $\leq^*$ simultaneously on canonical types and sets of canonical types as follows (γ ranges over canonical types; $\hat{\tau}$ ranges over sets of canonical types):

$$
\begin{array}{lll}
\sigma \leq^* \sigma' & \text{iff} & \sigma^* \leq^* \sigma'^* \\
\hat{\tau} \leq^* \hat{\tau}' & \text{iff} & \forall \gamma' \in \hat{\tau}'.\ \exists \gamma \in \hat{\tau}.\ \gamma \leq^* \gamma' \\
\hat{\tau}_1 \rightarrow \gamma_2 \leq^* \hat{\tau}_1' \rightarrow \gamma_2' & \text{iff} & \hat{\tau}_1' \leq^* \hat{\tau}_1 \ \wedge \ \gamma_2 \leq^* \gamma_2' \\
\rho \leq^* \rho' & \text{iff} & \rho \leq_{pr} \rho' \\
\rho \not\leq^* \tau_1' \rightarrow \gamma_2' & & \\
\tau_1 \rightarrow \gamma_2 \not\leq^* \rho'. & &
\end{array}
$$

The central difficulty in extending this approach to types with variables is that whereas here the set of canonical types on the left of $\leq^*$ represents a disjunction and the set of the right represents a conjunction, the need to replace variables on the right with either their lower bounds *or* with something like frozen versions of themselves leads to disjunctions on the right as well.

5.2 Extensions

This section briefly mentions some of the ways in which the work presented in this paper should be extended.

5.2.1 Bounded Quantifiers

The present formulation of types and subtyping was always intended to be a step toward working with types containing bounded quantifiers. This "easy first step" turned out to present sufficient challenge that we have not yet thought seriously about the general case.

Two directions of study seem worth pursuing:

Prenex bounded quantification stands in the same relation to the pure system of intersection types as ML's prenex quantifiers to the system of simple types. We know of no current research into extending this system of types with bounds on the quantifiers or into combining it with intersection types, but believe that both may be profitable.

Full bounded quantification has recently become an object of considerable interest in the theoretical research community. Our ultimate goal is to show how this system may be combined with intersection types.

5.2.2 Record Types

Conjunctive types allow records and record inheritance to be treated in a very elegant way. For each field label ι, we add a unary type constructor "ι :" taking, for example, the type **int** to the type of records with a single field labeled ι, of type **int**. The types of records with multiple fields are just conjunctions of single-field record types. If we assume that ι : is monotone, then the usual subtyping rule for records [Car84,CW85] falls out automatically.

Adding record types to the definitions, algorithms, and proofs presented in this paper should be a straightforward process.

5.2.3 A void Type

Having to specify both upper and lower bounds for every variable makes it impossible to use bounded quantification in one of the ways that is most often proposed. In the literature, a bounded quantifier is normally used with the intended interpretation, "For any type α that is a subtype of τ..."—for example, "For every record type with at least the fields a and b...." But there are an infinite number of these, of which none is the "smallest." (Records are not crucial to this problem. It is also possible to construct infinite descending chains of types in the system without records.)

The solution here is to add a "bottom" element to the preorder of types, perhaps called void because it has no instances. We have not examined this possibility in depth, but we know of no immediate difficulties that would arise.

5.2.4 Disjunctive Types

Another infelicity of the present formulation of bounded variables is that it is only possible to declare a single upper bound and a single lower bound for each variable. This is not so bad in the case of upper bounds, since two upper bounds can simply be combined into one using &. But two lower bounds must be combined by explicitly calculating their least upper bound (which always exists, fortunately).

The most obvious solution is to introduce "disjunctive" types. Some of the difficulties arising from attempts to do this are discussed in [Rey88].

6 Acknowledgements

John Reynolds suggested the problem of proving the equivalence of le_Q and $\leq_Q$, and provided valauble advice and guidance toward its solution. Bob Harper and Peter Lee gave thoughtful comments on a previous draft of this paper.

References

[BCD83] H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4), 1983.

[Car84] Luca Cardelli. A semantics of multiple inheritance. In G. Kahn, D. MacQueen, and G. Plotkin, editors, *Semantics of Data Types*, pages 51–67, Springer-Verlag, 1984.

[CDV80] M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Principal type schemes and lambda calculus semantics. In *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 535–560, Academic Press, New York, 1980.

[CDV81] Mario Coppo, Maria Dezani-Ciancaglini, and B. Venneri. Functional character of solvable terms. *Zeitschrift für mathematische Logic und Grundlagen der Mathematik*, 27:45–58, 1981.

[CW85] Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *Computing Surveys*, December 1985.

[Mil78] Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17:348–375, August 1978.

[RdRV84] S. Ronchi della Rocca and B. Venneri. Principal type schemes for an extended type theory. *Theoretical Computer Science*, 28:151–169, 1984.

[Rey88] John C. Reynolds. *Preliminary Design of the Programming Language Forsythe.* Technical Report CMU-CS-88-159, Carnegie Mellon University, June 1988.

[Rey89] John C. Reynolds. *Syntactic Control of Interference, Part 2.* Report CMU-CS-89-130, Carnegie Mellon University, April 1989.