Math. Struct. in Comp. Science (1996), *vol.* 11, *pp.* 1–000

Intersection Types and Bounded Polymorphism

BENJAMIN C. PIERCE

Computer Laboratory, University of Cambridge

Received 1994

Intersection types and bounded quantification are complementary extensions of a first-order programming language with subtyping. We define a typed λ-calculus combining these extensions, illustrate its unusual properties, and develop basic proof-theoretic and semantic results leading to algorithms for subtyping and typechecking.

1. Introduction

Intersection types were developed in the late 1970s by Coppo and Dezani-Ciancaglini (1978), and independently by Sallé (Coppo *et al.*, 1979; Sallé, 1982) and Pottinger (1980). Since then, they have been studied extensively by members of the group at Torino and many others (Barendregt *et al.*, 1983; Cardone & Coppo, 1990; Coppo & Dezani-Ciancaglini, 1980; Coppo *et al.*, 1983; Coppo *et al.*, 1980; Coppo *et al.*, 1981; Coppo *et al.*, 1987; Dezani-Ciancaglini & Margaria, 1984; Dezani-Ciancaglini & Margaria, 1986; Hindley, 1982; Reynolds, 1988; Reynolds, 1991; Ronchi della Rocca, 1988; Ronchi della Rocca & Venneri, 1984; van Bakel, 1991; van Bakel, 1992). Various extensions of the original intersection type discipline have also been explored, including the notion of infinite intersections (Leivant, 1990) and the dual notion of union types (Barbanera & Dezani-Ciancaglini, 1991; Hayashi, 1991; Pierce, 1991b; Cardone *et al.*, 1994; Barbanera *et al.*, 1995). Some related extensions to ML-style type inference systems are the notions of refinement types (Freeman & Pfenning, 1991; Hayashi, 1991; Pierce, 1991b) and soft typing (Cartwright & Fagan, 1991; Fagan, 1990; Wright & Cartwright, 1994), as well as work by Coppo and Giannini (1995) and Jim (Jim, 1996) on type inference for decidable restrictions of intersection types. Reynolds made the first demonstration that intersection types can be used as the basis for practical programming languages (1981; 1988).

Among the most intriguing properties of intersection types is their ability to express an essentially unbounded (though finite) amount of information about a program. For example, the addition function $+$ can be given the type $Int{\to}Int{\to}Int \wedge Real{\to}Real{\to}Real$, capturing both the general fact that the sum of two real numbers is always real and the more specialized observation that the sum of two integers is an integer. A compiler for a language with intersection types might even provide two different object-code sequences for $+$, one using a floating point addition instruction and one using integer addition. For each instance of $+$ in a program, the compiler would check whether both arguments are integers and generate the more efficient code sequence in this case. This kind of *finitary polymorphism* or *coherent overloading* is so expressive that the set of all valid typings for an untyped λ-term amounts to a complete characterization of its behavior (cf. (Cardone & Coppo, 1990)).

Bounded quantification was introduced by Cardelli and Wegner (1985) in the language Fun. Based on informal ideas by Cardelli and formalized using techniques developed by Mitchell (1984), Fun integrated Girard-Reynolds polymorphism (Girard, 1972; Reynolds, 1974) with Cardelli's first-order calculus of subtyping (1984). The original Fun was simplified and slightly generalized by Bruce and Longo (1990), and again by Curien and Ghelli (1992). Curien and Ghelli's formulation, called *minimal Bounded Fun* or $F_{\leq}$ ("F-sub"), has become standard and is the one considered here. Cardelli and Wegner's paper gives the first programming examples using bounded quantification; more are developed in Cardelli's study of power kinds (1988). Curien and Ghelli (Curien & Ghelli, 1992; Ghelli, 1990) address a number of syntactic properties of $F_{\leq}$. Semantic aspects of closely related systems have been studied by Bruce and Longo (1990), Martini (1988), Breazu-Tannen, Coquand, Gunter, and Scedrov (1991), Cardone (1989), Cardelli and Longo (1991), Cardelli, Martini, Mitchell, and Scedrov (1994), Curien and Ghelli (1992; 1991), and Bruce and Mitchell (1992). $F_{\leq}$ has been extended to include record types and richer notions of subtyping by Cardelli and Mitchell (1991), Bruce (1991), Cardelli (1992), and Canning, Cook, Hill, Olthoff, and Mitchell (1989a), and Pierce and Steffen (1996). It forms the basis of several recent accounts of object-oriented programming (Bruce, 1994; Pierce & Turner, 1994; Hofmann & Pierce, 1995; Abadi *et al.*, 1996). Bounded quantification also plays a key role in Cardelli's programming language Quest (1991; 1991) and in the Abel language developed at HP Labs (Canning *et al.*, 1989b; Canning *et al.*, 1989a; Canning *et al.*, 1988; Cook *et al.*, 1990), in Mitchell's Rapide language (Mitchell *et al.*, 1991; Katiyar *et al.*, 1994) and in Bruce's TOOPL (1994) and TOIL (1995).

Cardelli and Wegner integrated impredicative polymorphism with the notion of subtyping by allowing a quantified type to give a *bound* for its parameter; for example, an element of the type $\forall\alpha{\leq}\mathit{Student}.\ \mathit{List}(\alpha)\to\mathit{List}(\alpha)$ takes as its first parameter an arbitrary subtype of the type *Student* and returns a function on lists of this type. This *universal* or *parametric polymorphism* is both broader (since the number of possible instantiations of a polymorphic type is infinite) and more rigid (since all instances must have the same basic shape) than the finitary polymorphism given by intersection types. Its main practical advantages are brevity and compile-time efficiency: it allows polymorphic expressions to be written, typechecked, and compiled just once.

Intersection types and bounded quantification thus complement and, in a sense, complete each another: universal quantifiers can be used to compactly capture the common cases where all the desired types of a term are structurally similar, which lead to long and unwieldy typings when expressed only in terms of intersections, while intersections allow a degree of refinement that is impossible with pure polymorphism. Moreover, intersections can be used in the bounds of quantifiers to express a simple form of "multiple inheritance"; for example, the type $\forall\alpha{\leq}\mathit{Student}\wedge\mathit{Employee}.\ \mathit{List}(\alpha)\to\mathit{List}(\alpha)$ describes polymorphic functions applicable only to the common subtypes of *Student* and *Employee*. Indeed, a higher-order extension of the system described in this paper (Compagnoni, 1994; Compagnoni, 1995) has been used in a type-theoretic model of object-oriented multiple inheritance (Compagnoni & Pierce, 1996). Related calculi combining restricted forms of intersection types with higher-order polymorphism and dependent types have been studied by Pfenning (Pfenning, 1993).

Following a more detailed discussion of the pure systems of intersections and bounded quantification (Section 2), we describe, in Section 3, a typed λ-calculus called $F_\wedge$ ("F-meet") integrating the features of both. Section 4 gives some examples illustrating this system's expressive power. Section 5 presents the main results of the paper: a proof-theoretic analysis of $F_\wedge$'s subtyping and typechecking relations leading to algorithms for checking subtyping and for synthesizing minimal types for terms. Section 6 discusses semantic aspects of the calculus, obtaining a simple soundness proof for the typing rules by interpreting types as partial equivalence relations; however, another proof-theoretic result, the nonexistence of least upper bounds for arbitrary pairs of types, implies that typed models may be more difficult to construct. Section 7 offers concluding remarks.

2. Background

We set the stage for the $F_\wedge$ calculus by establishing notational conventions and reviewing its two immediate ancestors. We begin with a common core calculus, a simply typed λ-calculus with subtyping, and then discuss two extensions, a first-order calculus with intersection types and a second-order calculus with bounded quantification.

2.1. *Notational Preliminaries*

The metavariables α and β range over type variables; σ, τ, θ, ϕ, and ψ range over types; e and f range over terms; x and y range over term variables. A finite sequence with elements x_1 through x_n is written $[x_1..x_n]$. Concatenation of finite sequences is written $X_1 * X_2$. Single elements are adjoined to the right or left of sequences with a comma: $[x_a, X_b]$ or $[X_a, x_b]$. Sequences are sometimes written using the "comprehension" notation $[x \mid \ldots]$; for example, if $T \equiv [\sigma{\rightarrow}\tau,\ \psi{\rightarrow}\psi,\ \sigma{\rightarrow}\theta]$, then the comprehension $[\zeta_2 \mid \zeta_1{\rightarrow}\zeta_2 \in T \text{ and } \zeta_1{\equiv}\sigma]$ stands for the sequence $[\tau, \theta]$. A *context* Γ is a finite sequence of typing assumptions $x{:}\tau$ and subtyping assumptions $\alpha{\leq}\tau$. It is convenient to view a context Γ as a finite function and write $dom(\Gamma)$ for its domain; the *range* of Γ is the collection of right-hand sides of bindings in Γ. $\Gamma(x)$ denotes the type of x in Γ, if it has one; similarly $\Gamma(\alpha)$ denotes the upper bound of α in Γ, if it has one. The set of free variables of a term e is written $FV(e)$.

A type τ is *closed* with respect to a context Γ if its free (type) variables are all in $dom(\Gamma)$. A term e is closed with respect to Γ if its free (term and type) variables are all in $dom(\Gamma)$. A context Γ is closed if $\Gamma \equiv \{\}$, or if $\Gamma \equiv \Gamma_1, \alpha{\leq}\tau$ with Γ_1 closed and τ closed with respect to Γ_1, or if $\Gamma \equiv \Gamma_1, x{:}\tau$ with Γ_1 closed and τ closed with respect to Γ_1. A subtyping statement $\Gamma \vdash \sigma \leq \tau$ is closed if Γ is closed and σ and τ are closed with respect to Γ; a typing statement $\Gamma \vdash e \in \tau$ is closed if Γ is closed and e and τ are closed with respect to Γ. In the following, we assume that all statements under discussion are closed; in particular, we allow only closed statements in instances of inference rules.

Types, terms, contexts, and statements that differ only in the names of bound variables are considered identical. That is, we think of variables not as names but as pointers to their binders in the surrounding context, as suggested by deBruijn (1972).

Examples are set in a typewriter font; λ-calculus notation is transliterated as follows: $\top$ is written as `T`, λ as `\`, Λ as `\\`, $\forall$ as `All`, and $\leq$ as `<`. Lines of input to the typechecker are prefixed with `>` and followed by the system's response. The type constructors $\rightarrow$ and $\forall$ bind more tightly than $\wedge$. Also, $\rightarrow$ associates to the right and $\forall$ obeys the usual "dot rule" where the body τ of a quantified type $\forall\alpha{\leq}\sigma.\ \tau$ is taken to extend to the right as far as possible.

2.2. *Simply Typed λ-Calculus with Subtyping*

The $F_\wedge$ calculus may be viewed as a "least upper bound" of two calculi: a first-order λ-calculus with intersection types ($\lambda_\wedge$) and a second-order λ-calculus with bounded quantification ($F_\leq$). These, in turn, are both extensions of the simply typed λ-calculus enriched with a subtyping relation ($\lambda_\leq$). The latter system was proposed by Cardelli (1984) as a "core calculus of subtyping" in a foundational framework for object-oriented programming languages. We briefly review its definition.

2.2.1. Definition: The types of $\lambda_\leq$ consist of a set of primitive types (ranged over by the metavariable ρ) closed under the function space type constructor $\rightarrow$:

$$\tau \quad ::= \quad \rho \quad | \quad \tau_1{\rightarrow}\tau_2$$

The terms of $\lambda_\leq$ consist of a countable set of variables (ranged over by x), together with all the phrases that can be built from these by functional abstraction and application:

$$e \quad ::= \quad x \quad | \quad \lambda x{:}\tau.\ e \quad | \quad e_1\ e_2$$

The presence of the *domain-type annotation* τ in the syntax of λ-abstractions marks a fundamental design choice, which we shall maintain throughout the paper: all of the calculi we consider are *explicitly typed* systems (as opposed to *type assignment* systems). This requires that a programmer exert firm control over the typechecker's behavior, making programs more verbose but rendering typechecking decidable in many cases where type inference would be undecidable.

2.2.2. Definition: A $\lambda_\leq$ context is a sequence of typing assumptions

$$\Gamma \quad ::= \quad \{\} \quad | \quad \Gamma,\ x{:}\tau.$$

The typing relation of $\lambda_\leq$ is formalized as a collection of inference rules for deriving typing statements of the form $\Gamma \vdash e \in \tau$ ("under assumptions Γ, expression e has type τ"), where Γ contains a typing assumption for each of the free variables of e. The rules for variables, abstractions, and applications are exactly the same as in the ordinary simply typed λ-calculus (Church, 1940). In addition, we introduce a rule of *subsumption* stating that whenever a term e has a type σ and σ is a subtype of τ, the type of e may be promoted to τ.

2.2.3. Definition: The $\lambda_\leq$ typing relation $\Gamma \vdash e \in \tau$ is the least three-place relation closed under the following rules:

$$\Gamma \vdash x \in \Gamma(x) \tag{Var}$$

$$\frac{\Gamma,\, x{:}\tau_1 \;\vdash\; e \;\in\; \tau_2}{\Gamma \;\vdash\; \lambda x{:}\tau_1.\; e \;\in\; \tau_1 \to \tau_2} \qquad \text{(Arrow-I)}$$

$$\frac{\Gamma \;\vdash\; e_1 \;\in\; \tau_1 \to \tau_2 \qquad \Gamma \;\vdash\; e_2 \;\in\; \tau_1}{\Gamma \;\vdash\; e_1\, e_2 \;\in\; \tau_2} \qquad \text{(Arrow-E)}$$

$$\frac{\Gamma \;\vdash\; e \;\in\; \tau_1 \qquad \Gamma \;\vdash\; \tau_1 \;\leq\; \tau_2}{\Gamma \;\vdash\; e \;\in\; \tau_2} \qquad \text{(Sub)}$$

This definition may be viewed in two different ways:

1 as a three-place relation constructed as the limit of a sequence beginning with the empty relation and successively enriching it according to the rules Var, Arrow-I, Arrow-E, and Sub, or

2 as a simple logic whose derivable judgements are those appearing as conclusions of valid derivation trees built from these rules.

We adopt both views, interchangeably, in what follows. We sometimes write $c :: J$ to mean "c is a derivation tree whose final conclusion is the statement J."

By analogy with types, we might expect the definition of $\lambda_{\leq}$ terms to include a collection of constants. These can be added to the calculus, but they are not strictly necessary: we can write programs involving "built-in values" like numbers and arithmetic operators simply by considering these as variables and providing a *pervasive context* — call it Γ_P — that assigns them appropriate types. When contexts are extended in Section 2.4 to allow assumptions for type variables, primitive types can also be dropped, although not every conceivable ordering on the primitive types can be encoded as a pervasive context (cf. 3.3.2.4).

It remains to define the subtype relation. Intuitively, a subtyping statement $\Gamma \vdash \sigma \leq \tau$ corresponds to the assertion that σ is a *refinement* of τ, in the sense that every element of σ contains enough information to meaningfully be regarded as an element of τ. In some models this means simply that σ is a subset of τ; more generally, it implies the existence of a distinguished *coercion function* from σ to τ.

These considerations immediately entail that the subtype relation should be both reflexive and transitive — i.e., that it should be a preorder. We assume that the subtype relation on primitive types is given in advance by some preorder $\leq_P$. This relation is extended to the smallest preorder closed under the following subtyping rule for function types: $\sigma_1 \to \sigma_2$ is a subtype of $\tau_1 \to \tau_2$ iff τ_1 is a subtype of σ_1 and σ_2 is a subtype of τ_2. Notice that, as usual, this relation is *covariant* in the right-hand side and *contravariant* in the left-hand side of the $\to$ constructor: a collection of functions can be refined either by narrowing the range into which their results must fall or by enlarging the domain over which they must behave properly.

A key feature of this notion of subtyping is that it is *structural*: the ordering of two arrow types is completely determined by their left- and right-hand sides. In logical terms, the only extended theories we consider are those whose non-logical rules are restricted to statements about primitive types. This feature is retained in the other calculi we consider.

In fact, the subtype relation of $\lambda_{\leq}$ is not only structural, but *compositional*: the ordering on arrow types may be computed as a function of the ordering of their left- and right-hand sides. The introduction of intersection types in the next section will invalidate this stronger property.

2.2.4. Definition: The $\lambda_{\leq}$ subtyping relation $\Gamma \vdash \sigma \leq \tau$ is the least three-place relation closed under the following rules:

$$\Gamma \vdash \tau \leq \tau \qquad \text{(Sub-Refl)}$$

$$\frac{\Gamma \vdash \tau_1 \leq \tau_2 \qquad \Gamma \vdash \tau_2 \leq \tau_3}{\Gamma \vdash \tau_1 \leq \tau_3} \qquad \text{(Sub-Trans)}$$

$$\frac{\Gamma \vdash \rho_1 \leq_P \rho_2}{\Gamma \vdash \rho_1 \leq \rho_2} \qquad \text{(Sub-Prim)}$$

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \sigma_1 \to \sigma_2 \leq \tau_1 \to \tau_2} \qquad \text{(Sub-Arrow)}$$

Of course, the context Γ plays no part in these rules and could be dropped without changing the system. We include it here for notational compatibility with later systems, in which contexts will also contain subtyping assumptions about type variables.

When $\Gamma \vdash \sigma \leq \tau$, we call τ a *supertype* of σ. When $\Gamma \vdash \sigma \leq \tau$ and $\Gamma \vdash \tau \leq \sigma$, we say that σ and τ are *equivalent* under Γ, written $\Gamma \vdash \sigma \sim \tau$. We write $\Gamma \not\vdash \sigma \leq \tau$ to *deny* the derivability of the statement $\Gamma \vdash \sigma \leq \tau$.

2.3. *Intersection Types*

The original motivation for the introduction of intersection types was the desire for a type-assignment system in the spirit of Curry (Curry & Feys, 1958), but with two additional properties:

1 The typing of a term should be invariant under β-conversion. (Under Curry's system, β-reduction preserves types but β-expansion, in general, does not.)

2 Every term possessing a normal form should be given a meaningful typing.

2.3.1. Definition: The first-order calculus of intersection types, $\lambda_{\wedge}$, is formed from $\lambda_{\leq}$ by adding intersections to the language of types:

$$\tau \quad ::= \quad \rho \quad | \quad \tau_1 \to \tau_2 \quad | \quad \bigwedge[\tau_1..\tau_n]$$

For presenting examples, we introduce the following abbreviations:

$$\begin{array}{lll} \sigma \wedge \tau & \stackrel{\text{def}}{=} & \bigwedge[\sigma, \tau] \\ \top & \stackrel{\text{def}}{=} & \bigwedge[]. \end{array}$$

(Of course, the whole system could equally well be formulated in terms of a binary constructor $\wedge$ and a nullary constructor $\top$.) The notations $\sigma \cap \tau$, $\sigma \& \tau$, and $\sigma \wedge \tau$ have all been used in the literature to denote the intersection of σ and τ; the universal type (usually corresponding to a nullary intersection) has been written as both ω and *ns* ("nonsense"). The phrase $\sigma \wedge \tau$ (or $\bigwedge[\sigma, \tau]$) is pronounced "σ meet τ," "σ intersect τ," or "σ and τ." For the nullary intersection, we use the symbol $\top$ ("top") by analogy with the binary case.

Two new subtyping rules capture the order-theoretic properties of the $\wedge$ operator:

$$\frac{\text{for all } i,\ \Gamma \vdash \sigma \leq \tau_i}{\Gamma \vdash \sigma \leq \bigwedge[\tau_1..\tau_n]} \qquad \text{(SUB-INTER-G)}$$

$$\Gamma \vdash \bigwedge[\tau_1..\tau_n] \leq \tau_i \qquad \text{(SUB-INTER-LB)}$$

(Complete sets of inference rules for $\lambda_\wedge$ and the other systems introduced in this paper appear in Appendix B.)

One additional subtyping rule captures the relation between intersections and function spaces, allowing the two constructors to "distribute" when an intersection appears on the right-hand side of an arrow:

$$\Gamma \vdash \bigwedge[\sigma\to\tau_1 \,..\, \sigma\to\tau_n] \leq \sigma \to \bigwedge[\tau_1..\tau_n] \qquad \text{(SUB-DIST-IA)}$$

(This inclusion is actually an equivalence, since the other direction may be proved from the rules for meets and arrows.) This rule will have a strong effect on both syntactic and semantic properties of the language. For example, it implies that $\top \leq \sigma\to\top$ for any σ.

The typing rules must also be extended slightly. As for any type constructor, we expect to find a pair of an introduction rule, by which terms can be shown to possess intersection types, and an elimination rule, by which this fact may later be exploited. The introduction rule allows an intersection type to be derived for a term whenever each of the elements of the intersection can be derived for it separately:

$$\frac{\text{for all } i,\ \Gamma \vdash e \in \tau_i}{\Gamma \vdash e \in \bigwedge[\tau_1..\tau_n]} \qquad \text{(INTER-I)}$$

The corresponding elimination rule would allow us to infer, on the basis of a derivation of a statement like $\Gamma \vdash e \in \bigwedge[\tau_1..\tau_n]$, that e possesses every τ_i individually. But this follows already from the rule SUB-INTER-LB and the rule of subsumption; we need not add the elimination rule explicitly to the calculus.

The nullary case of this rule is worth particular notice, since it allows the type $\top$ to be derived for *every* term of the calculus, including terms whose evaluation encounters a run time error or fails to terminate.

The system as we have described it so far supports the use of intersection types in programming only to a limited degree. Suppose, for example, that the primitive subtype relation has $Int \leq Real$ and the addition function in the pervasive context is overloaded to operate on both integers and reals:

$$\Gamma_P(+) = Int\to Int\to Int \wedge Real\to Real\to Real.$$

Expressions involving addition of integers and reals will be given type *Int* if possible, otherwise type *Real*:

```
> x = plus 0 0;
x : Int

> x = plus pi pi;
x : Real
```

```
> x = plus 0 pi;
x : Real
```

(Note that the typechecker attempts to simplify the type it derives for each term, so that, for example, the type of `plus 0 0` is printed as `Int` instead of as `Int/\Real`.)

But, using just the constructs introduced so far, there is no way of writing new functions that behave in this way. For example, the doubling function $\lambda x{:}?.\ x + x$ cannot be given the type $Int \to Int \wedge Real \to Real$, since replacing the ? with either *Int* or *Real* (or even $Int \wedge Real$) gives a typing that is too restrictive:

```
> double1 = \x:Int. plus x x;
double1 : Int -> Int

> double2 = \x:Real. plus x x;
double2 : Real -> Real

> double3 = \x:Int/\Real. plus x x;
double3 : Int -> Int
```

This led Reynolds (1988) to introduce a generalized form of λ-abstraction allowing explicit programmer-controlled generation of alternative typings for terms:

$$e \quad ::= \quad \ldots \quad | \quad \lambda x{:}\tau_1..\tau_n.\ e$$

The typing rule for this form allows the typechecker to make a choice of any of the σ's as the type of x in the body:

$$\frac{\Gamma,\ x{:}\sigma_i\ \vdash\ e\ \in\ \tau_i}{\Gamma\ \vdash\ \lambda x{:}\sigma_1..\sigma_n.\ e\ \in\ \sigma_i \to \tau_i} \qquad \text{(ARROW-I')}$$

This rule can be used together with INTER-I to generate a set of up to n alternate typings for the body and then form their intersection as the type of the λ-abstraction:

```
> double = \x:Int,Real. plus x x;
double : Real->Real /\ Int->Int
```

One peculiar property of the generalized λ is that adding extra alternatives to the set of possible domain types for x can only improve the typing of the whole expression. If some alternative results in a "typechecking failure," the best type for the body under this assumption will be equivalent to $\top$ (typically via the SUB-DIST-IA rule), and may therefore be dropped from the final type of the expression without changing its equivalence class in the subtype ordering:

```
> double = \x:Int,Real,Char. plus x x;
double : Int->(Real/\Int) /\ Real->/\[Real] /\ Char->T
    i.e. Real->Real /\ Int->Int
```

When necessary to prevent confusion with other calculi, turnstiles in $\lambda_\wedge$ derivations are written $\vdash^{\lambda\wedge}$.

2.4. *Bounded Polymorphism*

We now review the other major subsystem of our calculus, System $F_{\leq}$. aLike other second-order λ-calculi, the terms of $F_{\leq}$ include the variables, abstractions, and applications of $\lambda_{\leq}$, plus the type abstractions and type applications of the second-order λ-calculus. The latter are slightly refined to take account of the subtype relation: each type abstraction gives a *bound* for the type variable it introduces and each type application must satisfy the constraint that the argument type is a subtype of the bound of the polymorphic function being applied. Also, like that of $\lambda_{\wedge}$, the $F_{\leq}$ subtype ordering includes a maximal element. Since the two are not exactly the same (cf. 3.3.1), the maximal $F_{\leq}$ type is called by its conventional name, *Top*, instead of $\top$.

$$\begin{array}{lcl} \tau & ::= & Top \quad | \quad \alpha \quad | \quad \tau_1 \to \tau_2 \quad | \quad \forall\alpha{\leq}\tau_1.\ \tau_2 \\ e & ::= & x \quad | \quad \lambda x{:}\tau.\ e \quad | \quad e_1\ e_2 \quad | \quad \Lambda\alpha{\leq}\tau.\ e \quad | \quad e[\tau] \end{array}$$

To accomodate the subtyping assumptions introduced by type abstractions, we enrich the contexts of the previous systems to include bindings for both term variables and type variables.

Type abstractions have almost the same typing rule as in other second-order λ-calculi; they are checked by moving their stated bound for the type variable they introduce into the context and checking the body of the abstraction under the enriched set of assumptions:

$$\frac{\Gamma, \alpha{\leq}\tau_1 \vdash e \in \tau_2}{\Gamma \vdash \Lambda\alpha{\leq}\tau_1.\ e \in \forall\alpha{\leq}\tau_1.\ \tau_2} \qquad \text{(ALL-I)}$$

Type applications check that the type being passed as a parameter is indeed a subtype of the bound of the corresponding quantifier:

$$\frac{\Gamma \vdash e \in \forall\alpha{\leq}\tau_1.\ \tau_2 \qquad \Gamma \vdash \tau \leq \tau_1}{\Gamma \vdash e[\tau] \in \{\tau/\alpha\}\tau_2} \qquad \text{(ALL-E)}$$

The subtype relation of $\lambda_{\leq}$ is also extended with several rules. First, we stipulate that *Top* is a maximal element of the subtype order:

$$\Gamma \vdash \sigma \leq Top \qquad \text{(SUB-TOP)}$$

One of the main uses of *Top* — in fact, the original reason it was introduced by Cardelli and Wegner — is to recover ordinary unbounded quantification as a special case of bounded quantification: $\forall\alpha.\ \tau$ becomes $\forall\alpha{\leq} Top.\ \tau$.

Type variables are subtypes of the bounds given for them in the prevailing context:

$$\Gamma \vdash \alpha \leq \Gamma(\alpha) \qquad \text{(SUB-TVAR)}$$

Like arrow types, subtyping of quantified types is contravariant in their bounds and covariant in their bodies:

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma, \alpha{\leq}\tau_1 \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \forall\alpha{\leq}\sigma_1.\ \sigma_2 \leq \forall\alpha{\leq}\tau_1.\ \tau_2} \qquad \text{(SUB-ALL)}$$

This rule deserves a closer look, since it causes considerable difficulties (cf. Section 6.2 and (Ghelli, 1995; Pierce, 1994; Ghelli, 1993)). Intuitively, it reads as follows:

> A type $\tau \equiv \forall\alpha{\leq}\tau_1.\ \tau_2$ describes a collection of polymorphic values (functions from types to values), each mapping subtypes of τ_1 to instances of τ_2. If τ_1 is a subtype of σ_1, then the domain of τ is smaller than that of $\sigma \equiv \forall\alpha{\leq}\sigma_1.\ \sigma_2$, so σ is a stronger constraint and describes a smaller collection of polymorphic values. Moreover, if, for each type θ that is an acceptable argument to the functions in both collections (i.e., one that satisfies the more stringent requirement $\theta \leq \tau_1$), the θ-instance of σ_2 is a subtype of the θ-instance of τ_2, then σ is a "pointwise stronger" constraint and again describes a smaller collection of polymorphic values.

Thus, quantified types may be thought of as a kind of function spaces. We sometimes abuse this analogy and speak of the bound and body of a quantified type as its "left-hand" and "right-hand" sides.

The rules defining $F_\leq$ do not directly constitute an algorithm for checking the subtype relation, since they are not syntax-directed. In particular, the rule TRANS cannot effectively be applied backwards, since this would involve "guessing" an appropriate value for the intermediate type τ_2. Curien and Ghelli (as well as Cardelli and others) use the following reformulation:

2.4.1. Definition: $F_\leq^N$ ("N" for "normal form") is the least relation closed under the following rules:

$$\Gamma \vdash \sigma \leq Top \qquad \text{(NTOP)}$$

$$\Gamma \vdash \alpha \leq \alpha \qquad \text{(NREFL)}$$

$$\frac{\Gamma \vdash \Gamma(\alpha) \leq \tau}{\Gamma \vdash \alpha \leq \tau} \qquad \text{(NVAR)}$$

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \sigma_1 \to \sigma_2 \leq \tau_1 \to \tau_2} \qquad \text{(NARROW)}$$

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma, \alpha{\leq}\tau_1 \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \forall\alpha{\leq}\sigma_1.\ \sigma_2 \leq \forall\alpha{\leq}\tau_1.\ \tau_2} \qquad \text{(NALL)}$$

The reflexivity rule here is restricted to type variables. Transitivity is eliminated, except for instances of the form

$$\frac{\Gamma \vdash \alpha \leq \Gamma(\alpha) \qquad \Gamma \vdash \Gamma(\alpha) \leq \tau}{\Gamma \vdash \alpha \leq \tau,}$$

which are packaged together as instances of the new rule NVAR.

These rules may be read as a (semi-) algorithm (i.e., a recursively defined procedure, not necessarily terminating) for checking the subtype relation:

$check(\Gamma \vdash \sigma \leq \tau) =$

1. *if* $\tau \equiv Top$
   then true
2. *else if* $\sigma \equiv \sigma_1 \rightarrow \sigma_2$ *and* $\tau \equiv \tau_1 \rightarrow \tau_2$
   then $check(\Gamma \vdash \tau_1 \leq \sigma_1)$
   and $check(\Gamma \vdash \sigma_2 \leq \tau_2)$
3. *else if* $\sigma \equiv \forall\alpha{\leq}\sigma_1.\ \sigma_2$ *and* $\tau \equiv \forall\alpha{\leq}\tau_1.\ \tau_2$
   then $check(\Gamma \vdash \tau_1 \leq \sigma_1)$
   and $check(\Gamma,\ \alpha{\leq}\tau_1 \vdash \sigma_2 \leq \tau_2)$
4. *else if* $\sigma \equiv \alpha$ *and* $\tau \equiv \alpha$
   then true
5. *else if* $\sigma \equiv \alpha$
   then $check(\Gamma \vdash \Gamma(\alpha) \leq \tau)$

n. *else*
   false.

We write $F^N_{\leq}$ to refer either to the algorithm or to the inference system.

2.4.2. Lemma: [Curien and Ghelli] The relations $F_{\leq}$ and $F^N_{\leq}$ coincide: $\Gamma \vdash \sigma \leq \tau$ is derivable in $F_{\leq}$ iff it is derivable in $F^N_{\leq}$.

The algorithm $F^N_{\leq}$ may be thought of as incrementally attempting to build a normal form derivation of a statement J, starting from the root and recursively building sub-derivations for the premises. By Lemma 2.4.2, if there is any derivation whatsoever of a statement J, there is one in normal form; the algorithm is guaranteed to recapitulate this derivation and halt in finite time.

2.4.3. Fact: [Curien and Ghelli] $\Gamma \vdash \sigma \leq \tau$ is derivable in $F^N_{\leq}$ iff the algorithm $F^N_{\leq}$ halts and returns *true* when given this statement as input.

Unfortunately, the algorithm is *not* a decision procedure for the subtype relation. Indeed, this relation can be shown to be undecidable (Ghelli, 1995; Pierce, 1994).

3. The $\mathbf{F}_\wedge$ Calculus

We now introduce System $F_\wedge$, an explicitly typed second-order lambda calculus with bounded quantification and intersection types. $F_\wedge$ can roughly be characterized as the union of the concrete syntax and typing rules for the systems $\lambda_\wedge$ and $F_{\leq}$. To achieve a compact and elegant calculus, however, a few small modifications and extensions are needed:

— Since $F_{\leq}$ allows primitive types to be encoded as elements of the pervasive context, we drop the primitive types of $\lambda_{\leq}$ and $\lambda_\wedge$ and the rule SUB-PRIM.

— Since $\top$ and *Top* both function as maximal elements of their respective subtype orderings, we drop *Top* and let $\top$ take over its job. Section 3.3 discusses this design decision in more detail.

— Since $\forall$ behaves like a kind of function space constructor, we add a new law SUB-DIST-IQ, analogous to SUB-DIST-IA, which allows intersections to be distributed over quantifiers on the right-hand side.

— We use the ordinary form of λ-abstraction from $F_{\leq}$ rather than the generalized one introduced by Reynolds for $\lambda_{\wedge}$. The need to derive alternative typings for subphrases under different sets of assumptions is satisfied by a new syntactic form, *for*.

The *for* construct is described in detail in Section 3.1. Section 3.2 then summarizes the concrete syntax, subtyping rules, and typing rules of $F_{\wedge}$. Section 3.3 discusses some design choices.

3.1. *Explicit Alternation: The for Construct*

The notions of type variables and type substitution inherited from $F_{\leq}$ can be used to define an elegant generalization of the alternation inherent in $\lambda_{\wedge}$'s λ-abstractions with multiple type annotations. We extend the concrete syntax of terms with a *for* form

$$e \quad ::= \quad \ldots \quad | \quad \mathit{for}\ \alpha\ \mathit{in}\ \sigma_1..\sigma_n.\ e$$

whose typing rule allows a choice of any of the σ's as a replacement for α in the body:

$$\frac{\Gamma \vdash \{\sigma_i/\alpha\}e \ \in\ \tau_i}{\Gamma \vdash \mathit{for}\ \alpha\ \mathit{in}\ \sigma_1..\sigma_n.\ e \ \in\ \tau_i} \qquad (\textsc{For})$$

This rule, like the generalized arrow introduction rule ARROW-I' of $\lambda_{\wedge}$, can be used together with INTER-I to generate a set of up to n alternative typings for the body and then form their intersection as the type of the whole *for* expression:

```
> double = for A in Int,Real. \x:A. plus x x;
double : Real->Real /\ Int->Int
```

Indeed, $\lambda_{\wedge}$'s λ-abstraction may be reintroduced as a syntactic abbreviation:

$$\lambda x{:}\sigma_1..\sigma_n.\ e \stackrel{\text{def}}{=} \mathit{for}\ \alpha\ \mathit{in}\ \sigma_1..\sigma_n.\ \lambda x{:}\alpha.\ e,$$

where α is fresh.

Besides separating the mechanisms of functional abstraction and alternation, the *for* construct extends the expressive power of the language by providing a name for the "current choice" being made by the type checker:

```
> for A in Int,Real.
>   \\B<A. \f:A->B. \x:A.
>     f (double x);
it : All B<Real. (Real->B)->Real->B /\ All B<Int. (Int->B)->Int->B
```

Indeed, the finer control over alternation allowed by the explicit *for* construct may be used to improve the efficiency of typechecking even for first-order languages with intersections. For example, Forsythe's generalized λ-abstraction allows the definition of polynomial functions like the following:

```
> poly =
>   \w:Int,Real. \x:Int,Real. \y:Int,Real. \z:Int,Real.
>       plus (double x) (plus (plus w y) z);
poly : Real->Real->Real->Real->Real /\ Int->Int->Int->Int->Int
```

But the behavior of the typechecker on such programs is unnecessarily inefficient. It is easy to see, from the types of `plus` and `double`, that, when all the arguments to `poly` have type `Int`, the result type will be `Int`, and that otherwise the result type will be `Real`. So if we choose `Real` for the type of any of the four parameters, we might as well choose `Real` for the others too. We can realize a substantial gain in typechecking efficiency by making this observation explicit with a *for* expression:

```
> poly =
>   for A in Int,Real.
>     \w:A. \x:A. \y:A. \z:A.
>         plus (double x) (plus (plus w y) z);
poly : Real->Real->Real->Real->Real /\ Int->Int->Int->Int->Int
```

The second version of `poly` requires that the body be checked only twice, as compared to sixteen times for the first version.

3.2. *Syntax, Subtyping, and Typing*

We now give a precise definition of the $F_\wedge$ calculus, which forms the main object of study for the remainder of the paper. Since all of its components have already been discussed in detail, we present just the bare facts. These definitions are also summarized in Appendix B for quick reference.

3.2.1. Definition: The set of $F_\wedge$ types is defined by the following grammar:

$$
\begin{array}{lll}
\tau & ::= & \alpha \\
 & | & \tau_1 \to \tau_2 \\
 & | & \forall\alpha{\leq}\tau_1.\ \tau_2 \\
 & | & \bigwedge[\tau_1..\tau_n]
\end{array}
$$

The set of $F_\wedge$ terms is defined by the following grammar:

$$
\begin{array}{lll}
e & ::= & x \\
 & | & \lambda x{:}\tau.\ e \\
 & | & e_1\ e_2 \\
 & | & \Lambda\alpha{\leq}\tau.\ e \\
 & | & e[\tau] \\
 & | & \mathit{for}\ \alpha\ \mathit{in}\ \tau_1..\tau_n.\ e
\end{array}
$$

3.2.2. Definition: The three-place $F_\wedge$ *subtype relation* $\Gamma \vdash \sigma \leq \tau$ is the least relation closed under the following rules:

$$\Gamma \vdash \tau \leq \tau \qquad \text{(Sub-Refl)}$$

$$\frac{\Gamma \vdash \tau_1 \leq \tau_2 \qquad \Gamma \vdash \tau_2 \leq \tau_3}{\Gamma \vdash \tau_1 \leq \tau_3} \qquad \text{(Sub-Trans)}$$

$$\Gamma \vdash \alpha \leq \Gamma(\alpha) \qquad \text{(Sub-TVar)}$$

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \sigma_1{\to}\sigma_2 \leq \tau_1{\to}\tau_2} \qquad \text{(Sub-Arrow)}$$

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma, \alpha{\leq}\tau_1 \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \forall\alpha{\leq}\sigma_1.\ \sigma_2 \leq \forall\alpha{\leq}\tau_1.\ \tau_2} \qquad \text{(SUB-ALL)}$$

$$\frac{\text{for all } i,\ \Gamma \vdash \sigma \leq \tau_i}{\Gamma \vdash \sigma \leq \bigwedge[\tau_1..\tau_n]} \qquad \text{(SUB-INTER-G)}$$

$$\Gamma \vdash \bigwedge[\tau_1..\tau_n] \leq \tau_i \qquad \text{(SUB-INTER-LB)}$$

$$\Gamma \vdash \bigwedge[\sigma{\rightarrow}\tau_1 \mathrel{..} \sigma{\rightarrow}\tau_n] \leq \sigma \rightarrow \bigwedge[\tau_1..\tau_n] \qquad \text{(SUB-DIST-IA)}$$

$$\Gamma \vdash \bigwedge[\forall\alpha{\leq}\sigma.\ \tau_1 \mathrel{..} \forall\alpha{\leq}\sigma.\ \tau_n] \leq \forall\alpha{\leq}\sigma.\ \bigwedge[\tau_1..\tau_n] \qquad \text{(SUB-DIST-IQ)}$$

3.2.3. Definition: The three-place $F_\wedge$ *typing relation* $\Gamma \vdash e \in \tau$ is the least relation closed under the following rules:

$$\Gamma \vdash x \in \Gamma(x) \qquad \text{(VAR)}$$

$$\frac{\Gamma, x{:}\tau_1 \vdash e \in \tau_2}{\Gamma \vdash \lambda x{:}\tau_1.\ e \in \tau_1{\rightarrow}\tau_2} \qquad \text{(ARROW-I)}$$

$$\frac{\Gamma \vdash e_1 \in \tau_1{\rightarrow}\tau_2 \qquad \Gamma \vdash e_2 \in \tau_1}{\Gamma \vdash e_1\ e_2 \in \tau_2} \qquad \text{(ARROW-E)}$$

$$\frac{\Gamma, \alpha{\leq}\tau_1 \vdash e \in \tau_2}{\Gamma \vdash \Lambda\alpha{\leq}\tau_1.\ e \in \forall\alpha{\leq}\tau_1.\ \tau_2} \qquad \text{(ALL-I)}$$

$$\frac{\Gamma \vdash e \in \forall\alpha{\leq}\tau_1.\ \tau_2 \qquad \Gamma \vdash \tau \leq \tau_1}{\Gamma \vdash e[\tau] \in \{\tau/\alpha\}\tau_2} \qquad \text{(ALL-E)}$$

$$\frac{\Gamma \vdash \{\sigma_i/\alpha\}e \in \tau_i}{\Gamma \vdash \mathit{for}\ \alpha\ \mathit{in}\ \sigma_1..\sigma_n.\ e \in \tau_i} \qquad \text{(FOR)}$$

$$\frac{\text{for all } i,\ \Gamma \vdash e \in \tau_i}{\Gamma \vdash e \in \bigwedge[\tau_1..\tau_n]} \qquad \text{(INTER-I)}$$

$$\frac{\Gamma \vdash e \in \tau_1 \qquad \Gamma \vdash \tau_1 \leq \tau_2}{\Gamma \vdash e \in \tau_2} \qquad \text{(SUB)}$$

When necessary to prevent confusion, turnstiles in $F_\wedge$ derivations are written $\overset{\wedge}{\vdash}$.

3.3. *Discussion*

We pause now to discuss some design choices that arise in the formulation of $F_\wedge$ and explore some of its properties.

3.3.1. *Top vs.* $\top$ In forming $F_\wedge$ from $\lambda_\wedge$ and $F_\leq$, we find, pleasantly, that most of their features are quite orthogonal: for a given feature, either it is found already in $\lambda_\leq$ or else it exists in either $\lambda_\wedge$ or $F_\leq$ in a form that interacts smoothly with all the features of the other. The one exception is the maximal types $\top$ and *Top*. In the best case, we might hope that these would coincide in $F_\wedge$, but this, unfortunately, is not the case.

The difference arises from the INTER-I rule of $\lambda_\wedge$, which, in its nullary form, states that any term whatsoever has type $\top$. $F_\leq$ has no such rule: the only way a term e can be assigned type *Top* is by the rules SUB and SUB-TOP, which require that the term already have some type σ with $\sigma \leq \tau$. In other words:

— *Top* is the type of all *well-typed* terms;

— $\top$ is the type of *all* terms.

Order-theoretically, the two types are equivalent (each is a subtype of the other), since each is explicitly axiomatized as a maximal type.

For the sake of conceptual parsimony, we drop *Top* here and retain $\top$, since $\top$ can perform the same jobs as *Top* (in particular, it allows unbounded quantification to be recovered from bounded quantification), while requiring no extra typing or subtyping rules beyond those defining the behavior of general n-ary intersections.

The alternative system with both $\top$ and *Top*, though messier, makes reasonable syntactic sense and does not appear to be much harder to typecheck. More interesting, though, is the system with *Top* instead of $\top$, where intersections are restricted to two or more elements: this language would have much of the practical expressiveness of $F_\wedge$ while avoiding the unfamiliar notion of a "type of all terms, even ill-behaved ones": this system supports the notion of *typechecking failure*, which in $F_\wedge$ is simply identified with $\top$.

3.3.2. *Encoding Primitive Subtyping* As in $F_{\leq}$, both term and type constants are absent from $F_\wedge$, and programs involving them are expressed as terms with free variables whose typing and subtyping behavior are declared in a pervasive context Γ_P. The presence of intersection types allows a greater variety of primitive subtype relations to be encoded in this way than was possible in $F_{\leq}$. For example, the relation in which *Bit* is a subtype of both *Bool* and *Int*, which are both subtypes of *Int*, is represented by the pervasive context

$$\Gamma_P \equiv \mathit{Value}{\leq}\top,\ \mathit{Int}{\leq}\,\mathit{Value},\ \mathit{Bool}{\leq}\,\mathit{Value},\ \mathit{Bit}{\leq}\mathit{Int}{\wedge}\mathit{Bool}.$$

Infinite primitive subtype relations and primitive subtype relations that are not partially ordered, such as

$$\mathit{PolarComplex} \rightleftarrows \mathit{RectComplex},$$

are still not expressible.

3.3.2.1. Definition: A *topological sort* of a finite collection of primitive types P is a bijective mapping $\mathit{index}_P \in P \to \{1\ ..\ |P|\}$ such that $\alpha \leq_P \beta$ implies $\mathit{index}_P(\alpha) \geq \mathit{index}_P(\beta)$.

3.3.2.2. Definition: Let P be a finite collection of primitive types topologically sorted by index_P. Let $\alpha_i \equiv \mathit{index}_P^{-1}(i)$. Then P is *encoded* by the following $F_\wedge$ context:

$$\Gamma_P = \ldots,\ \alpha_i\ \leq\ \bigwedge[\gamma \in P \mid \alpha_i {\leq_P} \gamma],\ \ldots$$

Note that every finite partial order can be topologically sorted.

3.3.2.3. Lemma: Let Γ be an $\lambda_\wedge$ context and assume that the primitive subtype relation $\leq_P$ can be topologically sorted by some function index_P. If $\Gamma \vdash \sigma \leq \tau$ is derivable in $\lambda_{\leq}$, then $\Gamma_P, \Gamma \vdash \sigma \leq \tau$ is derivable in $F_\wedge$.

Proof: By induction on the structure of the given derivation. All of the $\lambda_\wedge$ rules translate directly into $F_\wedge$ rules, except for Sub-Prim; an instance of this rule with conclusion $\Gamma \vdash \alpha \leq \beta$ is translated into the following $F_\wedge$ derivation:

$$\dfrac{\dfrac{\scriptstyle\text{(SUB-TVAR)}}{\Gamma_P, \Gamma \ \vdash^{\wedge} \ \alpha \ \leq \ \bigwedge[\gamma \in P \mid \alpha {\leq_P} \gamma]} \qquad \dfrac{\scriptstyle\text{(SUB-INTER-LB)}}{\Gamma_P, \Gamma \ \vdash^{\wedge} \ \bigwedge[\gamma \in P \mid \alpha {\leq_P} \gamma] \ \leq \ \beta}}{\Gamma_P, \Gamma \ \vdash^{\wedge} \ \alpha \ \leq \ \beta.} \ \scriptstyle\text{(SUB-TRANS)}$$

□

3.3.2.4. Fact: Let Γ by a $\lambda_\wedge$ context and assume that the primitive subtype relation $\leq_P$ can be topologically sorted by some function $index_P$. Then $\Gamma \vdash^{\lambda\wedge} e \in \tau$ only if $\Gamma_P, \Gamma \vdash^{\wedge} e \in \tau$.

(In fact, the derivation-normalization results of Section 5 can be used to show the converse, so $\Gamma \vdash^{\lambda\wedge} e \in \tau$ iff $\Gamma_P, \Gamma \vdash^{\wedge} e \in \tau$.)

4. Examples

Intersection types allow very refined types to be assigned to expressions — much more refined than is possible in conventional polymorphic languages. Instead of a single uniform description, each expression may be assigned any finite collection of descriptions, each capturing some aspect of its behavior. Since we are working in an explicitly typed calculus, this requires effort from the programmer in the form of type assumptions or annotations; in general, as more effort is expended, better typings are obtained.

Many functional languages provide a primitive type `Bool` with two elements, `true` and `false`. Here we can introduce two subtypes of `Bool`, called `True` and `False`, and give more exact types for the constants `true` and `false` in terms of these refinements:

```
> Bool < T,
> True < Bool,
> False < Bool;
```

```
> true : True,
> false : False;
```

The polymorphic `if` primitive can be given a more refined type than usual: if we know whether the value of the test lies in the type `True` or the type `False`, we can tell in advance which of the branches will be chosen. An optimizing compiler might use this information to generate more efficient code.

```
> if : All A.    (True -> A -> T -> A)
>             /\ (False -> T -> A -> A)
>             /\ (Bool -> A -> A -> A);
```

(The third typing is needed here because $F_\wedge$'s types cannot express the idea that every element of `Bool` is an element of either `True` or `False`. This shortcoming, while not serious in practice, has motivated the investigation of a dual notion of *union types* (Barbanera & Dezani-Ciancaglini, 1991; Hayashi, 1991; Pierce, 1991b; Cardone *et al.*, 1994; Barbanera *et al.*, 1995).) The refinement in the types of `true`, `false`, and `if` can now be exploited in typing new functions:

```
> or =
>    \x:True,False,Bool. \y:True,False,Bool.
>       for R in True,False,Bool.
>          if [R] x true y;
or : Bool->(Bool->Bool/\True->True)
  /\ False->False->False
  /\ True->Bool->True
```

In fact, we can carry out the same construction in the *pure* $F_\wedge$ calculus with no assumptions about predefined types or constants, using a generalization of the familiar Church encoding of booleans in the polymorphic λ-calculus (Böhm & Berarducci, 1985).

```
> True  == All B. All TT<B. All FF<B. TT -> T  -> TT,
> False == All B. All TT<B. All FF<B. T  -> FF -> FF,
> Bool  == All B. All TT<B. All FF<B. TT -> FF -> B;
```

```
> true = \\B. \\TT<B. \\FF<B. \x:TT. \y:T. x,
> false = \\B. \\TT<B. \\FF<B. \x:T. \y:FF. y;
true : True
false : False
```

```
> or = for M in True,False,Bool.
>        for N in True,False,Bool.
>          \m:M. \n:N.
>             m [Bool] [True] [N] true n;
or : Bool->Bool->Bool
  /\ False->(False->False/\True->True)
  /\ True->Bool->True
```

Similar refinements of the encodings of natural numbers, lists, etc. can also be given (Pierce, 1991a).

5. Typechecking

This section develops the proof theory of the $F_\wedge$ calculus, leading up to the definition and correctness proof of an algorithm for synthesizing minimal types of $F_\wedge$ terms. The major results we establish are as follows:

— We give an alternative formulation of the subtype relation in terms of "canonical types," where intersections appear only on the left of arrows and quantifiers. This formulation is equivalent to the original, in the sense that there is some canonical type in the equivalence class of each ordinary type. More formally, the "flattening" map from ordinary to canonical types both preserves and reflects derivability of subtyping statements.

— A proof-normalization argument based on the one used by Curien and Ghelli (1992) shows that every derivable canonical subtyping statement has a "normal form" derivation with a particular, restricted shape.

— The existence of normal-form canonical derivations is used to prove the semi-completeness of a syntax-directed (semi-) algorithm for checking the subtype relation.

— The soundness and completeness of a syntax-directed type synthesis algorithm for $F_\wedge$ terms is established by showing that there exist finite bases for the collections of arrow types and quantified types lying above a given type. This argument also shows that the subroutine for checking the subtype relation is the only source of possible nontermination in the typechecking algorithm — i.e. the typechecking algorithm is a decision procedure, given an oracle for subtyping.

— The shapes of the typing derivations discovered by this algorithm are used to prove that $F_\wedge$ is a conservative extension of the first-order intersection calculus $\lambda_\wedge$. (Because of the different behavior of *Top* and $\top$, however, $F_\leq$ cannot similarly be embedded in $F_\wedge$.)

5.1. *Basic Properties*

We begin by establishing some basic proof-theoretic properties of the subtype relation $\Gamma \vdash \sigma \leq \tau$ and the typing relation $\Gamma \vdash e \in \tau$. First, we state some useful derived rules of inference.

5.1.1. Lemma:

$$\Gamma \vdash \bigwedge[\sigma{\to}\tau_1 \mathinner{..} \sigma{\to}\tau_n] \sim \sigma \to \bigwedge[\tau_1 \mathinner{..} \tau_n] \qquad \text{(D-DIST-IA)}$$

$$\Gamma \vdash \bigwedge[\forall\alpha{\leq}\sigma.\ \tau_1 \mathinner{..} \forall\alpha{\leq}\sigma.\ \tau_n] \sim \forall\alpha{\leq}\sigma.\ \bigwedge[\tau_1 \mathinner{..} \tau_n] \qquad \text{(D-DIST-IQ)}$$

$$\Gamma \vdash \bigwedge[\bigwedge T_1 \mathinner{..} \bigwedge T_n] \sim \bigwedge(T_1 * \cdots * T_n) \qquad \text{(D-ABSORB)}$$

$$\frac{T \text{ and } T' \text{ enumerate the same finite set}}{\Gamma \vdash \bigwedge T \sim \bigwedge T'} \qquad \text{(D-REINDEX)}$$

$$\frac{\text{for all } \tau_j \text{ there is some } \sigma_i \text{ such that } \Gamma \vdash \sigma_i \leq \tau_j}{\Gamma \vdash \bigwedge[\sigma_1 \mathinner{..} \sigma_m] \leq \bigwedge[\tau_1 \mathinner{..} \tau_n]} \qquad \text{(D-ALL-SOME)}$$

Proof: Straightforward. □

5.1.2. Lemma: [Permutation] If Γ' is a permutation of Γ and both Γ and Γ' are closed, then

1 $\Gamma \vdash \sigma \leq \tau$ iff $\Gamma' \vdash \sigma \leq \tau$

2 $\Gamma \vdash e \in \tau$ iff $\Gamma' \vdash e \in \tau$.

Proof: By induction on derivations. Note, furthermore, that the antecedent and consequent derivations are isomorphic. □

Lemma 5.1.2 justifies a notational simplification: two closed contexts Γ and Γ' that differ only in the order of their bindings will be considered *identical* from now on.

5.1.3. Lemma: [Weakening] If $\Gamma_1,\ \Gamma_2$ is closed and $\Gamma_1 \vdash \sigma \leq \tau$, then $\Gamma_1,\ \Gamma_2 \vdash \sigma \leq \tau$.

Proof: By induction on the structure of a derivation of $\Gamma_1 \vdash \sigma \leq \tau$. At each stage of the induction, we proceed by a case analysis on the rule used in the last step of the derivation. The most interesting case is Sub-All:

Case Sub-All*:* $\sigma \equiv \forall\alpha{\leq}\sigma_1.\ \sigma_2 \qquad \tau \equiv \forall\alpha{\leq}\tau_1.\ \tau_2$

By assumption, $\Gamma_1 \vdash \tau_1 \leq \sigma_1$ and $\Gamma_1,\ \alpha{\leq}\tau_1 \vdash \sigma_2 \leq \tau_2$. We may also assume that $\alpha \notin dom(\Gamma_1,\ \Gamma_2)$. Then $\Gamma_1,\ \Gamma_2,\ \alpha{\leq}\tau_1$ is closed, and, by the induction hypothesis, $\Gamma_1,\ \Gamma_2 \vdash \tau_1 \leq \sigma_1$ and $\Gamma_1,\ \Gamma_2,\ \alpha{\leq}\tau_1 \vdash \sigma_2 \leq \tau_2$. By Sub-All, $\Gamma_1,\ \Gamma_2 \vdash \forall\alpha{\leq}\sigma_1.\ \sigma_2 \leq \forall\alpha{\leq}\tau_1.\ \tau_2$. □

A different kind of weakening lemma will also be needed. Rather than adding a new variable to the context, this one states that a derivable subtyping statement remains derivable when the bounds of some of the existing type variables are replaced by "narrower" bounds.

5.1.4. Lemma: [Narrowing] Let Γ and Γ' be closed contexts such that, for each $\alpha_i \in dom(\Gamma)$, $\Gamma' \vdash \Gamma'(\alpha_i) \leq \Gamma(\alpha_i)$. Then $\Gamma \vdash \sigma \leq \tau$ implies $\Gamma' \vdash \sigma \leq \tau$.

Proof: See Appendix A. □

Using narrowing, we can show that the equivalence relation induced by the subtype relation is a congruence:

5.1.5. Lemma:

$$\frac{\Gamma \vdash \tau_1 \sim \tau_1' \qquad \Gamma \vdash \tau_2 \sim \tau_2'}{\Gamma \vdash \tau_1{\rightarrow}\tau_2 \sim \tau_1'{\rightarrow}\tau_2'} \tag{D-Cong-Arrow}$$

$$\frac{\Gamma \vdash \tau_1 \sim \tau_1' \qquad \Gamma,\ \alpha{\leq}\tau_1' \vdash \tau_2 \sim \tau_2'}{\Gamma \vdash \forall\alpha{\leq}\tau_1.\ \tau_2 \sim \forall\alpha{\leq}\tau_1'.\ \tau_2'} \tag{D-Cong-All}$$

$$\frac{\text{for all } i,\ \Gamma \vdash \tau_i \sim \tau_i'}{\Gamma \vdash \bigwedge[\tau_1..\tau_n] \sim \bigwedge[\tau_1'..\tau_n']} \tag{D-Cong-Inter}$$

Proof: Easy except for D-Cong-All, which requires narrowing. □

5.1.6. Lemma: [Subtyping Substitution] If $\Gamma_1,\ \alpha{\leq}\psi,\ \Gamma_2 \vdash \sigma \leq \tau$ and $\Gamma_1 \vdash \phi \leq \psi$, then $\Gamma_1,\ \{\phi/\alpha\}\Gamma_2 \vdash \{\phi/\alpha\}\sigma \leq \{\phi/\alpha\}\tau$.

Proof: See Appendix A. □

We would also like to show that whenever a variable α or x in $dom(\Gamma)$ is unused in a statement $\Gamma \vdash \sigma \leq \tau$ or $\Gamma \vdash e \in \tau$, we may drop it from Γ without affecting derivability. Because the rule Sub-Trans allows subtyping derivations that can contain internal uses of arbitrary types, we cannot prove this property for type variables with the machinery we have developed so far (it will be an easy corollary of Theorem 5.2.6.10). But for term variables it is straightforward:

5.1.7. Lemma: [Term variable strengthening]

1 If $\Gamma,\ x{:}\theta \vdash \sigma \leq \tau$, then $\Gamma \vdash \sigma \leq \tau$.

2 If $\Gamma,\ x{:}\theta \vdash e \in \tau$ and $x \notin FV(e)$, then $\Gamma \vdash e \in \tau$.

Proof: By induction on derivations. □

5.1.8. Lemma: [Term substitution] If $\Gamma,\ x{:}\sigma \vdash f \in \tau$ and $\Gamma \vdash v \in \sigma$, then $\Gamma \vdash \{v/x\}f \in \tau$.

Proof: By induction on a derivation of $\Gamma,\ x{:}\sigma \vdash f \in \tau$, using weakening for the ARROW-I and ALL-I cases and term strengthening for the ALL-E and SUB cases. □

5.1.9. Lemma: [Type substitution in terms] If $\Gamma_1,\ \alpha{\le}\sigma,\ \Gamma_2 \vdash f \in \tau$, then $\Gamma_1,\ \{\sigma/\alpha\}\Gamma_2 \vdash \{\sigma/\alpha\}f \in \{\sigma/\alpha\}\tau$.

Proof: By induction on a derivation of $\Gamma_1,\ \alpha{\le}\sigma,\ \Gamma_2 \vdash f \in \tau$, using the subtyping substitution lemma for the ALL-E and SUB cases. □

5.2. *Subtyping*

In this section, we give a straightforward semi-decision procedure for the $F_\wedge$ subtype relation. This relation can be shown to be undecidable (Pierce, 1994), so a semi-decision procedure is the best we can hope for; however, the same algorithm is a decision procedure for some useful fragments of $F_\wedge$ (see Section 7).

We present the algorithm as a collection of syntax-directed rules and then show that the relation defined by these rules coincides with $F_\wedge$ subtyping. It is technically convenient to make this argument using an intermediate representation called *canonical types*, roughly analogous to conjunctive-normal-form formulas in logic:

— We identify the set of canonical types (Section 5.2.1) and define a canonical subtyping relation (marked $\overset{\flat}{\vdash}$) over this set (Section 5.2.2).

— We then show (Section 5.2.3) that derivations of canonical subtyping statements can be transformed into *normal form* derivations of a certain restricted shape (Section 5.2.4).

— We give a *flattening transformation* $\flat$ mapping $F_\wedge$ types into canonical types (Section 5.2.5) and show that $\Gamma \vdash \sigma \le \tau$ iff $\Gamma^\flat \overset{\flat}{\vdash} \sigma^\flat \le \tau^\flat$.

— Finally, we define a syntax-directed subtyping relation on ordinary types (marked $\overset{\text{!}}{\vdash}$) and show (Section 5.2.6) that it coincides with canonical subtyping after flattening: $\Gamma^\flat \overset{\flat}{\vdash} \sigma^\flat \le \tau^\flat$ iff $\Gamma \overset{\text{!}}{\vdash} \sigma \le \tau$.

5.2.1. *Canonical Types* We might naively hope to develop an algorithm for checking $F_\wedge$ subtyping simply by extending the $F_\le$ subtyping algorithm $F_\le^N$ to include a case for intersections. Keeping cases *1* to *5* as before, the complete algorithm would then be:

check$(\Gamma \vdash \sigma \le \tau)$ =
- *1.* *if* $\tau \equiv Top$
  - *then true*
- *2.* *else if* $\sigma \equiv \sigma_1 {\to} \sigma_2$ *and* $\tau \equiv \tau_1 {\to} \tau_2$
  - *then* *check*$(\Gamma \vdash \tau_1 \le \sigma_1)$
  - *and check*$(\Gamma \vdash \sigma_2 \le \tau_2)$
- *3.* *else if* $\sigma \equiv \forall\alpha{\le}\sigma_1.\ \sigma_2$ *and* $\tau \equiv \forall\alpha{\le}\tau_1.\ \tau_2$

then $\quad check(\Gamma \vdash \tau_1 \leq \sigma_1)$
and $check(\Gamma,\, \alpha{\leq}\tau_1 \vdash \sigma_2 \leq \tau_2)$

4. *else if* $\sigma \equiv \alpha$ *and* $\tau \equiv \alpha$
then true

5. *else if* $\sigma \equiv \alpha$
then $check(\Gamma \vdash \Gamma(\alpha) \leq \tau)$

6. *else if* $\sigma \equiv \bigwedge[\sigma_1..\sigma_m]$ *and* $\tau \equiv \bigwedge[\tau_1..\tau_n]$
then for each τ_i
choose some σ_j
such that $check(\Gamma \vdash \sigma_j \leq \tau_i)$

n. *else*
false.

That is, to check whether $\bigwedge[\sigma_1..\sigma_m]$ is a subtype of $\bigwedge[\tau_1..\tau_n]$, we check that for each τ_i there is some σ_j such that $\sigma_j \leq \tau_i$. The selection of the σ_j is expressed here as a nondeterministic choice; in practice, this is implemented using backtracking.

But case 6, though clearly sound (cf. Lemma 5.1.1), is not complete, since there are many cases where a meet lies above or below a type that does not have the form of a meet. For example,

$$\alpha{\leq}\top \vdash \alpha \leq \bigwedge[\alpha]$$
$$\alpha{\leq}\top \vdash \bigwedge[\alpha] \leq \alpha.$$

These two cases can be handled by splitting the proposed rule into two,

6a. *else if* $\tau \equiv \bigwedge[\tau_1..\tau_n]$
then for each τ_i
$check(\Gamma \vdash \sigma \leq \tau_i)$

6b. *else if* $\sigma \equiv \bigwedge[\sigma_1..\sigma_m]$
then choose some σ_j
such that $check(\Gamma \vdash \sigma_j \leq \tau)$,

but we must be careful to apply *6a* before *6b* in order to respond correctly to the input

$$\alpha{\leq}\top,\, \beta{\leq}\top \;\vdash\; \bigwedge[\alpha, \beta] \;\leq\; \bigwedge[\alpha, \beta].$$

Unfortunately, the real problem is more subtle than this: in the presence of the distributivity axioms Sub-Dist-IA and Sub-Dist-IQ, steps *2* and *3* of the $F^N_{\leq}$ algorithm are also incomplete. For example, the judgement

$$\alpha{\leq}\top,\, \beta{\leq}\top \;\vdash\; \alpha{\rightarrow}\alpha \;\leq\; \beta{\rightarrow}\top,$$

is derivable (using Sub-Dist-IA and Sub-Trans, with intermediate type $\top$), although

$$\alpha{\leq}\top,\, \beta{\leq}\top \;\not\vdash\; \beta \;\leq\; \alpha.$$

To get matters under control, we need to deal with the distributivity laws before doing anything else. We take the inverses of these laws (i.e., the other half of the equivalences D-Dist-IA and D-Dist-IQ given in 5.1.1) as rewrite rules,

$$\sigma \rightarrow \bigwedge[\tau_1..\tau_n] \quad\longrightarrow\quad \bigwedge[\sigma{\rightarrow}\tau_1 \,..\, \sigma{\rightarrow}\tau_n]$$
$$\forall\alpha{\leq}\sigma.\ \bigwedge[\tau_1..\tau_n] \quad\longrightarrow\quad \bigwedge[\forall\alpha{\leq}\sigma.\ \tau_1 \,..\, \forall\alpha{\leq}\sigma.\ \tau_n],$$

and apply them to a given $F_\wedge$ type as many times as possible to obtain an equivalent type with no intersections on the right-hand sides of arrows or quantifiers. For example,

$$\bigwedge[\alpha_1,\ \alpha_2] \ \rightarrow\ \bigwedge[\alpha_3,\ (\forall\beta{\leq}\alpha_4.\ \bigwedge[\alpha_5,\ \alpha_6])]$$

becomes

$$\begin{array}{l}\bigwedge[\bigwedge[\alpha_1,\ \alpha_2] \ \rightarrow\ \alpha_3, \\ \quad \bigwedge[\bigwedge[\alpha_1,\ \alpha_2] \ \rightarrow\ \forall\beta{\leq}\alpha_4.\ \alpha_5, \\ \qquad \bigwedge[\alpha_1,\ \alpha_2] \ \rightarrow\ \forall\beta{\leq}\alpha_4.\ \alpha_6]].\end{array}$$

Since this type contains no meets on the right-hand sides of arrows or quantifiers, there is no way to use it (or any subphrase) in an instance of either of the distributivity rules.

Once the distributivity rules are eliminated, there remain just four subtyping rules in Definition 3.2.2 that can be used to prove subtyping statements involving meets: Sub-Refl, Sub-Trans, Sub-Inter-G, and Sub-Inter-LB. The reflexivity and transitivity rules can be eliminated by a normalization argument that shows how to transform any derivation into one that applies transitivity and reflexivity only to variables (cf. Sections 5.2.3 and 5.2.4). This leaves just Sub-Inter-G and Sub-Inter-LB, which are accurately captured by our rules *6a* and *6b*.

In the technical development, it is convenient to work with types in an even more restricted form, in which every type has a single $\bigwedge$ as its outermost constructor and as the outermost constructor on the left-hand sides of arrows and quantifiers, and where no immediate component of a $\bigwedge$ is another $\bigwedge$. In this form, our example becomes

$$\begin{array}{l}\bigwedge[\bigwedge[\alpha_1,\ \alpha_2] \ \rightarrow\ \alpha_3, \\ \quad \bigwedge[\alpha_1,\ \alpha_2] \ \rightarrow\ \forall\beta{\leq}\bigwedge[\alpha_4].\ \alpha_5, \\ \quad \bigwedge[\alpha_1,\ \alpha_2] \ \rightarrow\ \forall\beta{\leq}\bigwedge[\alpha_4].\ \alpha_6].\end{array}$$

This transformation effectively separates the set of types into two syntactic classes: those whose outermost constructor is $\bigwedge$ (called "composite canonical types") and those whose outermost constructor is $\rightarrow$, $\forall$, or a variable (called "individual canonical types"). This separation is useful because our proposed rule *6* now captures *all* of the valid subtyping statements involving pairs of composite canonical types, while the original rules *2–5* are valid and complete for pairs of individual canonical types.

5.2.1.1. Definition: The sets of *composite canonical types* K and *individual canonical types* κ are defined by the following grammar:

$$\begin{array}{lll} K & ::= & \bigwedge[\kappa_1..\kappa_n] \\ \kappa & ::= & \alpha \quad | \quad K{\rightarrow}\kappa \quad | \quad \forall\alpha{\leq}K.\ \kappa \end{array}$$

The metavariables K and I range over composite canonical types; κ and ι range over individual canonical types; k and i range over both sorts of canonical types. It will often be convenient to treat a composite canonical type $\bigwedge[\kappa_1..\kappa_n]$ as a finite set whose elements are the individual canonical types κ_1 through κ_n. The following conventions support this point of view:

$$\begin{array}{lll} \iota \in K & \stackrel{\text{def}}{=} & K \equiv \bigwedge[\kappa_1..\kappa_n] \text{ and } \iota{\equiv}\kappa_i \text{ for some } i \\ \bigwedge[F(\iota) \mid \iota \in K] & \stackrel{\text{def}}{=} & \bigwedge[F(\kappa_1)\ ..\ F(\kappa_n)], \text{ where } K \equiv \bigwedge[\kappa_1..\kappa_n] \\ K \cup I & \stackrel{\text{def}}{=} & \bigwedge[\kappa_1\ ..\ \kappa_m,\ \iota_1\ ..\ \iota_n], \text{ where } K \equiv \bigwedge[\kappa_1..\kappa_m] \text{ and } I \equiv \bigwedge[\iota_1..\iota_n]. \end{array}$$

Our canonical types are not as sparse as we might make them. For example, we might imagine a notion of "fully canonical types" such that each $\sim$-equivalence class of types contains exactly one fully canonical type. But for present purposes, the canonical types we have defined are refined enough.

The idea of canonical types comes from a proof by Reynolds [personal communication, 1988] of the soundness and completeness of a decision procedure for the subtype relation of Forsythe (Reynolds, 1988). Related formulations of intersection types are studied in (Coppo & Dezani-Ciancaglini, 1980; Coppo *et al.*, 1980; Coppo *et al.*, 1981; Ronchi della Rocca & Venneri, 1984; van Bakel, 1991; van Bakel, 1992).

5.2.2. *Canonical Subtyping* We now define the canonical subtype relation formally.

5.2.2.1. Definition: A *canonical context* Δ is a context whose range contains only composite canonical types.

5.2.2.2. Definition: The subtype relation on canonical types is defined as follows:

$$\frac{\forall i.\ \exists j.\quad \Delta \vdash \kappa_j \leq \iota_i}{\Delta \vdash \bigwedge[\kappa_1..\kappa_m] \leq \bigwedge[\iota_1..\iota_n]} \qquad \text{(CSub-AE)}$$

$$\Delta \vdash k \leq k \qquad \text{(CSub-Refl)}$$

$$\frac{\Delta \vdash k_1 \leq k_2 \qquad \Delta \vdash k_2 \leq k_3}{\Delta \vdash k_1 \leq k_3} \qquad \text{(CSub-Trans)}$$

$$\frac{\Delta \vdash \Delta(\alpha) \leq \bigwedge[\iota]}{\Delta \vdash \alpha \leq \iota} \qquad \text{(CSub-TVar)}$$

$$\frac{\Delta \vdash I_1 \leq K_1 \qquad \Delta \vdash \kappa_2 \leq \iota_2}{\Delta \vdash K_1 \rightarrow \kappa_2 \leq I_1 \rightarrow \iota_2} \qquad \text{(CSub-Arrow)}$$

$$\frac{\Delta \vdash I_1 \leq K_1 \qquad \Delta, \alpha{\leq}I_1 \vdash \kappa_2 \leq \iota_2}{\Delta \vdash \forall\alpha{\leq}K_1.\ \kappa_2 \leq \forall\alpha{\leq}I_1.\ \iota_2} \qquad \text{(CSub-All)}$$

Individual and composite canonical types cannot be mixed in canonical subtyping statements: we have statements of the form $\Delta \vdash K \leq I$ and $\Delta \vdash \kappa \leq \iota$, but never $\Delta \vdash \kappa \leq I$ or $\Delta \vdash K \leq \iota$. In particular, in each instance of CSub-Trans, either k_1, k_2, and k_3 are all canonical types or they are all composite canonical types. Also, anticipating the requirements of the normal-form derivations to be defined in the following section, we have slightly generalized the type variable rule, in effect embedding an instance of CSub-Trans in each instance of CSub-TVar. (Alternatively, we could have embedded an instance of CSub-Trans *and* an instance of CSub-AE in the hypotheses of CSub-TVar, allowing $\Delta \vdash \alpha \leq \iota$ to be derived whenever $\Delta \vdash \kappa_i \leq \iota$ for every $\kappa_i \in \Delta(\alpha)$.) The turnstile symbol is sometimes decorated $\overset{\flat}{\vdash}$ to distinguish canonical subtyping derivations from derivations in other calculi.

The metavariables C and D range over derivations of subtyping statements between composite canonical types; ξ and δ range over derivations of subtyping statements between individual canonical types; c and d (and sometimes e and f) range over both sorts of derivations.

As for ordinary subtyping we adopt the convention that closed canonical contexts differing only in the ordering of their bindings are regarded as identical.

5.2.3. *Normalization of Canonical Subtyping Derivations* To construct an algorithm for checking the canonical subtype relation, we need a notion of *normal form* and an effective procedure for transforming arbitrary derivations into this form. The main task of the normalization procedure is to push instances of CSub-Trans toward the leaves of the derivation until they eventually disappear into instances of the CSub-TVar rule. For example, if $\Delta \equiv \alpha_1{\leq}\top,\ \alpha_2{\leq}\bigwedge[\alpha_1],\ \alpha_3{\leq}\bigwedge[\alpha_2]$, then the derivation

$$\frac{\Delta \vdash \alpha_3 \leq \alpha_2 \qquad \Delta \vdash \alpha_2 \leq \alpha_1}{\Delta \vdash \alpha_3 \leq \alpha_1}\ \text{(CSUB-TRANS)}$$

becomes the normal-form derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Delta \vdash \alpha_1 \leq \alpha_1}\ \text{(CSUB-REFL)}}{\Delta \vdash \bigwedge[\alpha_1] \leq \bigwedge[\alpha_1]}\ \text{(CSUB-AE)}}{\Delta \vdash \alpha_2 \leq \alpha_1}\ \text{(CSUB-TVAR)}}{\Delta \vdash \bigwedge[\alpha_2] \leq \bigwedge[\alpha_1]}\ \text{(CSUB-AE)}}{\Delta \vdash \alpha_3 \leq \alpha_1.}\ \text{(CSUB-TVAR)}$$

A subtyping derivation ending with the rule CSub-Trans is called a *compound derivation* or *cut*. The *cut type* of a compound derivation

$$\frac{\Delta \vdash k_1 \leq k_2 \qquad \Delta \vdash k_2 \leq k_3}{c :: k_1 \leq k_3,}\ \text{(CSSUB-TRANS)}$$

is k_2. A cut whose conclusion involves individual canonical types is called an *individual cut*; a cut whose conclusion involves composite canonical types is called a *composite cut*. The *cut size* of c is $size(k_2)$ (the number of symbols in k_2). A cut c is an *innermost* cut of size m if no proper subderivation of c is a cut of size m.

The proof transformations used to normalize canonical subtyping derivations rely on analogues of the weakening and narrowing lemmas from Section 5.1. We restate them explicitly here because it is important to keep track of the sizes of the cuts they may create.

5.2.3.1. Lemma: [Weakening for canonical subtyping] If $c :: \Delta \vdash k_1 \leq k_2$ and $\alpha \notin dom(\Delta)$, then there is a derivation $c' :: \Delta,\ \alpha{\leq}K \vdash k_1 \leq k_2$ containing cuts of exactly the same sizes as c.

Proof: Similar to 5.1.3. □

5.2.3.2. Lemma: [Narrowing for canonical subtyping] Suppose $c :: \Delta,\ \alpha{\leq}I \vdash k \leq k'$ and $D :: \Delta \vdash K \leq I$. Then there is a derivation $c' :: \Delta,\ \alpha{\leq}K \vdash k \leq k'$ such that the cuts in c' are just those in c with (possibly) some new cuts of the size of I.

Proof: Similar to 5.1.4. Notice that the new cuts created during this operation all have cut-type I. □

5.2.3.3. Definition: We say that a derivation c is in *composite normal form* if every composite subderivation of c has the form of a single instance of rule CSub-AE.

5.2.3.4. Lemma: [Normalization of composite subderivations] If $c :: \Delta \vdash k_1 \leq k_2$ is a derivation all of whose cuts have size at most m, then there is a derivation $c' :: \Delta \vdash k_1 \leq k_2$ in composite normal form whose cuts also have size at most m.

Proof: By induction on the size of c, with a case analysis on the last rule used. When the conclusion of c is a statement about individual canonical types, the argument goes by straightforward induction. The cases where the conclusion of c is a statement about composite canonical types are more interesting.

Case CSub-AE*:*

All the immediate subderivations of c end with statements about individual canonical types. By the induction hypothesis, these can be transformed into composite-normal form, from which the result follows by a final application of CSub-AE.

Case CSub-Refl (composite)*:* $k_1 = k_2 = \bigwedge[\kappa_1..\kappa_m]$

By CSub-Refl, we have $\Delta \vdash \kappa_i \leq \kappa_i$ for each i. So by CSub-AE, $\Delta \vdash \bigwedge[\kappa_1..\kappa_m] \leq \bigwedge[\kappa_1..\kappa_m]$. This derivation ends with CSub-AE and contains no cuts.

Case CSub-Trans (composite)*:* $\Delta \vdash \bigwedge[\kappa_1..\kappa_m] \leq \bigwedge[\iota_1..\iota_o]$
$\Delta \vdash \bigwedge[\iota_1..\iota_o] \leq \bigwedge[\lambda_1..\lambda_n]$

By the induction hypothesis, there are composite-normal derivations $C_1 :: \Delta \vdash \bigwedge[\kappa_1..\kappa_m] \leq \bigwedge[\iota_1..\iota_o]$ and $C_2 :: \Delta \vdash \bigwedge[\iota_1..\iota_o] \leq \bigwedge[\lambda_1..\lambda_n]$, both ending with CSub-AE and containing only cuts of size at most m:

$$\dfrac{\dfrac{\xi_i :: \Delta \vdash \kappa_{f(i)} \leq \iota_i}{C_1 :: \Delta \vdash \bigwedge[\kappa_1..\kappa_m] \leq \bigwedge[\iota_1..\iota_o]} \qquad \dfrac{\delta_j :: \Delta \vdash \iota_{g(j)} \leq \lambda_j}{C_2 :: \Delta \vdash \bigwedge[\iota_1..\iota_o] \leq \bigwedge[\lambda_1..\lambda_n]}}{\Delta \vdash \bigwedge[\kappa_1..\kappa_m] \leq \bigwedge[\lambda_1..\lambda_n]} \text{ (CSUB-TRANS)}$$

for some $f \in \{1..o\}\rightarrow\{1..m\}$ and $g \in \{1..n\}\rightarrow\{1..o\}$. Finish by constructing the derivation

$$\dfrac{\dfrac{\xi_{g(j)} :: \Delta \vdash \kappa_{f(g(j))} \leq \iota_{g(j)} \qquad \delta_j :: \Delta \vdash \iota_{g(j)} \leq \lambda_j}{\Delta \vdash \kappa_{f(g(j))} \leq \lambda_j} \text{ (CSUB-TRANS)}}{c' :: \Delta \vdash \bigwedge[\kappa_1..\kappa_m] \leq \bigwedge[\lambda_1..\lambda_n]} \text{ (CSUB-AE)}$$

and noting that all cuts in c' have size smaller than $size(\bigwedge[\iota_1..\iota_o])$, since the cuts we have just added all have cut-types that are proper subphrases of $\bigwedge[\iota_1..\iota_o]$. □

5.2.3.5. Lemma: [Elimination of a maximal individual cut] Suppose we are given a composite-normal derivation c ending with a cut of the form

$$\dfrac{d :: \Delta \vdash k_1 \leq k_2 \qquad e :: \Delta \vdash k_2 \leq k_3}{\Delta \vdash k_1 \leq k_3}$$

where the size of the final cut is m and where d and e contain only cuts of size strictly less than m. Then there is a derivation $c' :: \Delta \vdash k_1 \leq k_3$ containing only cuts of size strictly less than m.

Proof: By induction on the total size of d and e, with a case analysis on their final rules.

Case Any/CSub-Refl*:* $d :: k_1 \leq k_3 \qquad e :: k_3 \leq k_3$

Immediate: d satisfies the requirements.

Case CSub-Refl/Any*:* $d :: k_1 \leq k_1 \qquad e :: k_1 \leq k_3$

Immediate: e satisfies the requirements.

Case Any/CSub-Trans*:*

We are given:

$$\frac{e_1 :: \Delta \vdash k_2 \leq l \qquad e_2 :: \Delta \vdash l \leq k_3}{e :: \Delta \vdash k_2 \leq k_3}$$

First, note that all the types here are individual canonical types, since there are no composite cuts in a composite-normal derivation. Now, construct a derivation

$$\frac{d :: \Delta \vdash k_1 \leq k_2 \qquad e_1 :: \Delta \vdash k_2 \leq l}{f :: \Delta \vdash k_1 \leq l} \text{ (CSUB-TRANS)}$$

with cut-size m. Since f is smaller than c, the induction hypothesis applies to give a new derivation $f' :: \Delta \vdash k_1 \leq l$ using only cuts strictly smaller than m. Now combine f' and e_2 using CSub-Trans to form $c' :: \Delta \vdash k_1 \leq l \leq k_3$, and note that this final cut has size less than m by assumption.

Case CSub-Trans/Any*:*

Similar.

Case CSub-AE/CSub-AE*:*

Cannot occur, by the assumption that c is in composite-normal form.

Case CSub-TVar/Any*:*

We are given

$$\frac{\dfrac{\xi_i :: \Delta \vdash \kappa_i \leq k_2}{\Delta \vdash \bigwedge[\kappa_1..\kappa_n] \leq \bigwedge[k_2]} \text{ (CSUB-AE)}}{d :: \Delta \vdash \alpha \leq k_2} \text{ (CSUB-TVAR)}$$

where $\Delta(\alpha) = \bigwedge[\kappa_1..\kappa_n]$. For each i, form the derivation

$$\frac{\xi_i :: \Delta \vdash \kappa_i \leq k_2 \qquad e :: \Delta \vdash k_2 \leq k_3}{\delta_i :: \Delta \vdash \kappa_i \leq k_3} \text{ (CSUB-TRANS)}$$

These derivations are all smaller than the original, are all in composite-normal form (since the new cuts are all between individual canonical types), all have cuts of size m at the root, and all contain only smaller cuts otherwise, so we may apply the induction hypothesis to obtain derivations $\delta_i' :: \Delta \vdash \kappa_i \leq k_3$ such that all the cuts in the δ_i have size strictly less than m. Finally, construct the derivation

$$\frac{\dfrac{\delta_i' :: \Delta \vdash \kappa_i \leq k_3}{\Delta \vdash \bigwedge[\kappa_1..\kappa_n] \leq \bigwedge[k_3]} \text{ (CSUB-AE)}}{c' :: \Delta \vdash \alpha \leq k_3,} \text{ (CSUB-TVAR)}$$

which again contains only cuts of size strictly less than m.

Case CSub-Arrow/CSub-Arrow*:*

We are given

$$\frac{D :: \Delta \vdash K_2 \leq K_1 \qquad \delta :: \Delta \vdash \kappa_1 \leq \kappa_2}{d :: \Delta \vdash K_1 \rightarrow \kappa_1 \leq K_2 \rightarrow \kappa_2}\ (\text{CSUB-ARROW})$$

and

$$\frac{E :: \Delta \vdash K_3 \leq K_2 \qquad \epsilon :: \Delta \vdash \kappa_2 \leq \kappa_3}{e :: \Delta \vdash K_2 \rightarrow \kappa_2 \leq K_3 \rightarrow \kappa_3.}\ (\text{CSUB-ARROW})$$

Form the derivation

$$\frac{\dfrac{E :: \Delta \vdash K_3 \leq K_2 \quad D :: \Delta \vdash K_2 \leq K_1}{\Delta \vdash K_3 \leq K_1} \qquad \dfrac{\delta :: \Delta \vdash \kappa_1 \leq \kappa_2 \quad \epsilon :: \Delta \vdash \kappa_2 \leq \kappa_3}{\Delta \vdash \kappa_1 \leq \kappa_3}}{c' :: \Delta \vdash K_1 \rightarrow \kappa_1 \leq K_3 \rightarrow \kappa_3}$$

and note that the two new cuts both have size strictly smaller than $m = \mathit{size}(K_2 \rightarrow \kappa_2)$.

Case CSub-All/CSub-All*:*

We are given

$$\frac{D :: \Delta \vdash K_2 \leq K_1 \qquad \delta :: \Delta, \alpha \leq K_2 \vdash \kappa_1 \leq \kappa_2}{d :: \Delta \vdash \forall \alpha \leq K_1.\ \kappa_1 \leq \forall \alpha \leq K_2.\ \kappa_2}\ (\text{CSUB-ALL})$$

and

$$\frac{E :: \Delta \vdash K_3 \leq K_2 \qquad \epsilon :: \Delta, \alpha \leq K_3 \vdash \kappa_2 \leq \kappa_3}{e :: \Delta \vdash \forall \alpha \leq K_2.\ \kappa_2 \leq \forall \alpha \leq K_3.\ \kappa_3}\ (\text{CSUB-ALL})$$

First, form the derivation

$$\frac{E :: \Delta \vdash K_3 \leq K_2 \qquad D :: \Delta \vdash K_2 \leq K_1}{C' :: \Delta \vdash K_3 \leq K_1.}$$

Next, use narrowing (5.2.3.2) on δ to obtain a derivation

$\delta' :: \Delta, \alpha \leq K_3 \vdash \kappa_1 \leq \kappa_2.$

Note that the cuts in C' and δ' all have size strictly less than $m = \mathit{size}(\forall \alpha \leq K_2.\ \kappa_2)$. Finally, build the derivation

$$\frac{C' \qquad \dfrac{\delta' :: \Delta, \alpha \leq K_3 \vdash \kappa_1 \leq \kappa_2 \qquad \epsilon :: \Delta, \alpha \leq K_3 \vdash \kappa_2 \leq \kappa_3}{\Delta, \alpha \leq K_3 \vdash \kappa_1 \leq \kappa_3}}{c' :: \Delta \vdash \forall \alpha \leq K_1.\ \kappa_1 \leq \forall \alpha \leq K_3.\ \kappa_3}$$

and note that the last new cut also has size strictly less than m. □

5.2.3.6. Definition: A cut c of size m is called an *innermost cut of size* m if no proper subderivation of c is a cut of size greater than or equal to m.

5.2.3.7. Definition: The *total complexity* of a derivation c, written $\mathit{total}(c)$, is the pair $\langle m, n \rangle$, where m is the maximum complexity of any cut of c and n is the number of cuts of complexity m. These pairs are ordered lexicographically.

5.2.3.8. Proposition: [Elimination of one maximal cut] If d is a derivation containing cuts, then there is some derivation d' of smaller total complexity than d.

Proof: Choose an innermost cut c of size m, where m is the maximum size of any cut in d.

— If c is a composite cut, we first use Lemma 5.2.3.4 to put both of its subderivations in composite-normal form, yielding a new derivation c'' of the form

$$\dfrac{\dfrac{\xi_i :: \Delta \vdash \kappa_{f(i)} \leq \iota_i}{\Delta \vdash \bigwedge[\kappa_1..\kappa_m] \leq \bigwedge[\iota_1..\iota_o]} \qquad \dfrac{\delta_j :: \Delta \vdash \iota_{g(j)} \leq \lambda_j}{\Delta \vdash \bigwedge[\iota_1..\iota_o] \leq \bigwedge[\lambda_1..\lambda_n]}}{c'' :: \Delta \vdash \bigwedge[\kappa_1..\kappa_m] \leq \bigwedge[\lambda_1..\lambda_n]} \ (\text{CSUB-TRANS})$$

for some $f \in \{1..o\} \rightarrow \{1..m\}$ and $g \in \{1..n\} \rightarrow \{1..o\}$, such that all cuts in c'' have size strictly less than $m = \mathit{size}(\bigwedge[\iota_1..\iota_o])$. Now construct the derivation

$$\dfrac{\xi_{g(j)} :: \Delta \vdash \kappa_{f(g(j))} \leq \iota_{g(j)} \qquad \delta_j :: \Delta \vdash \iota_{g(j)} \leq \lambda_j}{c' :: \Delta \vdash \bigwedge[\kappa_1..\kappa_m] \leq \bigwedge[\lambda_1..\lambda_n]} \ (\text{CSUB-AE})$$

and note that all cuts in c' again have size smaller than $\mathit{size}(\bigwedge[\iota_1..\iota_o])$, since the cuts we have just added all have cut-types that are proper subphrases of $\bigwedge[\iota_1..\iota_o]$.

— If c is an individual cut, we again use Lemma 5.2.3.4 to put both of its subderivations (and hence c itself) in composite-normal form, yielding a new derivation c'' that still has a cut of maximum size m at its root. Now apply Lemma 5.2.3.5 to produce a derivation c' containing only cuts of size less than m. □

5.2.3.9. Corollary: [Cut elimination] If $d :: k_1 \leq k_2$, then there is some cut-free derivation $d' :: k_1 \leq k_2$.

Proof: By lexicographic induction on the total complexity of d. At each step, we use Proposition 5.2.3.8 to eliminate one cut of maximal size, either reducing the maximal size of cuts in the resulting derivation or keeping the same maximal size but reducing the number of cuts of this size. □

5.2.3.10. Lemma: [Reflexivity elimination] If c is a cut-free derivation, then there is some cut-free derivation c' with the same conclusion such that all instances of CSub-Refl in c' are applied to type variables:

$$\dfrac{}{\Delta \vdash \alpha \leq \alpha} \ (\text{CSUB-REFL})$$

Proof: Any instance $d :: k \leq k$ in c of CSub-Refl whose subject k is *not* a type variable can be replaced by a derivation $c' :: \Delta \vdash k \leq k$ of the required form, using the rules CSub-AE, CSub-Arrow, CSub-All, and (for variables) CSub-Refl. □

5.2.3.11. Definition: A derivation c is in *normal form* if it contains no cuts and uses CSub-Refl only on variables.

5.2.3.12. Theorem: If $\Delta \vdash k_1 \leq k_2$, then there is a normal-form derivation with the same conclusion.

Proof: By Corollary 5.2.3.9 and Lemma 5.2.3.10. □

5.2.4. *Shapes of Normal-Form Subtyping Derivations*

5.2.4.1. Lemma: [Syntax-directedness of canonical subtyping]

1 If $\Delta \vdash K \leq I$, then for every $\iota \in I$ there is some $\kappa \in K$ such that $\Delta \vdash \kappa \leq \iota$.

2 If $\Delta \vdash K_1 \rightarrow \kappa_2 \leq \iota$, then $\iota \equiv I_1 \rightarrow \iota_2$ with $\Delta \vdash I_1 \leq K_1$ and $\Delta \vdash \kappa_2 \leq \iota_2$.

3 If $\Delta \vdash \forall \alpha \leq K_1.\ \kappa_2 \leq \iota$, then $\iota \equiv \forall \alpha \leq I_1.\ \iota_2$ with $\Delta \vdash I_1 \leq K_1$ and $\Delta, \alpha \leq I_1 \vdash \kappa_2 \leq \iota_2$.

4 If $\Delta \vdash \alpha \leq \kappa$, then either $\kappa \equiv \alpha$ or else $\Delta \vdash \Delta(\alpha) \leq \bigwedge[\kappa]$.

Moreover, when the given derivation is in normal form, the derivations promised in each clause are proper subderivations of the original.

Proof: In each case, we are given a derivation of $\Delta \vdash k \leq i$. By Theorem 5.2.3.12, there exists a normal-form derivation of the same statement. This derivation cannot end with an instance of CSub-Refl (except CSub-Refl applied to variables) or CSub-Trans. The result follows by inspection of the remainder of the CSub rules. □

5.2.4.2. Corollary: If $\Delta \vdash \kappa \leq \alpha$, then either $\kappa \equiv \alpha$ or else $\kappa \equiv \beta$ for some β with $\Delta \vdash \Delta(\beta) \leq \bigwedge[\alpha]$.

5.2.5. *Equivalence of Ordinary and Canonical Subtyping* Our next task is to establish that the subtyping relations on ordinary types and on the corresponding canonical types are equivalent in an appropriate sense. We accomplish this by defining a *flattening mapping* $\flat$ from ordinary types to canonical types with the following properties:

1 Flattening preserves subtyping: if $\Gamma \overset{\wedge}{\vdash} \sigma \leq \tau$, then $\Gamma^\flat \overset{\flat}{\vdash} \sigma^\flat \leq \tau^\flat$.

2 Flattening yields a type equivalent to the original: $\Gamma \overset{\wedge}{\vdash} \tau^\flat \sim \tau$.

The second observation (plus narrowing and the fact that the identity injection from the set of canonical types into the set of ordinary types preserves subtyping — if $\Delta \overset{\flat}{\vdash} \kappa \leq \iota$, then $\Delta \overset{\wedge}{\vdash} \kappa \leq \iota$) implies the converse of the first: if the translation of a statement is provable in the canonical system then the original statement is also provable. Thus, the flattening mapping also *reflects* subtyping.

5.2.5.1. Definition: The *flattening mapping* $\flat$ from ordinary types to composite canonical types is defined as follows:

$$\begin{array}{lcl}
\alpha^\flat & = & \bigwedge[\alpha] \\
(\sigma \rightarrow \tau)^\flat & = & \bigwedge[\sigma^\flat \rightarrow \kappa \mid \kappa \in \tau^\flat] \\
(\forall \alpha \leq \sigma.\ \tau)^\flat & = & \bigwedge[\forall \alpha \leq \sigma^\flat.\ \kappa \mid \kappa \in \tau^\flat] \\
\bigwedge[\tau_1..\tau_n]^\flat & = & \bigcup_i \tau_i{}^\flat
\end{array}$$

This mapping is extended pointwise to contexts:

$$\begin{array}{lcl}
\{\}^\flat & = & \{\} \\
(\Gamma, \alpha \leq \tau)^\flat & = & \Gamma^\flat, \alpha \leq \tau^\flat \\
(\Gamma, x{:}\tau)^\flat & = & \Gamma^\flat, x{:}\tau^\flat
\end{array}$$

5.2.5.2. Lemma: [$\flat$ preserves subtyping] If $\Gamma \overset{\wedge}{\vdash} \sigma \leq \tau$ then $\Gamma^\flat \overset{\flat}{\vdash} \sigma^\flat \leq \tau^\flat$.

Proof: See Appendix A. □

Next, we remark that subtyping on canonical types is preserved when canonical types are read as ordinary types.

5.2.5.3. Lemma: If $\Delta \vdash^{\flat} k \leq i$, then $\Delta \vdash^{\wedge} k \leq i$.

Proof: By induction on the structure of a canonical subtyping derivation. □

The last fact needed to establish the equivalence of subtyping on ordinary and canonical types is that the flattening transformation always yields a type equivalent to the original.

5.2.5.4. Lemma: $\Gamma \vdash^{\wedge} \tau^{\flat} \sim \tau$ for all τ and Γ.

Proof: See Appendix A. □

Combining this with the previous lemma, we can show that the translation from ordinary to canonical types reflects subtyping in $F_{\wedge}$.

5.2.5.5. Lemma: [$\flat$ reflects subtyping] If $\Gamma^{\flat} \vdash^{\flat} \sigma^{\flat} \leq \tau^{\flat}$, then $\Gamma \vdash^{\wedge} \sigma \leq \tau$.

Proof: By Lemma 5.2.5.3, $\Gamma^{\flat} \vdash^{\wedge} \sigma^{\flat} \leq \tau^{\flat}$. By Lemma 5.2.5.4, $\Gamma \vdash^{\wedge} \Gamma(\alpha) \leq \Gamma^{\flat}(\alpha)$ for each $\alpha \in dom(\Gamma^{\flat})$. By narrowing (5.2.3.2), $\Gamma \vdash^{\wedge} \sigma^{\flat} \leq \tau^{\flat}$. By Lemma 5.2.5.4 again, $\Gamma \vdash^{\wedge} \sigma \leq \sigma^{\flat}$ and $\Gamma \vdash^{\wedge} \tau^{\flat} \leq \tau$. By two applications of SUB-TRANS, $\Gamma \vdash^{\wedge} \sigma \leq \tau$. □

Lemmas 5.2.5.2 and 5.2.5.5 together show that the subtype relations on ordinary and canonical types correspond appropriately:

5.2.5.6. Theorem: [Equivalence of ordinary and canonical subtyping]

$$\Gamma \vdash^{\wedge} \sigma \leq \tau \quad \text{iff} \quad \Gamma^{\flat} \vdash^{\flat} \sigma^{\flat} \leq \tau^{\flat}.$$

5.2.6. *Subtyping Algorithm* The definition of canonical subtyping leads directly to an algorithm for deciding the subtype relation on $F_{\wedge}$ types: to check whether $\Gamma \vdash \sigma \leq \tau$, flatten Γ, σ, and τ and then check whether $\Gamma^{\flat} \vdash^{\flat} \sigma^{\flat} \leq \tau^{\flat}$. In this section we describe a more efficient algorithm that operates directly on $F_{\wedge}$ types, effectively performing the flattening translation on the fly. The algorithm presented here generalizes one described by Reynolds for deciding the subtype relation of Forsythe [personal communication, 1988].

Given Γ, σ, and τ, this algorithm first performs a complete analysis of the structure of τ. Intuitively, whenever τ has the form $\tau_1 \rightarrow \tau_2$ or $\forall \alpha \leq \tau_1.\ \tau_2$, it pushes the left-hand side — τ_1 or $\alpha \leq \tau_1$ — onto a queue of pending left-hand sides and proceeds recursively with the analysis of τ_2. When τ has the form of an intersection, it calls itself recursively on each of the elements. When τ has finally been reduced to a type variable, the algorithm begins analyzing σ, matching left-hand sides of arrow and polymorphic types against the queue of pending left-hand sides from τ. In the base case, when both σ and τ have been reduced to variables, the algorithm first checks whether they are identical; if so, and if the queue of pending left-hand sides is empty, the algorithm immediately returns *true*. Otherwise, the variable σ is replaced by its upper bound from Γ and the analysis continues as before.

5.2.6.1. Definition: Let X be a finite sequence of elements of the set

$$\{\tau \mid \tau \text{ a type}\} \cup \{\alpha \leq \tau \mid \alpha \text{ a type variable and } \tau \text{ a type}\}.$$

Define the type $X \Rightarrow \tau$ as follows:

$$\begin{array}{lcl} [\,]\Rightarrow\tau & = & \tau \\ [\sigma, X]\Rightarrow\tau & = & \sigma \to (X\Rightarrow\tau) \\ [\alpha{\leq}\sigma, X]\Rightarrow\tau & = & \forall\alpha{\leq}\sigma.\ (X\Rightarrow\tau). \end{array}$$

Note that every type τ has either the form $X\Rightarrow\alpha$ or the form $X\Rightarrow\bigwedge[\tau_1..\tau_n]$ for a unique X. From the definitions of $\flat$ and $X\Rightarrow\tau$, the following facts are immediate:

5.2.6.2. Lemma:

1 $(X\Rightarrow(\bigwedge[\tau_1..\tau_n]))^\flat = \bigcup_i (X\Rightarrow\tau_i)^\flat$.

2 $(X\Rightarrow(\tau_1\to\tau_2))^\flat = ([X, \tau_1] \Rightarrow \tau_2))^\flat$.

3 $(X\Rightarrow(\forall\alpha{\leq}\tau_1.\ \tau_2))^\flat = ([X, \alpha{\leq}\tau_1] \Rightarrow \tau_2)^\flat$.

5.2.6.3. Definition: The four-place algorithmic subtyping relation $\Gamma \overset{A}{\vdash} \sigma \leq X\Rightarrow\tau$ is the least relation closed under the following rules:

$$\frac{\text{for all } i,\ \Gamma \vdash \sigma \leq X \Rightarrow \tau_i}{\Gamma \vdash \sigma \leq X \Rightarrow \bigwedge[\tau_1..\tau_n]} \qquad \text{(ASubR-Inter)}$$

$$\frac{\text{for some } i,\ \Gamma \vdash \sigma_i \leq X \Rightarrow \alpha}{\Gamma \vdash \bigwedge[\sigma_1..\sigma_n] \leq X \Rightarrow \alpha} \qquad \text{(ASubL-Inter)}$$

$$\frac{\Gamma \vdash \tau_1 \leq [\,] \Rightarrow \sigma_1 \qquad \Gamma \vdash \sigma_2 \leq X_2 \Rightarrow \alpha}{\Gamma \vdash \sigma_1\to\sigma_2 \leq [\tau_1, X_2] \Rightarrow \alpha} \qquad \text{(ASubL-Arrow)}$$

$$\frac{\Gamma \vdash \tau_1 \leq [\,] \Rightarrow \sigma_1 \qquad \Gamma, \beta{\leq}\tau_1 \vdash \sigma_2 \leq X_2 \Rightarrow \alpha}{\Gamma \vdash \forall\beta{\leq}\sigma_1.\ \sigma_2 \leq [\beta{\leq}\tau_1, X_2] \Rightarrow \alpha} \qquad \text{(ASubL-All)}$$

$$\Gamma \vdash \alpha \leq [\,] \Rightarrow \alpha \qquad \text{(ASubL-Refl)}$$

$$\frac{\Gamma \vdash \Gamma(\beta) \leq X \Rightarrow \alpha}{\Gamma \vdash \beta \leq X \Rightarrow \alpha} \qquad \text{(ASubL-TVar)}$$

In this presentation, we have slightly optimized the above description, replacing all the individual steps of pushing left-hand sides of arrows and quantifiers onto the queue with a "matching" syntax that does it all at once. It may be helpful to think of the algorithm as being implicitly augmented with these rules:

$$\frac{\Gamma \vdash \sigma \leq [X, \tau] \Rightarrow \tau'}{\Gamma \vdash \sigma \leq X \Rightarrow (\tau\to\tau')}$$

$$\frac{\Gamma \vdash \sigma \leq [X, \tau_i] \Rightarrow \tau'}{\Gamma \vdash \sigma \leq X \Rightarrow \forall\alpha{\leq}\tau.\ \tau'}$$

We sometimes decorate the turnstile symbol $\overset{A}{\vdash}$ to distinguish algorithmic derivations from derivations in other calculi.

5.2.6.4. Definition: We write $\text{Dist}^*_{\Gamma,\bigwedge[X\Rightarrow\tau_1..X\Rightarrow\tau_n]}$ (or just Dist^* when the appropriate subscript is clear) for the following compound derivation:

$$\textsc{Dist}^*_{\Gamma,\bigwedge[[]\Rightarrow\tau_1..[]\Rightarrow\tau_n]} \quad = \quad \dfrac{\text{(SUB-REFL)}}{\Gamma \overset{\wedge}{\vdash} \bigwedge[[]\Rightarrow\tau_1 \,..\, []\Rightarrow\tau_n] \le []\Rightarrow\bigwedge[\tau_1..\tau_n]}$$

$$\textsc{Dist}^*_{\Gamma,\bigwedge[\sigma\to(X'\Rightarrow\tau_1)..\sigma\to(X'\Rightarrow\tau_n)]} \quad =$$

$$\dfrac{\dfrac{\text{(SUB-DIST-IA)}}{\begin{array}{lll}\Gamma & \overset{\wedge}{\vdash} & \bigwedge[\sigma\to(X'\Rightarrow\tau_1) \,..\, \sigma\to(X'\Rightarrow\tau_n)] \\ & \le & \sigma\to\bigwedge[X'\Rightarrow\tau_1 \,..\, X'\Rightarrow\tau_n]\end{array}} \qquad \dfrac{\dfrac{}{\Gamma\vdash\sigma\le\sigma} \qquad \textsc{Dist}^*_{\Gamma,\bigwedge[X'\Rightarrow\tau_1..X'\Rightarrow\tau_n]}}{\begin{array}{lll}\Gamma & \overset{\wedge}{\vdash} & \sigma\to\bigwedge[X'\Rightarrow\tau_1 \,..\, X'\Rightarrow\tau_n] \\ & \le & \sigma\to(X'\Rightarrow\bigwedge[\tau_1..\tau_n])\end{array}} \text{(SUB-ARROW)}}{\Gamma \overset{\wedge}{\vdash} \bigwedge[\sigma\to(X'\Rightarrow\tau_1) \,..\, \sigma\to(X'\Rightarrow\tau_n)] \le \sigma\to(X'\Rightarrow\bigwedge[\tau_1..\tau_n])} \text{(SUB-TRANS)}$$

$$\textsc{Dist}^*_{\Gamma,\bigwedge[\forall\alpha\le\sigma.\,(X'\Rightarrow\tau_1)..\forall\alpha\le\sigma.\,(X'\Rightarrow\tau_n)]} \quad =$$

$$\dfrac{\dfrac{\text{(SUB-DIST-IQ)}}{\begin{array}{lll}\Gamma & \overset{\wedge}{\vdash} & \bigwedge[\forall\alpha\le\sigma.\,(X'\Rightarrow\tau_1) \,..\, \forall\alpha\le\sigma.\,(X'\Rightarrow\tau_n)] \\ & \le & \forall\alpha\le\sigma.\,\bigwedge[X'\Rightarrow\tau_1 \,..\, X'\Rightarrow\tau_n]\end{array}} \qquad \dfrac{\dfrac{}{\Gamma\vdash\sigma\le\sigma} \qquad \textsc{Dist}^*_{(\Gamma,\alpha\le\sigma),\bigwedge[X'\Rightarrow\tau_1..X'\Rightarrow\tau_n]}}{\begin{array}{lll}\Gamma & \overset{\wedge}{\vdash} & \forall\alpha\le\sigma.\,\bigwedge[X'\Rightarrow\tau_1 \,..\, X'\Rightarrow\tau_n] \\ & \le & \forall\alpha\le\sigma.\,(X'\Rightarrow\bigwedge[\tau_1..\tau_n])\end{array}}}{\Gamma \overset{\wedge}{\vdash} \bigwedge[\forall\alpha\le\sigma.\,(X'\Rightarrow\tau_1) \,..\, \forall\alpha\le\sigma.\,(X'\Rightarrow\tau_n)] \le \forall\alpha\le\sigma.\,(X'\Rightarrow\bigwedge[\tau_1..\tau_n])}$$

5.2.6.5. Definition: Let $c :: \Gamma \overset{\cdot}{\vdash} \sigma \le \tau$ be an algorithmic subtyping derivation. Then $c^\wedge :: \Gamma \overset{\wedge}{\vdash} \sigma \le \tau$ is the following ordinary derivation:

$$\left(\dfrac{\text{for all } i,\ c_i :: \Gamma \overset{\cdot}{\vdash} \sigma \le X\Rightarrow\tau_i}{\Gamma \overset{\cdot}{\vdash} \sigma \le X\Rightarrow\bigwedge[\tau_1..\tau_n]} \text{(ASUBR-INTER)}\right)^\wedge \quad =$$

$$\dfrac{\dfrac{c_1^\wedge \cdots c_n^\wedge}{\Gamma \overset{\wedge}{\vdash} \sigma \le \bigwedge[X\Rightarrow\tau_1 \,..\, X\Rightarrow\tau_n]} \text{(SUB-INTER-G)} \qquad \begin{array}{r}\textsc{Dist}^* :: \Gamma \overset{\wedge}{\vdash} \bigwedge[X\Rightarrow\tau_1 \,..\, X\Rightarrow\tau_n] \\ \le X\Rightarrow\bigwedge[\tau_1..\tau_n]\end{array}}{\Gamma \overset{\wedge}{\vdash} \sigma \le X\Rightarrow\bigwedge[\tau_1..\tau_n]}$$

$$\left(\dfrac{\text{for some } i,\ c_i :: \Gamma \overset{\cdot}{\vdash} \sigma_i \le X\Rightarrow\alpha}{\Gamma \overset{\cdot}{\vdash} \bigwedge[\sigma_1..\sigma_n] \le X\Rightarrow\alpha} \text{(ASUBL-INTER)}\right)^\wedge \quad =$$

$$\dfrac{\dfrac{}{\Gamma \overset{\wedge}{\vdash} \bigwedge[\sigma_1..\sigma_n] \le \sigma_i} \text{(SUB-INTER-LB)} \qquad c_i^\wedge}{\Gamma \overset{\wedge}{\vdash} \bigwedge[\sigma_1..\sigma_n] \le X\Rightarrow\alpha}$$

$$\left(\dfrac{c_1 :: \Gamma \overset{\cdot}{\vdash} \tau_1 \le []\Rightarrow\sigma_1 \qquad c_2 :: \Gamma \overset{\cdot}{\vdash} \sigma_2 \le X\Rightarrow\alpha}{\Gamma \overset{\cdot}{\vdash} \sigma_1\to\sigma_2 \le [\tau_1, X]\Rightarrow\alpha} \text{(ASUBL-ARROW)}\right)^\wedge \quad =$$

$$\dfrac{c_1^\wedge \qquad c_2^\wedge}{\Gamma \overset{\wedge}{\vdash} \sigma_1\to\sigma_2 \le [\tau_1, X]\Rightarrow\alpha} \text{(SUB-ARROW)}$$

$$\left(\frac{c_1 :: \Gamma \stackrel{\cdot}{\vdash} \tau_1 \leq [\,]{\Rightarrow}\sigma_1 \qquad c_2 :: \Gamma,\, \beta{\leq}\tau_1 \stackrel{\cdot}{\vdash} \sigma_2 \leq X{\Rightarrow}\alpha}{\Gamma \stackrel{\cdot}{\vdash} \forall\beta{\leq}\sigma_1.\ \sigma_2 \leq [\beta{\leq}\tau_1, X]{\Rightarrow}\alpha}\ \text{(ASUBL-ALL)}\right)^{\wedge} =$$

$$\frac{c_1^{\wedge} \qquad c_2^{\wedge}}{\Gamma \stackrel{\wedge}{\vdash} \forall\beta{\leq}\sigma_1.\ \sigma_2 \leq [\beta{\leq}\tau_1, X]{\Rightarrow}\alpha}\ \text{(SUB-ALL)}$$

$$\left(\frac{}{\Gamma \stackrel{\cdot}{\vdash} \alpha \leq [\,]{\Rightarrow}\alpha}\ \text{(ASUBL-REFL)}\right)^{\wedge} = \frac{}{\Gamma \stackrel{\wedge}{\vdash} \alpha \leq [\,]{\Rightarrow}\alpha}\ \text{(SUB-REFL)}$$

$$\left(\frac{c_1 :: \Gamma \stackrel{\cdot}{\vdash} \Gamma(\beta) \leq X{\Rightarrow}\alpha}{\Gamma \stackrel{\cdot}{\vdash} \beta \leq X{\Rightarrow}\alpha}\ \text{(ASUBL-TVAR)}\right)^{\wedge} = \frac{\dfrac{}{\Gamma \stackrel{\wedge}{\vdash} \beta \leq \Gamma(\beta)}\ \text{(SUB-TVAR)} \qquad c_1^{\wedge}}{\Gamma \stackrel{\wedge}{\vdash} \beta \leq X{\Rightarrow}\alpha}\ \text{(SUB-TRANS)}$$

5.2.6.6. Theorem: [Soundness of the algorithm] If $\Gamma \stackrel{\cdot}{\vdash} \sigma \leq X{\Rightarrow}\tau$ then $\Gamma \stackrel{\wedge}{\vdash} \sigma \leq X{\Rightarrow}\tau$.

Proof: By the well-formedness of the translation in Definition 5.2.6.5. □

We must now check that the relation defined by these rules coincides with the subtype relation on canonical types, from which it follows, by Theorem 5.2.5.6, that the algorithm gives a semi-decision procedure for the $F_\wedge$ subtype relation.

5.2.6.7. Lemma: [Completeness of the algorithm with respect to canonical subtyping] If $\Gamma^\flat \stackrel{\flat}{\vdash} \sigma^\flat \leq (X{\Rightarrow}\tau)^\flat$, then $\Gamma \stackrel{\cdot}{\vdash} \sigma \leq X{\Rightarrow}\tau$.

Proof: By induction on the size (n.b. not the structure) of a normal-form derivation of $\Gamma^\flat \stackrel{\flat}{\vdash} \sigma^\flat \leq (X{\Rightarrow}\tau)^\flat$, with a sub-induction on the form of τ and, when $\tau \equiv \alpha$, a sub-sub-induction on the form of σ. Proceed by cases on the form of τ and σ.

Case: $\tau \equiv \bigwedge[\tau_1..\tau_n] \qquad \Gamma^\flat \stackrel{\flat}{\vdash} \sigma^\flat \leq (X{\Rightarrow}\bigwedge[\tau_1..\tau_n])^\flat$

By Lemma 5.2.6.2(1), $\Gamma^\flat \stackrel{\flat}{\vdash} \sigma^\flat \leq \bigcup_i (X{\Rightarrow}\tau_i)^\flat$. By the syntax-directedness of canonical subtyping (5.2.4.1(1)), for every $\iota \in \bigcup_i (X{\Rightarrow}\tau_i)^\flat$ there is some $\kappa \in \sigma^\flat$ and a subderivation of the original whose conclusion is $\Gamma^\flat \stackrel{\flat}{\vdash} \kappa \leq \iota$. In particular, for each i and every $\iota \in (X{\Rightarrow}\tau_i)^\flat$ there is some $\kappa \in \sigma^\flat$ such that $\Gamma^\flat \stackrel{\flat}{\vdash} \kappa \leq \iota$. By CSub-AE, $\Gamma^\flat \stackrel{\flat}{\vdash} \sigma^\flat \leq (X{\Rightarrow}\tau_i)^\flat$. This derivation is no larger than the original and τ_i is smaller than τ, so, by the main or sub-induction hypothesis, $\Gamma \stackrel{\cdot}{\vdash} \sigma \leq X{\Rightarrow}\tau_i$. Then by rule ASubR-Inter, $\Gamma \stackrel{\cdot}{\vdash} \sigma \leq X{\Rightarrow}\bigwedge[\tau_1..\tau_n]$.

Case: $\tau \equiv \alpha \qquad \sigma \equiv \bigwedge[\sigma_1..\sigma_m] \qquad \Gamma^\flat \stackrel{\flat}{\vdash} \bigwedge[\sigma_1..\sigma_n]^\flat \leq (X{\Rightarrow}\alpha)^\flat$

Since $(X{\Rightarrow}\alpha)^\flat \equiv \bigwedge[\iota]$ is a singleton, the syntax-directedness of canonical subtyping (5.2.4.1(1)) implies that for some $\kappa \in \sigma^\flat$ we have $\Gamma^\flat \stackrel{\flat}{\vdash} \kappa \leq \iota$ as a subderivation of the original. Since $\sigma^\flat \equiv \bigcup_i \sigma_i^\flat$, there is some σ_i such that $\kappa \in \sigma_i^\flat$. CSub-AE then gives $\Gamma^\flat \stackrel{\flat}{\vdash} \sigma_i^\flat \leq (X{\Rightarrow}\alpha)^\flat$. This derivation is no larger than the original and σ_i is strictly smaller than σ, so by either the main or the sub-sub-induction hypothesis, $\Gamma \stackrel{\cdot}{\vdash} \sigma_i \leq X{\Rightarrow}\alpha$. By rule ASubL-Inter, $\Gamma \stackrel{\cdot}{\vdash} \sigma \leq X{\Rightarrow}\alpha$.

Case: $\tau \equiv \alpha \qquad \sigma \equiv \sigma_1{\rightarrow}\sigma_2 \qquad \Gamma^\flat \stackrel{\flat}{\vdash} (\sigma_1{\rightarrow}\sigma_2)^\flat \leq (X{\Rightarrow}\alpha)^\flat$

Since $(X{\Rightarrow}\alpha)^\flat \equiv \bigwedge[\iota]$ is a singleton, the syntax-directedness of canonical subtyping (5.2.4.1(1)) implies that for some $K_1{\rightarrow}\kappa_2 \in \sigma^\flat$ we have $\Gamma^\flat \stackrel{\flat}{\vdash} K_1{\rightarrow}\kappa_2 \leq \iota$ as a subderivation. By syntax-directedness again (5.2.4.1(2)), ι must have the form $I_1{\rightarrow}\iota_2$, with $\Gamma^\flat \overset{\flat}{\vdash} I_1 \leq K_1$ and $\Gamma^\flat \overset{\flat}{\vdash} \kappa_2 \leq \iota_2$ as subderivations. By Definition 5.2.6.1, $X \equiv [\tau_1, X_2]$, where $I_1 \equiv \tau_1^\flat$ and $\bigwedge[\iota_2] \equiv (X_2 \Rightarrow \alpha)^\flat$. Since $\kappa_2 \in {\sigma_2}^\flat$, we have $\Gamma^\flat \overset{\flat}{\vdash} {\sigma_2}^\flat \leq (X_2 \Rightarrow \alpha)^\flat$ by CSub-AE. This derivation is no larger than the original, and σ is strictly smaller, so by either the main or the sub-sub-induction hypothesis, $\Gamma \overset{!}{\vdash} \sigma_2 \leq X_2 \Rightarrow \alpha$. Also, by the main induction hypothesis, $\Gamma \overset{!}{\vdash} \tau_1 \leq [\,] \Rightarrow \sigma_1$. By rule ASubL-Arrow, $\Gamma \overset{!}{\vdash} \sigma_1 \to \sigma_2 \leq \tau_1 \to (X_2 \Rightarrow \tau_2)$, i.e., $\Gamma \overset{!}{\vdash} \sigma_1 \to \sigma_2 \leq [\tau_1, X_2] \Rightarrow \tau_2$.

Case: $\tau \equiv \alpha \qquad \sigma \equiv \forall \alpha {\leq} \sigma_1 .\ \sigma_2$

Similar.

Case: $\tau \equiv \alpha \qquad \sigma \equiv \beta \qquad \Gamma^\flat \overset{\flat}{\vdash} \beta^\flat \leq (X \Rightarrow \alpha)^\flat$

Since $\beta^\flat \equiv \bigwedge[\beta]$ and $(X \Rightarrow \alpha)^\flat \equiv \bigwedge[\iota]$ are both singletons, the syntax-directedness of canonical subtyping (5.2.4.1(1)) gives $\Gamma^\flat \overset{\flat}{\vdash} \beta \leq \iota$ as a subderivation. By syntax-directedness again (5.2.4.1(4)), either $\iota \equiv \beta$ or $\Gamma^\flat \overset{\flat}{\vdash} \Gamma^\flat(\beta) \leq \bigwedge[\iota]$. In the first case, rule ASubL-Refl gives the desired result immediately. In the second case, the main induction hypothesis gives $\Gamma \overset{!}{\vdash} \Gamma(\beta) \leq X \Rightarrow \alpha$, from which rule ASubL-TVar again yields $\Gamma \overset{!}{\vdash} \beta \leq X \Rightarrow \alpha$. □

5.2.6.8. Theorem: [Completeness of the algorithm with respect to ordinary subtyping] If $\Gamma \overset{\wedge}{\vdash} \sigma \leq X \Rightarrow \tau$ then $\Gamma \overset{!}{\vdash} \sigma \leq X \Rightarrow \tau$.

Proof: By the equivalence of ordinary and canonical subtyping (Theorem 5.2.5.6) and the completeness of the algorithm with respect to canonical subtyping (Lemma 5.2.6.7). □

5.2.6.9. Definition: The more convenient three-place relation $\Gamma \overset{!}{\vdash} \sigma \leq \tau$ may be defined as

$$\Gamma \overset{!}{\vdash} \sigma \leq \tau \quad \text{iff} \quad \Gamma \overset{!}{\vdash} \sigma \leq X \Rightarrow \phi,$$

where $\tau \equiv X \Rightarrow \phi$ and either $\phi \equiv \bigwedge[\phi_1 .. \phi_n]$ or $\phi \equiv \alpha$.

5.2.6.10. Theorem: [Equivalence of ordinary and syntax-directed subtyping]

$$\Gamma \overset{\wedge}{\vdash} \sigma \leq \tau \quad \text{iff} \quad \Gamma \overset{!}{\vdash} \sigma \leq \tau.$$

5.3. *Typechecking*

We now present an algorithm for synthesizing types for $F_\wedge$ terms. Given a term e and a context Γ (where e is closed in Γ), the algorithm constructs a minimal type σ for e under Γ — that is, a type σ such that $\Gamma \vdash e \in \sigma$, and such that any other type that can be derived for e from these rules is a supertype of σ.

The algorithm can be explained by separating the typing rules of Definition 3.2.2 into two sets: the *structural* or *syntax-directed* rules Var, Arrow-I, Arrow-E, All-I, All-E, and For, whose applicability depends on the form of e, and the non-structural rules Inter-I and Sub, which can be applied without regard to the form of e. The non-structural rules are then removed from the system and their possible effects accounted for by modifying the structural rules Var, Arrow-E, All-E, and For appropriately.

The main novel source of difficulty here is the application rules Arrow-E and All-E.

An application $(e_1\ e_2)$ in the original system has *every* type τ_2 such that e_1 can be shown to have some type $\tau_1 \rightarrow \tau_2$ and e_2 can be shown to have type τ_1, where the rule SUB may be used on both sides to promote the types of e_1 and e_2 to supertypes with appropriate shapes. For example, if

$$e_1 \in (\sigma_1 \rightarrow \sigma_2) \wedge (\forall\alpha{\leq}\sigma_3.\ \sigma_4) \wedge (\sigma_5 \rightarrow \sigma_6) \wedge (\sigma_7 \rightarrow \sigma_8)$$
$$e_2 \in \sigma_1 \wedge (\forall\alpha{\leq}\sigma_3.\ \sigma_4) \wedge \sigma_5,$$

then

$$(e_1\ e_2)$$

has both types σ_2 and σ_6, and hence (by INTER-I) also type $\sigma_2 \wedge \sigma_6$.

To deal with this flexibility deterministically, we observe that the set of supertypes of $(\sigma_1 \rightarrow \sigma_2) \wedge (\forall\alpha{\leq}\sigma_3.\ \sigma_4) \wedge (\sigma_5 \rightarrow \sigma_6) \wedge (\sigma_7 \rightarrow \sigma_8)$ that have the appropriate shape to appear as the type of e_1 in an instance of ARROW-E can be characterized finitely:

$$\begin{aligned}&\mathit{arrowbasis}((\sigma_1 \rightarrow \sigma_2) \wedge (\forall\alpha{\leq}\sigma_3.\ \sigma_4) \wedge (\sigma_5 \rightarrow \sigma_6) \wedge (\sigma_7 \rightarrow \sigma_8))\\ &\quad = [\sigma_1 \rightarrow \sigma_2,\ \sigma_5 \rightarrow \sigma_6,\ \sigma_7 \rightarrow \sigma_8].\end{aligned}$$

It is then a simple matter to characterize the possible types for $(e_1\ e_2)$ by checking whether the minimal type of e_2 is a subtype of each domain type in the finite arrow basis of the minimal type of e_1. Type applications are handled similarly.

5.3.1. *Finite Bases for Applications*

5.3.1.1. Definition: The functions $\mathit{arrowbasis}_\Gamma$ and $\mathit{allbasis}_\Gamma$ are defined as follows:

$$\begin{array}{lcl}
\mathit{arrowbasis}_\Gamma(\alpha) & = & \mathit{arrowbasis}_\Gamma(\Gamma(\alpha))\\
\mathit{arrowbasis}_\Gamma(\tau_1 \rightarrow \tau_2) & = & [\tau_1 \rightarrow \tau_2]\\
\mathit{arrowbasis}_\Gamma(\forall\alpha{\leq}\tau_1.\ \tau_2) & = & [\,]\\
\mathit{arrowbasis}_\Gamma(\bigwedge[\tau_1..\tau_n]) & = & \mathit{arrowbasis}_\Gamma(\tau_1) * \cdots * \mathit{arrowbasis}_\Gamma(\tau_n)\\
\\
\mathit{allbasis}_\Gamma(\alpha) & = & \mathit{allbasis}_\Gamma(\Gamma(\alpha))\\
\mathit{allbasis}_\Gamma(\tau_1 \rightarrow \tau_2) & = & [\,]\\
\mathit{allbasis}_\Gamma(\forall\alpha{\leq}\tau_1.\ \tau_2) & = & [\forall\alpha{\leq}\tau_1.\ \tau_2]\\
\mathit{allbasis}_\Gamma(\bigwedge[\tau_1..\tau_n]) & = & \mathit{allbasis}_\Gamma(\tau_1) * \cdots * \mathit{allbasis}_\Gamma(\tau_n).
\end{array}$$

To check that these definitions are proper, note that a closed context cannot contain cyclic chains of variable references where $\alpha_0 \in FTV(\Gamma(\alpha_1))$, $\alpha_1 \in FTV(\Gamma(\alpha_2))$, ..., $\alpha_n \in FTV(\Gamma(\alpha_0))$.

The next two lemmas verify that $\mathit{arrowbasis}_\Gamma$ and $\mathit{allbasis}_\Gamma$ compute finite bases for the sets of arrow types and polymorphic types above a given type.

5.3.1.2. Lemma: [Finite $\rightarrow$ basis computed by $\mathit{arrowbasis}_\Gamma$]

1 $\Gamma \overset{\wedge}{\vdash} \sigma \leq \bigwedge(\mathit{arrowbasis}_\Gamma(\sigma))$.

2 If $\Gamma \overset{\wedge}{\vdash} \sigma \leq \tau_1 \rightarrow \tau_2$, then $\Gamma \overset{\wedge}{\vdash} \bigwedge(\mathit{arrowbasis}_\Gamma(\sigma)) \leq \tau_1 \rightarrow \tau_2$.

Proof: See Appendix A. □

5.3.1.3. Lemma: [Finite $\forall$ basis computed by $\mathit{allbasis}_\Gamma$]

1 $\Gamma \vdash \sigma \leq \bigwedge(\mathit{allbasis}_\Gamma(\sigma))$.

2 If $\Gamma \vdash \sigma \leq (\forall\alpha{\leq}\tau_1.\ \tau_2)$, then $\Gamma \vdash \bigwedge(\mathit{allbasis}_\Gamma(\sigma)) \leq (\forall\alpha{\leq}\tau_1.\ \tau_2)$.

Proof: Similar. □

The crucial step in the correctness proof for the type synthesis algorithm is showing that application and type application are correctly characterized by the sets computed by *arrowbasis* and *allbasis*.

5.3.1.4. Lemma: [Application]
If

$$M \equiv [\phi_1{\rightarrow}\psi_1 \mathinner{..} \phi_n{\rightarrow}\psi_n]$$
$$V \equiv [\psi_i \mid \Gamma \vdash \sigma \leq \phi_i]$$
$$\Gamma \vdash \bigwedge M \leq \tau_1{\rightarrow}\tau_2$$
$$\Gamma \vdash \sigma \leq \tau_1,$$

then

$$\Gamma \vdash \bigwedge V \leq \tau_2.$$

Proof: See Appendix A. □

5.3.1.5. Lemma: [Type application]
If

$$M \equiv [(\forall\alpha{\leq}\phi_1.\ \psi_1) \mathinner{..} (\forall\alpha{\leq}\phi_n.\ \psi_n)]$$
$$V \equiv [\{\sigma/\alpha\}\psi_i \mid \Gamma \vdash \sigma \leq \phi_i]$$
$$\Gamma \vdash \bigwedge M \leq (\forall\alpha{\leq}\tau_1.\ \tau_2)$$
$$\Gamma \vdash \sigma \leq \tau_1,$$

then

$$\Gamma \vdash \bigwedge V \leq \{\sigma/\alpha\}\tau_2.$$

Proof: See Appendix A. □

5.3.2. *Type Synthesis*

5.3.2.1. Definition: The three-place *type synthesis relation* $\Gamma \vdash e \in \tau$ is the least relation closed under the following rules:

$$\Gamma \vdash x \in \Gamma(x) \tag{A-Var}$$

$$\frac{\Gamma,\ x{:}\tau_1 \vdash e \in \tau_2}{\Gamma \vdash \lambda x{:}\tau_1.\ e \in \tau_1{\rightarrow}\tau_2} \tag{A-Arrow-I}$$

$$\frac{\Gamma \vdash e_1 \in \sigma_1 \qquad \Gamma \vdash e_2 \in \sigma_2}{\Gamma \vdash e_1\ e_2 \in \bigwedge[\psi_i \mid (\phi_i{\rightarrow}\psi_i) \in \mathit{arrowbasis}_\Gamma(\sigma_1) \text{ and } \Gamma \vdash \sigma_2 \leq \phi_i]} \tag{A-Arrow-E}$$

$$\frac{\Gamma,\ \alpha{\leq}\tau_1 \vdash e \in \tau_2}{\Gamma \vdash \Lambda\alpha{\leq}\tau_1.\ e \in \forall\alpha{\leq}\tau_1.\ \tau_2} \tag{A-All-I}$$

$$\frac{\Gamma \vdash e \in \sigma}{\Gamma \vdash e[\tau] \in \bigwedge[\{\tau/\alpha\}\psi_i \mid (\forall\alpha{\leq}\phi_i.\ \psi_i) \in \mathit{allbasis}_\Gamma(\sigma) \text{ and } \Gamma \vdash \tau \leq \phi_i]} \tag{A-All-E}$$

$$\frac{\text{for all } i,\ \Gamma \vdash \{\sigma_i/\alpha\}e \in \tau_i}{\Gamma \vdash \mathit{for}\ \alpha\ \mathit{in}\ \sigma_1..\sigma_n.\ e \in \bigwedge[\tau_1..\tau_n]} \quad \text{(A-For)}$$

Turnstiles in type synthesis derivations are sometimes marked $\vdash$ to distinguish them from derivations in other calculi.

5.3.2.2. Lemma: [Syntax-directedness of the type synthesis rules] For given Γ and e, there is at most one rule that can be used to establish $\Gamma \vdash e \in \tau$ for some τ. Moreover, the existence of such a derivation can be established from the form of e and the results of applying the type synthesis procedure to proper subphrases of e plus a finite number of applications of the subroutine for checking the subtyping relation. In particular:

1 If $\Gamma \vdash x \in \theta$, then $\theta \equiv \Gamma(x)$.

2 If $\Gamma \vdash \lambda x{:}\tau_1.\ e \in \theta$, then $\theta \equiv \tau_1 \rightarrow \tau_2$, where $\Gamma,\ x{:}\tau_1 \vdash e \in \tau_2$ as a subderivation.

3 If $\Gamma \vdash e_1 e_2 \in \theta$, then $\theta \equiv \bigwedge[\psi_i \mid (\phi_i \rightarrow \psi_i) \in \mathit{arrowbasis}_\Gamma(\sigma_1)$ and $\Gamma \vdash \sigma_2 \leq \phi_i]$, where $\Gamma \vdash e_1 \in \sigma_1$ and $\Gamma \vdash e_2 \in \sigma_2$ as subderivations.

4 If $\Gamma \vdash \Lambda\alpha{\leq}\tau_1.\ e \in \theta$, then $\theta \equiv \forall\alpha{\leq}\tau_1.\ \tau_2$, where $\Gamma,\ \alpha{\leq}\tau_1 \vdash e \in \tau_2$ as a subderivation.

5 If $\Gamma \vdash e[\tau] \in \theta$, then $\theta \equiv \bigwedge[\{\tau/\alpha\}\psi_i \mid (\forall\alpha{\leq}\phi_i.\ \psi_i) \in \mathit{allbasis}_\Gamma(\sigma_1)$ and $\Gamma \vdash \tau \leq \phi_i]$, where $\Gamma \vdash e \in \sigma_1$ as a subderivation.

6 If $\Gamma \vdash \mathit{for}\ \alpha\ \mathit{in}\ \sigma_1..\sigma_n.\ e \in \theta$, then $\theta \equiv \bigwedge[\tau_1..\tau_n]$, where $\Gamma \vdash \{\sigma_i/\alpha\}e \in \tau_i$, for each i, as a subderivation.

Proof: By inspection. □

5.3.2.3. Remark: The syntax-directedness of algorithmic derivations permits us to skip introducing a linear shorthand, as we did for ordinary and canonical derivations, since terms themselves are essentially the shorthand we need.

5.3.2.4. Definition: Let $c :: \Gamma \vdash e \in \tau$ be a typing derivation from the algorithmic rules (5.3.2.1). Then $s^\wedge :: \Gamma \overset{\wedge}{\vdash} e \in \tau$ is the following derivation from the ordinary typing rules:

$$\left(\frac{}{\Gamma \vdash x \in \Gamma(x)}\ \text{(A-VAR)}\right)^\wedge = \frac{}{\Gamma \overset{\wedge}{\vdash} x \in \Gamma(x)}\ \text{(VAR)}$$

$$\left(\frac{s_1 :: \Gamma,\ x{:}\tau_1 \vdash e \in \tau_2}{\Gamma \vdash (\lambda x{:}\tau_1.\ e) \in \tau_1 \rightarrow \tau_2}\ \text{(A-ARROW-I)}\right)^\wedge = \frac{s_1^\wedge}{\Gamma \overset{\wedge}{\vdash} (\lambda x{:}\tau_1.\ e) \in \tau_1 \rightarrow \tau_2}\ \text{(ARROW-I)}$$

$$\left(\frac{s_1 :: \Gamma \vdash e_1 \in \sigma_1 \qquad s_2 :: \Gamma \vdash e_2 \in \sigma_2}{\Gamma \vdash (e_1\ e_2) \in \bigwedge[\psi_i \mid (\phi_i \rightarrow \psi_i) \in \mathit{arrowbasis}_\Gamma(\sigma_1) \text{ and } c_i :: \Gamma \vdash \sigma_2 \leq \phi_i]}\ \text{(A-ARROW-E)}\right)^\wedge =$$

$$\frac{\dots \quad \dfrac{\dfrac{s_1^\wedge \qquad 5.3.1.2 :: \Gamma \overset{\wedge}{\vdash} \sigma_1 \leq \phi_i \rightarrow \psi_i}{\Gamma \overset{\wedge}{\vdash} e_1 \in \phi_i \rightarrow \psi_i}\ \text{(SUB)} \qquad \dfrac{s_2^\wedge \qquad c_i}{\Gamma \overset{\wedge}{\vdash} e_2 \in \phi_i}\ \text{(SUB)}}{\Gamma \overset{\wedge}{\vdash} (e_1\ e_2) \in \psi_i}\ \text{(ARROW-E)} \quad \dots}{\Gamma \overset{\wedge}{\vdash} (e_1\ e_2) \in \bigwedge[\psi_i \mid (\phi_i \rightarrow \psi_i) \in \mathit{arrowbasis}_\Gamma(\sigma_1) \text{ and } \Gamma \vdash \sigma_2 \leq \phi_i]}\ \text{(INTER-I)}$$

$$\left(\frac{s_1 :: \Gamma,\ \alpha{\leq}\tau_1 \vdash e \in \tau_2}{\Gamma \vdash (\Lambda\alpha{\leq}\tau_1.\ e) \in \forall\alpha{\leq}\tau_1.\ \tau_2}\ \text{(A-ALL-I)}\right)^\wedge = \frac{s_1^\wedge}{\Gamma \overset{\wedge}{\vdash} (\Lambda\alpha{\leq}\tau_1.\ e) \in \forall\alpha{\leq}\tau_1.\ \tau_2}\ \text{(ALL-I)}$$

$$\left(\frac{s_1 :: \Gamma \mathrel{\dot\vdash} e \in \sigma_1}{\Gamma \mathrel{\dot\vdash} e\;[\tau] \in \bigwedge[\{\tau/\alpha\}\psi_i \mid (\forall\alpha{\leq}\phi_i.\ \psi_i) \in \mathit{allbasis}_\Gamma(\sigma_1) \text{ and } c_i :: \Gamma \vdash \tau \leq \phi_i]}\ \text{(A-ALL-E)}\right)^{\wedge} =$$

$$\frac{\cdots \qquad \dfrac{\dfrac{s_1^{\wedge} \qquad 5.3.1.3 :: \Gamma \mathrel{\hat\vdash} \sigma_1 \leq \forall\alpha{\leq}\phi_i.\ \psi_i}{\Gamma \mathrel{\hat\vdash} e \in \forall\alpha{\leq}\phi_i.\ \psi_i}\ \text{(SUB)} \qquad c_i}{\Gamma \mathrel{\hat\vdash} e\;[\tau] \in \{\tau/\alpha\}\psi_i}\ \text{(ALL-E)} \qquad \cdots}{\Gamma \mathrel{\hat\vdash} e\;[\tau] \in \bigwedge[\{\tau/\alpha\}\psi_i \mid (\forall\alpha{\leq}\phi_i.\ \psi_i) \in \mathit{allbasis}_\Gamma(\sigma_1) \text{ and } \Gamma \vdash \tau \leq \phi_i]}\ \text{(INTER-I)}$$

$$\left(\frac{\text{for all } i,\ s_i :: \Gamma \mathrel{\dot\vdash} \{\sigma_i/\alpha\}e \in \tau_i}{\Gamma \mathrel{\dot\vdash} (\mathit{for}\ \alpha\ \mathit{in}\ \sigma_1..\sigma_n.\ e) \in \bigwedge[\tau_1..\tau_n]}\ \text{(A-FOR)}\right)^{\wedge} =$$

$$\frac{\cdots \qquad \dfrac{s_i^{\wedge} :: \Gamma \mathrel{\hat\vdash} \{\sigma_i/\alpha\}e \in \tau_i}{\Gamma \mathrel{\hat\vdash} (\mathit{for}\ \alpha\ \mathit{in}\ \sigma_1..\sigma_n.\ e) \in \tau_i}\ \text{(FOR)} \qquad \cdots}{\Gamma \mathrel{\hat\vdash} (\mathit{for}\ \alpha\ \mathit{in}\ \sigma_1..\sigma_n.\ e) \in \bigwedge[\tau_1..\tau_n]}\ \text{(INTER-I)}$$

5.3.2.5. Theorem: If $s :: \Gamma \mathrel{\dot\vdash} e \in \tau$, then $s^{\wedge} :: \Gamma \mathrel{\hat\vdash} e \in \tau$.

Proof: By induction on the structure of s. □

5.3.2.6. Theorem: [Minimal typing] If $\Gamma \mathrel{\dot\vdash} e \in \sigma$ and $\Gamma \mathrel{\hat\vdash} e \in \tau$, then $\Gamma \vdash \sigma \leq \tau$.

Proof: By induction on a derivation of $\Gamma \mathrel{\hat\vdash} e \in \tau$. Proceed by cases on the final rule.

Case VAR*:* $e \equiv x \qquad \tau \equiv \Gamma(x)$

Immediate by A-VAR.

Case ARROW-I*:* $e \equiv \lambda x{:}\tau_1.\ e' \qquad \Gamma,\ x{:}\tau_1 \mathrel{\hat\vdash} e' \in \tau_2 \qquad \tau \equiv \tau_1 \rightarrow \tau_2$

By the syntax-directedness of the type synthesis rules (5.3.2.2), the last rule in the derivation of $\Gamma \mathrel{\dot\vdash} e \in \sigma$ must be A-ARROW-I, so $\Gamma,\ x{:}\tau_1 \vdash e' \in \sigma_2$ and $\sigma \equiv \tau_1 \rightarrow \sigma_2$. By the induction hypothesis, $\Gamma \mathrel{\hat\vdash} \sigma_2 \leq \tau_2$. By SUB-REFL and SUB-ARROW, $\Gamma \mathrel{\hat\vdash} \tau_1 \rightarrow \sigma_2 \leq \tau_1 \rightarrow \tau_2$.

Case ARROW-E*:* $e \equiv e_1\ e_2 \qquad \Gamma \mathrel{\hat\vdash} e_1 \in \tau_1 \rightarrow \tau_2 \qquad \Gamma \mathrel{\hat\vdash} e_2 \in \tau_1 \qquad \tau \equiv \tau_2$

By the syntax-directedness of the type synthesis rules (5.3.2.2), $\Gamma \mathrel{\dot\vdash} e_1 \in \sigma_1$, $\Gamma \mathrel{\dot\vdash} e_2 \in \sigma_2$, and $\sigma \equiv \bigwedge[\psi_i \mid (\phi_i \rightarrow \psi_i) \in \mathit{arrowbasis}_\Gamma(\sigma_1) \text{ and } \Gamma \mathrel{\dot\vdash} \sigma_2 \leq \phi_i]$. By the equivalence of ordinary and syntax-directed subtyping (5.2.6.10), $\sigma \equiv \bigwedge[\psi_i \mid (\phi_i \rightarrow \psi_i) \in \mathit{arrowbasis}_\Gamma(\sigma_1) \text{ and } \Gamma \mathrel{\hat\vdash} \sigma_2 \leq \phi_i]$. By the induction hypothesis, $\Gamma \mathrel{\hat\vdash} \sigma_1 \leq \tau_1 \rightarrow \tau_2$ and $\Gamma \mathrel{\hat\vdash} \sigma_2 \leq \tau_1$. Since $\mathit{arrowbasis}_\Gamma(\sigma_1)$ is a finite basis for the arrow types above σ_1 (5.3.1.2), $\Gamma \mathrel{\hat\vdash} \bigwedge(\mathit{arrowbasis}_\Gamma(\sigma_1)) \leq \tau_1 \rightarrow \tau_2$. By the application lemma (5.3.1.4), $\Gamma \mathrel{\hat\vdash} \bigwedge[\psi_i \mid (\phi_i \rightarrow \psi_i) \in \mathit{arrowbasis}_\Gamma(\sigma_1) \text{ and } \Gamma \mathrel{\hat\vdash} \sigma_2 \leq \phi_i] \leq \tau_2$.

Case ALL-I*:* $e \equiv \Lambda\alpha{\leq}\tau_1.\ e' \qquad \Gamma,\ \alpha{\leq}\tau_1 \mathrel{\hat\vdash} e' \in \tau_2 \qquad \tau \equiv \forall\alpha{\leq}\tau_1.\ \tau_2$

By the syntax-directedness of the type synthesis rules (5.3.2.2), $\Gamma,\ \alpha{\leq}\tau_1 \mathrel{\dot\vdash} e' \in \sigma_2$ and $\sigma \equiv \forall\alpha{\leq}\tau_1.\ \sigma_2$. By the induction hypothesis, $\Gamma,\ \alpha{\leq}\tau_1 \mathrel{\hat\vdash} \sigma_2 \leq \tau_2$. By SUB-REFL and SUB-ALL, $\Gamma \mathrel{\hat\vdash} (\forall\alpha{\leq}\tau_1.\ \sigma_2) \leq (\forall\alpha{\leq}\tau_1.\ \tau_2)$.

Case ALL-E*:* $e \equiv e'[\tau']$ $\quad \Gamma \vdash^{\wedge} e' \in \forall\alpha{\leq}\tau_1.\ \tau_2$ $\quad \Gamma \vdash^{\wedge} \tau' \leq \tau_1$ $\quad \tau \equiv \{\tau'/\alpha\}\tau_2$

By the syntax-directedness of the type synthesis rules (5.3.2.2), $\Gamma \dot{\vdash} e' \in \sigma_1$ and $\sigma \equiv \bigwedge[\{\tau'/\alpha\}\psi_i \mid (\forall\alpha{\leq}\phi_i.\ \psi_i) \in \mathit{allbasis}_\Gamma(\sigma_1)$ and $\Gamma \dot{\vdash} \tau' \leq \phi_i]$. By the equivalence of ordinary and syntax-directed subtyping (5.2.6.10), $\sigma \equiv \bigwedge[\{\tau'/\alpha\}\psi_i \mid (\forall\alpha{\leq}\phi_i.\ \psi_i) \in \mathit{allbasis}_\Gamma(\sigma_1)$ and $\Gamma \vdash^{\wedge} \tau' \leq \phi_i]$. By the induction hypothesis, $\Gamma \vdash^{\wedge} \sigma_1 \leq \forall\alpha{\leq}\tau_1.\ \tau_2$. Since $\mathit{allbasis}_\Gamma(\sigma_1)$ is a finite basis for the polymorphic types above σ_1 (5.3.1.3), $\Gamma \vdash^{\wedge} \bigwedge(\mathit{allbasis}_\Gamma(\sigma_1)) \leq \forall\alpha{\leq}\tau_1.\ \tau_2$. By the type application lemma (5.3.1.5), $\Gamma \vdash^{\wedge} \bigwedge[\{\tau'/\alpha\}\psi_i \mid (\forall\alpha{\leq}\phi_i.\ \psi_i) \in \mathit{allbasis}_\Gamma(\sigma_1)$ and $\Gamma \vdash \tau' \leq \phi_i] \leq \{\tau'/\alpha\}\tau_2$.

Case FOR*:* $e \equiv \mathit{for}\ \alpha\ \mathit{in}\ \sigma_1..\sigma_n.\ e'$ $\quad \Gamma \vdash^{\wedge} \{\sigma_i/\alpha\}e \in \tau_i$ $\quad \tau \equiv \tau_i$

By the syntax-directedness of the type synthesis rules (5.3.2.2), for all j, $\Gamma \dot{\vdash} \{\sigma_j/\alpha\}e \in \phi_j$ and $\sigma \equiv \bigwedge[\phi_1..\phi_n]$. By the induction hypothesis, $\Gamma \vdash^{\wedge} \phi_i \leq \tau_i$. By SUB-INTER-LB and SUB-TRANS, $\Gamma \vdash^{\wedge} \bigwedge[\phi_1..\phi_n] \leq \tau_i$.

Case INTER-I*:* for all i, $\Gamma \vdash^{\wedge} e \in \tau_i$ $\quad \tau \equiv \bigwedge[\tau_1..\tau_n]$

By the induction hypothesis, for all i, $\Gamma \vdash^{\wedge} \sigma \leq \tau_i$. By SUB-INTER-G, $\Gamma \vdash^{\wedge} \sigma \leq \bigwedge[\tau_1..\tau_n]$.

Case SUB*:* $\Gamma \vdash^{\wedge} e \in \tau_1$ $\quad \Gamma \vdash^{\wedge} \tau_1 \leq \tau_2$ $\quad \tau \equiv \tau_2$

By the induction hypothesis, $\Gamma \vdash^{\wedge} \sigma \leq \tau_1$. By SUB-TRANS, $\Gamma \vdash^{\wedge} \sigma \leq \tau_2$. □

5.3.3. *Conservativity* $F_\wedge$ was described as essentially the union of the two simpler calculi $\lambda_\wedge$ and $F_\leq$. We can gauge the accuracy of this characterization by checking whether the features of the component calculi operate "orthogonally" so that each component system can be thought of as a restriction of $F_\wedge$— i.e., by asking whether $F_\wedge$ is a conservative extension of $\lambda_\wedge$ and of $F_\leq$.

5.3.3.1. Definition: Let C and E be two calculi and $E(\text{—}) \in C \to E$ an injective mapping from C statements to E statements. $E(\text{—})$ is said to be an *embedding* of C into E if, for every C statement J, J is derivable in C iff $E(J)$ is derivable in E.

Typically, $E(\text{—})$ is just an identity injection. For instance, this is the case for the embedding of $\lambda_\wedge$ into $F_\wedge$.

5.3.3.2. Definition: If the identity injection is an embedding of C into E, then E is said to be a *conservative extension* of C.

5.3.3.3. Theorem: Let σ and τ be $\lambda_\wedge$ types, Γ an $\lambda_\wedge$ context, and e an $\lambda_\wedge$ expression. Assume that the primitive subtype relation of $\lambda_\wedge$ is encoded as a context Γ_P (cf. 3.3.2.2). Then:

1 $\Gamma_P, \Gamma \vdash^{\wedge} \sigma \leq \tau$ iff $\Gamma \vdash^{\lambda\wedge} \sigma \leq \tau$.

2 $\Gamma_P, \Gamma \vdash^{\wedge} e \in \tau$ iff $\Gamma \vdash^{\lambda\wedge} e \in \tau$.

Proof: See Appendix A. □

The mapping from the other subsystem, $F_\leq$, into $F_\wedge$ must take the type *Top* into $\top$, and it is here that it fails to be an embedding (as we might expect from the discussion in Section 3.3). For example, the subtyping statement

$$\vdash\ \mathit{Top}\ \leq\ \forall\alpha{\leq}\mathit{Top}.\ \mathit{Top}$$

is derivable in $F_\wedge$(reading *Top* as $\top$), but not in $F_\leq$.

5.3.3.4. Conjecture: Since *Top* must be mapped to a maximal type by any embedding function from $F_\leq$ to $F_\wedge$ and $\top$ is the only such type (up to equivalence), it appears that there is *no* embedding of $F_\leq$ into $F_\wedge$.

5.3.3.5. Conjecture: By replacing *Top* with $\top$ in $F_\leq$ and adding appropriate distributivity laws to the subtyping relation and a $\top$-introduction rule to the typing relation, we can construct a system that can be embedded into $F_\wedge$ (indeed, such that $F_\wedge$ extends it conservatively). But this system is only a technical curiosity: it has most of the problematic features of $F_\wedge$ (the distributivity laws in particular) and is much less expressive.

6. Semantics

This section presents two results about the semantics of $F_\wedge$. Section 6.1 gives a simple untyped semantics for $F_\wedge$ based on Bruce and Longo's partial equivalence relation model for $F_\leq$ (1990). Section 6.2 discusses a negative technical result — the nonexistence of syntactic least upper bounds — with some serious implications for the difficulty of constructing a typed model for $F_\wedge$ in which the subtype relation is interpreted by semantic coercion functions.

6.1. *Untyped Semantics*

One of the simplest styles of semantics for typed λ-calculi is based on partial equivalence relations (PERs). A model in this style is essentially untyped: terms are interpreted by erasing all type information and interpreting the resulting pure λ-term as an element of the model. A type in this setting is just a subset of the model along with an appropriate notion of equivalence of elements. Coercions between types are interpreted as inclusion of PERs. The PER model given here for $F_\wedge$ is based on Bruce and Longo's model for $F_\leq$ (1990). However, the full generality of Bruce and Longo's construction, involving the category of ω-sets, is not required.

The usual interpretation of a quantified type $\forall\alpha.\ \tau$ in a second-order PER model is the PER-indexed intersection of all instances of τ. Bruce and Longo showed how to extend this definition to interpret a bounded quantifier $\forall\alpha{\leq}\sigma.\ \tau$ as the intersection of all the instances of τ where α is interpreted as a sub-PER of the interpretation of σ. This intuition also serves for intersection types: $\bigwedge[\tau_1..\tau_n]$ is interpreted as the intersection of the PERs interpreting each of the τ_i's.

We need to make one significant departure here from PER models of $F_\leq$: instead of allowing the elements of our PERs to be drawn from the carrier of an arbitrary partial combinatory algebra $\mathcal{D}$, we require that $\mathcal{D}$ be a *total* combinatory algebra. This restriction is needed to validate nullary instances of the distributive law SUB-DIST-IA, which have the form $\Gamma \vdash \top \leq \sigma{\rightarrow}\top$. To see why, let $\sigma \equiv \top$. The empty intersection $\top$ is interpreted by the everywhere-defined PER, i.e., $[\![\top]\!]$ relates every m to itself. To validate the distributivity law, it must therefore be the case that $[\![\top{\rightarrow}\top]\!]$ relates every element to itself. But this will only be true if the application of any element to any other element is defined. This observation is due to QingMing Ma [personal communication, 1991].

The notation and fundamental definitions used in this section are based on papers of Bruce and Longo (1990), Freyd, Mulry, Rosolini, and Scott (1990), and others. A good basic reference for PER models of second-order λ-calculi is (Mitchell, 1990); also see (Bruce *et al.*, 1990) for more general discussion of second-order models and (Barendregt, 1984; Hindley & Seldin, 1986; Gunter, 1992) for general discussion of combinatory models.

6.1.1. *Background*

6.1.1.1. Definition: A *total combinatory algebra* is a tuple $\mathcal{D} = \langle D, \cdot, k, s\rangle$ comprising

— a set D of *elements*,

— an *application function* $\cdot$ with type $D{\to}(D{\to}D)$,

— distinguished elements k, $s \in D$,

such that, for all d_1, d_2, $d_3 \in D$,

$$\begin{array}{l} k \cdot d_1 \cdot d_2 = d_1 \\ s \cdot d_1 \cdot d_2 \cdot d_3 = (d_1 \cdot d_3) \cdot (d_2 \cdot d_3). \end{array}$$

Throughout this section, we work with a fixed, but unspecified, total combinatory algebra. (Cf. (Scott, 1976) for examples.)

The set of *pure λ-terms* is:

$$M \quad ::= \quad x \quad | \quad \lambda x.\ M \quad | \quad M_1\ M_2$$

The set of *combinator terms* is:

$$C \quad ::= \quad x \quad | \quad C_1\, C_2 \quad | \quad K \quad | \quad S$$

The *bracket abstraction* of a combinator term C with respect to a variable x, written $\lambda^\star x.\ C$, is:

$$\begin{array}{lll} \lambda^\star x.\ C & = K\ C & \text{when } x \notin FV(C) \\ \lambda^\star x.\ x & = S\ K\ K & \\ \lambda^\star x.\ C_1\, C_2 & = S\,(\lambda^\star x.\ C_1)\,(\lambda^\star x.\ C_2) & \text{when } x \in FV(C_1\, C_2) \end{array}$$

The *combinator translation* of a pure λ-term M, written $|M|$, is defined as follows:

$$\begin{array}{ll} |x| & = x \\ |\lambda x.\ M| & = \lambda^\star x.\ |M| \\ |M_1\, M_2| & = |M_1|\ |M_2| \end{array}$$

An *environment* η is a finite function from type variables to PERs (defined below) and term variables to elements of D. When $x \notin dom(\eta)$, we write $\eta[x{\leftarrow}d]$ for the environment that maps x to d and agrees with η everywhere else; $\eta[\alpha{\leftarrow}A]$ is defined similarly. We write $\eta\backslash x$ for the environment like η except that $\eta(x)$ is undefined; $\eta\backslash\alpha$ similarly. We say that η' *extends* η when $dom(\eta) \subseteq dom(\eta')$ and η and η' agree on $dom(\eta)$.

Let C be a combinator term and η an environment such that $FV(C) \subseteq dom(\eta)$. Then the *interpretation* of C under η, written $[\![C]\!]_\eta$, is defined as follows:

$$
\begin{array}{ll}
[\![x]\!]_\eta & = \eta(x) \\
[\![C_1\, C_2]\!]_\eta & = [\![C_1]\!]_\eta \cdot [\![C_2]\!]_\eta \\
[\![K]\!]_\eta & = k \\
[\![S]\!]_\eta & = s
\end{array}
$$

6.1.1.2. Lemma: If η' extends η and $FV(C) \subseteq dom(\eta)$, then $[\![C]\!]_\eta = [\![C]\!]_{\eta'}$.

Proof: Straightforward induction on C. □

6.1.1.3. Lemma: $[\![\lambda^\star x.\ C]\!]_\eta \cdot m = [\![C]\!]_{\eta[x \leftarrow m]}$.

Proof: See Appendix A. □

6.1.1.4. Definition: A *partial equivalence relation* (PER) on $\mathcal{D}$ is a symmetric and transitive relation A on D. We write $m\ \{A\}\ n$ when A relates m and n. The *domain* of A, written $dom(A)$, is the set $\{n \mid n\ \{A\}\ n\}$. When $n \in dom(A)$, the *equivalence class* of n in A, written $[n]_A$, is the set $\{m \mid n\ \{A\}\ m\}$. Note that $m\ \{A\}\ n$ implies $m \in dom(A)$. When A and B are relations, $A{\rightarrow}B$ is the relation defined by

$$m\ \{A{\rightarrow}B\}\ n \quad \text{iff} \quad \text{for all } p, q \in D,\ \ p\ \{A\}\ q \text{ implies } m{\cdot}p\ \{B\}\ n{\cdot}q.$$

6.1.1.5. Lemma: $A{\rightarrow}B$ is a PER when A and B are PERs.

Proof: Standard. □

6.1.1.6. Definition: A is a *subrelation* of B, written $A \subseteq B$, iff $m\ \{A\}\ n$ implies $m\ \{B\}\ n$ for all $m, n \in D$.

6.1.1.7. Definition: Let $\{A_i\}_{i \in I}$ be a set of relations indexed by a set I. Then $\bigcap_{i \in I} A_i$ is the relation defined by

$$m\ \{\textstyle\bigcap_{i \in I} A_i\}\ n \quad \text{iff} \quad \text{for every } i,\ m\ \{A_i\}\ n.$$

6.1.1.8. Lemma: $\bigcap_{i \in I} A_i$ is a PER when all the A_i's are PERs.

Proof: Straightforward. □

6.1.2. *PER Interpretation of $F_\wedge$*

6.1.2.1. Definition: The *erasure* of an $F_\wedge$ term e, written $erase(e)$, is the pure λ-term defined as follows:

$$
\begin{array}{lcl}
erase(x) & = & x \\
erase(\lambda x{:}\tau.\ e) & = & \lambda x.\ erase(e) \\
erase(e_1\ e_2) & = & erase(e_1)\ erase(e_2) \\
erase(\Lambda\alpha \leq \tau.\ e) & = & erase(e) \\
erase(e\ [\tau]) & = & erase(e) \\
erase(\mathit{for}\ \alpha\ \mathit{in}\ \sigma_1..\sigma_n.\ e) & = & erase(e)
\end{array}
$$

6.1.2.2. Definition: Let η be an environment and e an expression such that $FV(e) \subseteq dom(\eta)$. Then the *interpretation* of e under η, written $[\![e]\!]_\eta$, is $[\![|erase(e)|]\!]_\eta$.

6.1.2.3. Lemma: $[\![\lambda x{:}\tau.\ e]\!]_\eta = [\![\lambda^\star x.\ |erase(e)|]\!]_\eta$.

Proof: Straightforward. □

6.1.2.4. Definition: Let η be an environment and τ a type expression such that $FTV(\tau) \subseteq dom(\eta)$. The *interpretation* of τ under η, written $[\![\tau]\!]_\eta$, is the PER defined as follows:

$$\begin{array}{lcll} [\![\alpha]\!]_\eta & = & \eta(\alpha) & \\ [\![\tau_1 \to \tau_2]\!]_\eta & = & [\![\tau_1]\!]_\eta \to [\![\tau_2]\!]_\eta & \\ [\![\forall\alpha{\leq}\tau_1.\ \tau_2]\!]_\eta & = & \bigcap_{A \subseteq [\![\tau_1]\!]_\eta} [\![\tau_2]\!]_{\eta[\alpha\leftarrow A]} & \text{where } A \text{ is a PER} \\ [\![\bigwedge[\tau_1..\tau_n]]\!]_\eta & = & \bigcap_{1\leq i\leq n} [\![\tau_i]\!]_\eta & \end{array}$$

6.1.2.5. Definition: An environment η *satisfies* a context Γ, written $\eta \models \Gamma$, if $dom(\eta) = dom(\Gamma)$ and

1 $\Gamma \equiv \{\}$, or

2 $\Gamma \equiv \Gamma_1,\ x : \tau$, where $\eta\backslash x$ satisfies Γ_1 and $\eta(x) \in dom([\![\tau]\!]_{\eta\backslash x})$, or

3 $\Gamma \equiv \Gamma_1,\ \alpha{\leq}\tau$, where $\eta\backslash\alpha$ satisfies Γ_1 and $\eta(\alpha) \subseteq [\![\tau]\!]_{\eta\backslash\alpha}$.

6.1.2.6. Lemma: If η' extends η and $FTV(\tau) \subseteq dom(\eta)$, then $[\![\tau]\!]_\eta = [\![\tau]\!]_{\eta'}$.

Proof: Straightforward. □

6.1.2.7. Lemma: (Soundness of subtyping) If $\Gamma \vdash \sigma \leq \tau$ and $\eta \models \Gamma$, then $[\![\sigma]\!]_\eta \subseteq [\![\tau]\!]_\eta$.

Proof: See Appendix A. □

6.1.2.8. Lemma:

1 $[\![e_1\, e_2]\!]_\eta = [\![e_1]\!]_\eta \cdot [\![e_2]\!]_\eta$.

2 $erase(\{\sigma/\alpha\}e) = erase(e)$.

3 $[\![\tau]\!]_{\eta[\alpha\leftarrow[\![\sigma]\!]_\eta]} = [\![\{\sigma/\alpha\}\tau]\!]_\eta$.

4 $[\![C]\!]_{\eta[x\leftarrow[\![C']\!]_\eta]} = [\![\{C'/x\}C]\!]_\eta$.

5 $\{|erase(v)|/x\}|erase(f)| = |erase(\{v/x\}f)|$.

Proof: Straightforward. □

6.1.2.9. Lemma: If

$$\begin{array}{l} \eta_1 \models \Gamma \\ \eta_2 \models \Gamma \\ \forall\alpha \in dom(\Gamma).\ \eta_1(\alpha) = \eta_2(\alpha) = \eta(\alpha) \\ \forall x \in dom(\Gamma).\ \eta_1(x)\ \{[\![\Gamma(x)]\!]_\eta\}\ \eta_2(x) \\ \Gamma\ \vdash\ e\ \in\ \tau, \end{array}$$

then

$$[\![e]\!]_{\eta_1}\ \{[\![\tau]\!]_\eta\}\ [\![e]\!]_{\eta_2}.$$

(Here η is just a convenient name for the portions of η_1 and η_2 dealing with type variables, which must be identical.)

Proof: See Appendix A. □

6.1.2.10. Corollary: (Soundness of typing) If $\Gamma \vdash e \in \tau$ and $\eta \models \Gamma$, then $[\![e]\!]_\eta \in dom([\![\tau]\!]_\eta)$.
Proof: Take $\eta_1 = \eta_2 = \eta$. □

6.1.2.11. Remark: If we now went on to study an equational theory for terms, it would be convenient to redefine the denotation of a typed term as an equivalence class in the model, rather than as an element of the underlying combinatory algebra, as we have done so far. When $\Gamma \vdash e \in T$, let

$$[\![e]\!]^T_\eta = \{n \mid n\ \{[\![T]\!]_\eta\}\ [\![e]\!]_\eta\}$$

This definition, which is justified by Corollary 6.1.2.10, is preferable to the one we have used so far because it satisfies extensionality.

6.2. *Nonexistence of Least Upper Bounds*

One important question about the order-theoretic properties of any calculus with subtyping is the existence or nonexistence of *least upper bounds* (lubs) for finite sets of types. When they are present, lubs often greatly simplify the presentations of both semantic and proof-theoretic arguments; for example, Reynolds's model construction for Forsythe depends on the existence and special properties of lubs. Unfortunately, like its component system $F_\leq$ (though not for exactly the same reason), $F_\wedge$ does *not* have a lub for every finite set of types.

To simplify the discussion, we consider only lubs of pairs of types. The fact that a calculus of intersection types may be formulated in terms of an n-ary meet constructor, as we have done here, or, equivalently, in terms of $\top$ and binary meets, implies that we may make this simplification without loss of generality.

6.2.1. Definition: Let σ and τ be types, both closed under a context Γ. Then a *least upper bound* of σ and τ under Γ is a supertype of both σ and τ and a subtype of every common supertype of σ and τ — that is, a type θ such that:

$$\Gamma \vdash \sigma \leq \theta$$
$$\Gamma \vdash \tau \leq \theta$$
$$\Gamma \vdash \sigma \leq \phi \quad \text{and} \quad \Gamma \vdash \tau \leq \phi \quad \text{imply} \quad \Gamma \vdash \theta \leq \phi.$$

In systems with intersection types, it is simplest to define least upper bounds for canonical types (cf. Section 5.2.1) and then transfer the definition to ordinary types. Here is Reynolds's definition of lubs for the canonical formulation of first-order intersection types:

6.2.2. Definition: Assume that we are given a partial function $\sqcup_P$ that yielding a least upper bound for every pair of primitive types with any upper bound. That is:

$$\begin{array}{llll} \text{if} & (\rho_1 \sqcup_P \rho_2)\downarrow & \text{then} & \rho_1 \leq_P (\rho_1 \sqcup_P \rho_2) \\ & & & \rho_2 \leq_P (\rho_1 \sqcup_P \rho_2) \\ & & & \rho_1 \leq_P \rho' \quad \text{and} \quad \rho_2 \leq_P \rho' \quad \text{imply} \quad (\rho_1 \sqcup_P \rho_2) \leq_P \rho' \\ \text{if} & (\rho_1 \sqcup_P \rho_2)\uparrow & \text{then} & \text{there is no } \rho' \text{ such that } \rho_1 \leq_P \rho' \text{ and } \rho_2 \leq_P \rho'. \end{array}$$

6.2.3. Definition: Let k and i be canonical $\lambda_\wedge$ types. Then the distinguished least upper bound of k and i, written $k \sqcup i$, is defined by the following function (partial on individual canonical types and total on composite canonical types):

$$
\begin{array}{lcl}
K \sqcup I & = & \bigwedge[\kappa \sqcup \iota \mid \kappa \in K \text{ and } \iota \in I \text{ and } (\kappa \sqcup \iota)\downarrow] \\
\rho_1 \sqcup \rho_2 & = & \rho_1 \sqcup_P \rho_2 \\
(K \to \kappa) \sqcup (I \to \iota) & = & (K \cup I) \to (\iota \sqcup \kappa) \\
(K \to \kappa) \sqcup \rho & = & \uparrow \\
\rho \sqcup (I \to \iota) & = & \uparrow .
\end{array}
$$

(Recall that $K \cup I$ is shorthand for the intersection of all the elements of K and I.)

6.2.4. Fact: [Reynolds]

1 If $\kappa \sqcup \iota$ is defined, then it is a least upper bound of κ and ι. If $\kappa \sqcup \iota$ is undefined, then κ and ι have no common upper bounds.

2 $K \sqcup I$ is a least upper bound of K and I.

The existence of lubs for canonical types is easily shown to be equivalent to the existence of lubs for ordinary types, using the first-order analog of Theorem 5.2.5.6.

In his Ph.D. thesis, Ghelli (1990) observed that $F_{\leq}$ possesses neither least upper bounds nor greatest lower bounds.

6.2.5. Definition: A pair of types σ and τ is *downward compatible* if there is some type that is a subtype of both σ and τ.

6.2.6. Fact: (Ghelli, 1990, p. 92) There exists a pair of downward-compatible $F_{\leq}$ types σ and τ with no greatest lower bound.

Proof: Consider the context

$$\Gamma = \alpha \leq Top,\ \beta \leq Top,\ \alpha' \leq \alpha,\ \beta' \leq \beta$$

and the types

$$
\begin{array}{lcl}
\sigma & = & \forall\gamma\leq\alpha\to\beta.\ \alpha\to\beta \\
\tau & = & \forall\gamma\leq\alpha'\to\beta'.\ \alpha'\to\beta'.
\end{array}
$$

Then both

$$\forall\gamma\leq\alpha'\to\beta.\ \alpha\to\beta'$$

and

$$\forall\gamma\leq\alpha'\to\beta.\ \gamma$$

are lower bounds for σ and τ, but these two types have no common supertype that is also a subtype of σ and τ. □

6.2.7. Fact: [Ghelli] There is a pair of $F_{\leq}$ types with no least upper bound.

Proof: Consider $\sigma \to Top$ and $\tau \to Top$. □

Since $F_{\wedge}$, by definition, possesses greatest lower bounds for every pair of types, we might hope that lubs would also be recovered in $F_{\wedge}$. Unfortunately, this is not the case.

To give a flavor of the argument, consider the individual canonical types

$$
\begin{array}{lcl}
\kappa & \equiv & \forall\alpha{\leq}\bigwedge[].\ \forall\beta{\leq}\bigwedge[].\ \alpha \\
\iota & \equiv & \forall\alpha{\leq}\bigwedge[].\ \forall\beta{\leq}\bigwedge[].\ \beta \\
\mu_1 & \equiv & \forall\alpha{\leq}\bigwedge[].\ \forall\beta{\leq}\bigwedge[\alpha].\ \alpha \\
\mu_2 & \equiv & \forall\alpha{\leq}\bigwedge[\nu].\ \forall\beta{\leq}\bigwedge[\nu].\ \nu,
\end{array}
$$

where ν is any closed individual canonical type with the property that $\alpha{\leq}\nu \not\vdash \nu \leq \alpha$. (For example, take $\nu \equiv \forall\gamma{\leq}\bigwedge[].\ \gamma$.) Then it is easy to check that the following subtype relations hold in the empty context:

$$
\begin{tikzcd}
\mu_1 & \mu_2 \\
\kappa \arrow[u] \arrow[ur] & \iota \arrow[u] \arrow[ul]
\end{tikzcd}
$$

Note, however, that $\not\vdash \mu_1 \leq \mu_2$.

Now, assume (for a contradiction) that κ and ι have some least upper bound; call it λ. Then by the syntax-directedness of canonical subtyping (5.2.4.1) and the fact that λ is a supertype of κ, λ must have the form

$$\lambda \equiv \forall\alpha{\leq}L_1.\ \forall\beta{\leq}L_2.\ \lambda_3.$$

By syntax-directedness again and the fact that $\vdash \lambda \leq \mu_1$ (since μ_1 is a common upper bound of κ and ι),

$\vdash\ \bigwedge[]\ \leq\ L_1$
i.e. $L_1 \equiv \bigwedge[]$

$\alpha{\leq}\bigwedge[]\ \vdash\ \bigwedge[\alpha]\ \leq\ L_2$
i.e. $\lambda_2\ \in\ L_2$ implies $\alpha{\leq}\bigwedge[]\ \vdash\ \alpha\ \leq\ \lambda_2$
i.e. $\lambda_2\ \in\ L_2$ implies $\lambda_2 \equiv \alpha$
i.e. $L_2 \equiv \bigwedge[]$ or $L_2 \equiv \bigwedge[\alpha]$ (up to equivalence) ,

and if $L_2 \equiv \bigwedge[\alpha]$ then

$\alpha{\leq}\bigwedge[], \beta{\leq}\bigwedge[\alpha]\ \vdash\ \lambda_3\ \leq\ \alpha$
i.e. $\lambda_3 \equiv \alpha$ or $\lambda_3 \equiv \beta$,

while if $L_2 \equiv \bigwedge[]$, then $\lambda_3 \equiv \alpha$.

Using the assumption that $\vdash \kappa \leq \lambda$, we may eliminate the case $\lambda_3 \equiv \beta$. Then, using $\vdash \iota \leq \lambda$, we may eliminate the case $L_2 \equiv \bigwedge[]$. In short, if κ and ι have any lub then it is equivalent to μ_1, which must therefore also be a lub. But μ_1 is not a subtype of μ_2, which is the common upper bound of κ and ι; so μ_1 is *not* a lub of κ and ι. This contradicts our assumption.

To show that composite canonical types lack lubs, we actually need to show something stronger about individual canonical types: that they do not even possess complete finite sets of upper bounds.

6.2.8. Definition: Let σ and τ be types, both closed under Γ. Then a complete finite set of upper bounds for σ and τ under Γ is a finite set $T \equiv \{\theta_1..\theta_n\}$ such that:

1 $\Gamma \vdash \sigma \leq \theta_i$ and $\Gamma \vdash \tau \leq \theta_i$ for each θ_i;

2 if ϕ is a type such that $\Gamma \vdash \sigma \leq \phi$ and $\Gamma \vdash \tau \leq \phi$, then there is some θ_i such that $\Gamma \vdash \theta_i \leq \phi$.

6.2.9. Definition: Define the following infinite series of individual canonical types:

$$\begin{array}{lll} \nu_0 & \equiv & \forall\alpha{\leq}\bigwedge[].\ \alpha \\ \nu_{n+1} & \equiv & \forall\alpha{\leq}\bigwedge[].\ \nu_n. \end{array}$$

6.2.10. Lemma: If $\Delta \vdash \nu_i \sim \gamma$ for some Δ and γ, then $\gamma \equiv \nu_i$.

Proof: By induction on i.

Case: $i = 0$

Since $\Delta \vdash \nu_0 \leq \gamma$, syntax-directedness (5.2.4.1) gives $\gamma \equiv \forall\alpha{\leq}I_1.\ \gamma_2$ and $\Delta,\ \alpha{\leq}I_1 \vdash \alpha \leq \gamma_2$. From $\Delta \vdash \gamma \leq \nu_0$, syntax-directedness gives $\Delta \vdash \bigwedge[] \leq I_1$, hence $I_1 \equiv \bigwedge[]$. Performing this substitution, we have $\Delta,\ \alpha{\leq}\bigwedge[] \vdash \alpha \leq \gamma_2$, hence (by syntax-directedness again) $\gamma_2 \equiv \alpha$.

Case: $i = n + 1$

By syntax-directedness, $\Delta \vdash \nu_{n+1} \leq \gamma$ gives

$$\begin{array}{l} \gamma \equiv \forall\alpha{\leq}I_1.\ \gamma_2 \\ \Delta,\ \alpha{\leq}I_1 \ \vdash\ \nu_n \ \leq\ \gamma_2. \end{array}$$

Using syntax-directedness on $\Delta \vdash \gamma \leq \nu_{n+1}$, we also have

$$\begin{array}{l} \Delta \ \vdash\ \bigwedge[] \ \leq\ I \\ \text{i.e. } I \equiv \bigwedge[] \\ \Delta,\ \alpha{\leq}\bigwedge[] \ \vdash\ \gamma_2 \ \leq\ \nu_n. \end{array}$$

By the induction hypothesis, $\gamma_2 \equiv \nu_n$, so $\gamma \equiv \forall\alpha{\leq}\bigwedge[].\ \nu_n$, which is just ν_{n+1}. □

6.2.11. Lemma: There exists a pair of individual canonical types in $F_\wedge$ with no complete finite set of upper bounds.

Proof: Assume, for a contradiction, that $B \equiv \{\lambda_1..\lambda_n\}$ is a complete finite set of upper bounds for the types

$$\begin{array}{lll} \kappa & \equiv & \forall\alpha{\leq}\bigwedge[].\ \forall\beta{\leq}\bigwedge[].\ \alpha \\ \iota & \equiv & \forall\alpha{\leq}\bigwedge[].\ \forall\beta{\leq}\bigwedge[].\ \beta \end{array}$$

and let

$$\mu_i \equiv \forall\alpha{\leq}\bigwedge[\nu_i].\ \forall\beta{\leq}\bigwedge[\nu_i].\ \nu_i$$

for every natural number i. Note that each μ_i is a common supertype of κ and ι. Also, since there are more μ's than λ's, we can choose some $\lambda \in B$ and some μ_i and μ_j (with $i \neq j$) such that $\vdash \lambda \leq \mu_i$ and $\vdash \lambda \leq \mu_j$.

From $\vdash \kappa \leq \lambda$ and $\vdash \iota \leq \lambda$, syntax-directedness gives

$$\begin{array}{l} \lambda \equiv \forall\alpha{\leq}L_1.\ \forall\beta{\leq}L_2.\ \gamma_3 \\ \alpha{\leq}L_1,\ \beta{\leq}L_2 \ \vdash\ \alpha \ \leq\ \gamma_3 \\ \alpha{\leq}L_1,\ \beta{\leq}L_2 \ \vdash\ \beta \ \leq\ \gamma_3. \end{array}$$

Since $\vdash \lambda \leq \mu_i$, syntax-directedness again yields

$$\begin{array}{l} \vdash \bigwedge[\nu_i] \leq L_1 \\ \alpha{\leq}\bigwedge[\nu_i] \vdash \bigwedge[\nu_i] \leq L_2 \\ \alpha{\leq}\bigwedge[\nu_i], \beta{\leq}\bigwedge[\nu_i] \vdash \gamma_3 \leq \nu_i. \end{array}$$

By canonical narrowing (5.2.3.2),

$$\begin{array}{l} \alpha{\leq}\bigwedge[\nu_i], \beta{\leq}L_2 \vdash \alpha \leq \gamma_3 \\ \alpha{\leq}\bigwedge[\nu_i], \beta{\leq}L_2 \vdash \beta \leq \gamma_3, \end{array}$$

and again

$$\begin{array}{l} \alpha{\leq}\bigwedge[\nu_i], \beta{\leq}\bigwedge[\nu_i] \vdash \alpha \leq \gamma_3 \\ \alpha{\leq}\bigwedge[\nu_i], \beta{\leq}\bigwedge[\nu_i] \vdash \beta \leq \gamma_3. \end{array}$$

Now by syntax-directedness,

$$\begin{array}{lll} \gamma_3 \equiv \alpha & \text{or} & \alpha{\leq}\bigwedge[\nu_i], \beta{\leq}\bigwedge[\nu_i] \vdash \bigwedge[\nu_i] \leq \bigwedge[\gamma_3] \\ \gamma_3 \equiv \beta & \text{or} & \alpha{\leq}\bigwedge[\nu_i], \beta{\leq}\bigwedge[\nu_i] \vdash \bigwedge[\nu_i] \leq \bigwedge[\gamma_3]. \end{array}$$

Since $\lambda \equiv \alpha$ and $\lambda \equiv \beta$ cannot both be true, we have $\alpha{\leq}\bigwedge[\nu_i], \beta{\leq}\bigwedge[\nu_i] \vdash \bigwedge[\nu_i] \leq \bigwedge[\gamma_3]$, i.e. (by syntax-directedness),

$$\alpha{\leq}\bigwedge[\nu_i], \beta{\leq}\bigwedge[\nu_i] \vdash \nu_i \leq \gamma_3.$$

Combining this with the type inclusion in the opposite direction (which we derived above), we get

$$\alpha{\leq}\bigwedge[\nu_i], \beta{\leq}\bigwedge[\nu_i] \vdash \nu_i \sim \gamma_3.$$

So, by Lemma 6.2.10, $\gamma_3 \equiv \nu_i$.

But starting from $\vdash \lambda \leq \mu_j$ and reasoning analogously, we can also obtain $\gamma_3 \equiv \nu_j$. Since $\nu_i \not\equiv \nu_j$, this is a contradiction. Our assumption that B is a complete finite set of upper bounds for κ and ι must therefore be false. □

Clearly, if κ and ι have no complete finite set of upper bounds, then the composite canonical types $\bigwedge[\kappa]$ and $\bigwedge[\iota]$ have no lub. As in the first-order case, by Theorem 5.2.5.6, $F_\wedge$ has lubs iff its canonical formulation does, so this counterexample for the canonical system amounts to a proof of the nonexistence of lubs for the original formulation of $F_\wedge$. (Also, in ordinary $F_\wedge$ a complete finite set of upper bounds can always be conjoined to form a single least upper bound, so the nonexistence of lubs is equivalent to the nonexistence of complete finite sets of upper bounds for ordinary types.)

The most immediate implication of the nonexistence of least upper bounds is that standard techniques developed by Reynolds (1991) for constructing and analyzing models of first-order intersection types will not generalize straightforwardly to $F_\wedge$.

Reynolds's model construction proceeds as follows. First, the set of canonical type expressions is defined as the limit of a series formed by beginning with the primitives and, at each stage, first closing under the $\rightarrow$ constructor and then forming all finite meets of the resulting set. The semantics of types is defined by induction on the same series of sets of types: the interpretation of a type τ at stage $n+1$ is defined in terms of the interpretations of the components of τ at stage n. The intended interpretation of an intersection $\sigma \wedge \tau$ in a typed semantics is the limit of a diagram containing the interpretations of σ and τ and all their common supertypes (cf. (Reynolds, 1988)). But even if σ and τ both exist at level n, there might be many common supertypes that will not appear until some later stage, so the limit with respect to only those supertypes that exist at level n might be too large. At each level, then, it appears that we would need to recalculate the interpretations of all the intersection types from previous levels. It is not obvious that this process would converge. Fortunately, in $\lambda_\wedge$, every σ and τ possess a least upper bound $\sigma \sqcup \tau$, which, furthermore, always appears at the first stage containing both σ and τ. So $\sigma \wedge \tau$ may be interpreted as the limit of a very tidy diagram

$$\begin{tikzcd} & \llbracket \sigma \sqcup \tau \rrbracket & \\ \llbracket \sigma \rrbracket \arrow[ur] & & \llbracket \tau \rrbracket \arrow[ul] \end{tikzcd}$$

with no fear that this interpretation will ever need to be revised.

The nonexistence of least upper bounds in $F_\wedge$ renders this important simplification useless. It is not clear whether a model could be constructed by "incrementally revising" the interpretations of intersections at each level, as described above. This kind of construction, if it worked at all, would almost certainly be much more complex than the known models of $\lambda_\wedge$.

7. Future Work

A primary practical concern for programming notations based on intersection types is the efficiency of typechecking for large programs. Naive implementations of the algorithms given here exhibit exponential behavior — in practice! — in both type synthesis (because of the *for* construct) and subtyping (because of rules ASUBR-INTER and ASUBL-INTER). Fortunately, this behavior normally occurs as a result of explicit programmer directives — requests, in effect, for an exponential amount of analysis of the program during typechecking. Still, a serious implementation must find ways to economize, for example by caching the partial results of previous analysis.

Another consideration for any language based on second-order polymorphism is the problem of verbosity. Without some means of abbreviation or partial type inference, even modest programs quickly become overburdened with type annotations. Cardelli's partial type inference method for $F_\leq$ (1993) offers one promising direction of investigation. Another possibility is to use an ω-order extension of $F_\wedge$, in which type operators can be used to express type information more succinctly (Compagnoni & Pierce, 1996; Compagnoni, 1994; Compagnoni, 1995).

Larger examples are needed to establish the practical need for intersection types and bounded quantification in their most general forms. It may be possible to obtain most of the practical power of $F_\wedge$ while remaining within a simpler, more tractable fragment.

Indeed, most of the known examples of bounded quantification and its variants and extensions can be treated within a system in which the troublesome SUB-ALL rule is replaced by a much more tractable "pointwise" rule

$$\frac{\Gamma, \alpha{\leq}\gamma \ \vdash \ \sigma_2 \ \leq \ \tau_2}{\Gamma \ \vdash \ \forall\alpha{\leq}\gamma.\ \sigma_2 \ \leq \ \forall\alpha{\leq}\gamma.\ \tau_2} \qquad \text{(SUB-ALL-KERNELFUN)}$$

originally used by Cardelli and Wegner (1985). It can be shown that this variant of $F_\wedge$ does possess least upper bounds for finite sets of types. All the techniques developed in this paper can be applied (often in simplified form) to this variant. (An interesting programming example that does require the full SUB-ALL rule can be found in (Abadi *et al.*, 1996).)

Acknowledgements

My Ph.D. thesis (Pierce, 1991a), from which this paper has been extracted, was written under the supervision of John Reynolds and Bob Harper. The other members of the thesis committee, Luca Cardelli and Nico Habermann, also contributed significantly to its final shape, and discussions with Peter Lee, Frank Pfenning, David Garlan, Nevin Heintze, Spiro Michaylov, QingMing Ma, and Tim Freeman led to the refinement of many of the ideas. Adriana Compagnoni, Simona Ronchi Della Rocca, and three journal referees made many useful comments on a later draft.

This research was sponsored in part by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U. S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, Arpa Order No. 7597; in part by the Office of Naval Research under Contract N00013-84-K-0415; in part by the National Science Foundation under Contract CCR-8922109; and in part by Siemens. The views and conclusions contained in this paper are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. Final revisions to this paper were carried out at the Laboratory for Foundations of Computer Science, University of Edinburgh, and at the Computer Laboratory, University of Cambridge.

A. Proofs of Technical Lemmas

Proof of Lemma 5.1.4: By induction on a derivation of $\Gamma \vdash \sigma \leq \tau$. Proceed by cases on the final rule.

Cases SUB-REFL, SUB-TRANS, SUB-ARROW, SUB-INTER-G, SUB-INTER-LB, SUB-DIST-IA, and SUB-DIST-IQ:

Either immediate or by straightforward use of the induction hypothesis.

Case SUB-TVAR*:* $\sigma \equiv \alpha \qquad \tau \equiv \Gamma(\alpha)$

By SUB-TVAR, $\Gamma' \vdash \alpha \leq \Gamma'(\alpha)$. By assumption, $\Gamma' \vdash \Gamma'(\alpha) \leq \Gamma(\alpha)$. By SUB-TRANS, $\Gamma' \vdash \alpha \leq \Gamma(\alpha)$.

Case SUB-ALL*:* $\sigma \equiv \forall\alpha{\leq}\sigma_1.\ \sigma_2 \qquad \tau \equiv \forall\alpha{\leq}\tau_1.\ \tau_2$

By assumption, $\Gamma \vdash \tau_1 \leq \sigma_1$ and $\Gamma,\ \alpha{\leq}\tau_1 \vdash \sigma_2 \leq \tau_2$. By the induction hypothesis, $\Gamma' \vdash \tau_1 \leq \sigma_1$. By assumption and weakening (Lemma 5.1.3), $\Gamma',\ \alpha{\leq}\tau_1 \vdash \Gamma'(\alpha_i) \leq \Gamma(\alpha_i)$ for each $\alpha_i \in dom(\Gamma)$. By SUB-REFL, $\Gamma',\ \alpha{\leq}\tau_1 \vdash (\Gamma',\ \alpha{\leq}\tau_1)(\alpha) \leq (\Gamma,\ \alpha{\leq}\tau_1)(\alpha)$. The induction hypothesis then gives $\Gamma',\ \alpha{\leq}\tau_1 \vdash \sigma_2 \leq \tau_2$. By SUB-ALL, $\Gamma' \vdash \forall\alpha{\leq}\sigma_1.\ \sigma_2 \leq \forall\alpha{\leq}\tau_1.\ \tau_2$. □

Proof of Lemma 5.1.6: By induction on a derivation of $\Gamma_1,\ \alpha{\leq}\psi,\ \Gamma_2 \vdash \sigma \leq \tau$.

Cases SUB-REFL, SUB-TRANS, SUB-ARROW, SUB-INTER-G, SUB-INTER-LB, SUB-DIST-IA, and SUB-DIST-IQ:

Either immediate or by straightforward use of the induction hypothesis.

Case SUB-TVAR*:* $\sigma \equiv \beta \qquad \tau \equiv (\Gamma_1,\ \alpha{\leq}\psi,\ \Gamma_2)(\beta)$

Subcase: $\alpha \equiv \beta$

By assumption, $\Gamma_1 \vdash \phi \leq \psi$. By weakening (5.1.3), $\Gamma_1,\ \{\phi/\alpha\}\Gamma_2 \vdash \phi \leq \psi$, that is, $\Gamma_1,\ \{\phi/\alpha\}\Gamma_2 \vdash \{\phi/\alpha\}\alpha \leq \{\phi/\alpha\}\psi$, as required.

Subcase: $\alpha \not\equiv \beta$

By SUB-TVAR, $\Gamma_1,\ \{\phi/\alpha\}\Gamma_2 \vdash \beta \leq (\Gamma_1,\ \{\phi/\alpha\}\Gamma_2)(\beta)$, that is, $\Gamma_1,\ \{\phi/\alpha\}\Gamma_2 \vdash \{\phi/\alpha\}\beta \leq \{\phi/\alpha\}(\Gamma_1,\ \alpha{\leq}\psi,\ \Gamma_2)(\beta)$.

Case SUB-ALL*:* $\sigma \equiv \forall\beta{\leq}\sigma_1.\ \sigma_2 \qquad \tau \equiv \forall\beta{\leq}\tau_1.\ \tau_2$

By assumption, $\Gamma_1,\ \alpha{\leq}\psi,\ \Gamma_2 \vdash \tau_1 \leq \sigma_1$ and $\Gamma_1,\ \alpha{\leq}\psi,\ \Gamma_2,\ \beta \leq \tau_1 \vdash \sigma_2{\leq}\tau_2$. By the induction hypothesis, $\Gamma_1,\ \{\phi/\alpha\}\Gamma_2 \vdash \{\phi/\alpha\}\tau_1 \leq \{\phi/\alpha\}\sigma_1$ and $\Gamma_1,\ \{\phi/\alpha\}\Gamma_2,\ \beta{\leq}\{\phi/\alpha\}\tau_1 \vdash \{\phi/\alpha\}\sigma_2 \leq \{\phi/\alpha\}\tau_2$. By SUB-ALL and the definition of substitution, $\Gamma_1,\ \{\phi/\alpha\}\Gamma_2 \vdash \{\phi/\alpha\}(\forall\beta{\leq}\sigma_1.\ \sigma_2) \leq \{\phi/\alpha\}(\forall\beta{\leq}\tau_1.\ \tau_2)$. □

Proof of Lemma 5.2.5.2: By induction on a derivation of $\Gamma \overset{\wedge}{\vdash} \sigma \leq \tau$. Proceed by cases on the final step of the derivation.

Case SUB-REFL*:* $\sigma \equiv \tau$

By CSUB-REFL.

Case SUB-TRANS*:*

By the induction hypothesis and CSUB-TRANS.

Case SUB-TVAR*:* $\sigma \equiv \alpha \qquad \tau \equiv \Gamma(\alpha)$

By the definition of canonical subtyping, $(\Gamma(\alpha))^\flat \equiv \Gamma^\flat(\alpha) \equiv \bigwedge[\kappa_1..\kappa_n]$. The desired result is checked by constructing the following derivation:

$$\dfrac{\dfrac{\dfrac{\Gamma^\flat \overset{\flat}{\vdash} \kappa_1 \leq \kappa_1}{\Gamma^\flat \overset{\flat}{\vdash} \Gamma^\flat(\alpha) \leq \bigwedge[\kappa_1]}\,\scriptstyle{(\text{CSUB-AE})}}{\Gamma^\flat \overset{\flat}{\vdash} \alpha \leq \kappa_1}\,\scriptstyle{(\text{CSUB-TVAR})} \qquad \cdots \qquad \dfrac{\dfrac{\Gamma^\flat \overset{\flat}{\vdash} \kappa_n \leq \kappa_n}{\Gamma^\flat \overset{\flat}{\vdash} \Gamma^\flat(\alpha) \leq \bigwedge[\kappa_n]}\,\scriptstyle{(\text{CSUB-AE})}}{\Gamma^\flat \overset{\flat}{\vdash} \alpha \leq \kappa_n}\,\scriptstyle{(\text{CSUB-TVAR})}}{\Gamma^\flat \overset{\flat}{\vdash} \bigwedge[\alpha] \leq \bigwedge[\kappa_1..\kappa_n]}\,\scriptstyle{(\text{CSUB-AE})}$$

Case SUB-ARROW: $\sigma \equiv \sigma_1 \rightarrow \sigma_2 \qquad \tau \equiv \tau_1 \rightarrow \tau_2$
$\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma \vdash \sigma_2 \leq \tau_2$

By the induction hypothesis, $\Gamma^\flat \not\vdash \tau_1^\flat \leq \sigma_1^\flat$ and $\Gamma^\flat \not\vdash \sigma_2^\flat \leq \tau_2^\flat$. By the syntax-directedness of canonical subtyping (5.2.4.1(2)), for every $\iota_j \in \tau_2^\flat$ there is some $\kappa_j \in \sigma_2^\flat$ such that $\Gamma^\flat \not\vdash \kappa_j \leq \iota_j$; in each case, CSUB-ARROW gives $\Gamma^\flat \not\vdash \sigma_1^\flat \rightarrow \kappa_j \leq \tau_1^\flat \rightarrow \iota_j$. By CSUB-AE, $\Gamma^\flat \not\vdash \bigwedge[\sigma_1^\flat \rightarrow \kappa \mid \kappa \in \sigma_2^\flat] \leq \bigwedge[\tau_1^\flat \rightarrow \iota \mid \iota \in \tau_2^\flat]$, as required.

Case SUB-ALL: $\sigma \equiv \forall\alpha{\leq}\sigma_1.\ \sigma_2 \qquad \tau \equiv \forall\alpha{\leq}\tau_1.\ \tau_2$

Similar.

Case SUB-INTER-G: $\tau \equiv \bigwedge[\tau_1..\tau_n] \qquad \Gamma \vdash \sigma \leq \tau_i$ for each τ_i

For each τ_i, the induction hypothesis gives $\Gamma^\flat \not\vdash \sigma^\flat \leq \tau_i^\flat$. By the syntax-directedness of canonical subtyping, for each $\iota_{ij} \in \tau_i^\flat$ there is some $\kappa_{ij} \in \sigma^\flat$ such that $\Gamma^\flat \not\vdash \kappa_{ij} \leq \iota_{ij}$. Combining these derivations for all the τ_i's, we have $\Gamma^\flat \not\vdash \sigma^\flat \leq \bigcup_i \tau_i^\flat$ by CSUB-AE.

Case SUB-INTER-LB: $\tau \equiv \sigma_i \qquad \sigma \equiv \bigwedge[\sigma_1..\sigma_n]$

Let ${\sigma_i}^\flat \equiv \bigwedge[\sigma_{i1} \mathinner{..} \sigma_{im}]$. Then the derivation

$$\frac{\Gamma^\flat \not\vdash \sigma_{i1}^\flat \leq \sigma_{i1}^\flat \qquad \cdots \qquad \Gamma^\flat \not\vdash \sigma_{im}^\flat \leq \sigma_{im}^\flat}{\Gamma^\flat \not\vdash \bigcup_i \sigma_i^\flat \leq \sigma_i^\flat} \text{ (CSUB-AE)}$$

establishes the desired result.

Case SUB-DIST-IA: $\sigma \equiv \bigwedge[\sigma' \rightarrow \tau_1 \mathinner{..} \sigma' \rightarrow \tau_n] \qquad \tau \equiv \sigma' \rightarrow \bigwedge[\tau_1..\tau_n]$

By the definition of flattening, $\sigma^\flat \equiv \tau^\flat$. The result follows by CSUB-REFL.

Case SUB-DIST-IQ: $\sigma \equiv \bigwedge[\forall\alpha{\leq}\sigma'.\ \tau_1 \mathinner{..} \forall\alpha{\leq}\sigma'.\ \tau_n] \qquad \tau \equiv \forall\alpha{\leq}\sigma'.\ \bigwedge[\tau_1..\tau_n]$

Similar. □

Proof of Lemma 5.2.5.4: By induction on the structure of τ.

Case: $\tau \equiv \alpha$

By SUB-INTER-LB and SUB-INTER-G.

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

By the definition of flattening (5.2.5.1), $\tau^\flat \equiv \bigwedge[\tau_1^\flat \rightarrow \kappa \mid \kappa \in \tau_2^\flat]$. By derived rule D-DIST-IA (5.1.1), $\Gamma \vdash \tau^\flat \sim (\tau_1^\flat \rightarrow \bigwedge[\kappa \mid \kappa \in \tau_2^\flat])$, i.e., $\Gamma \vdash \tau^\flat \sim \tau_1^\flat \rightarrow \tau_2^\flat$. By the induction hypothesis, $\Gamma \vdash \tau_1^\flat \sim \tau_1$ and $\Gamma \vdash \tau_2^\flat \sim \tau_2$. The desired result follows by D-CONG-ARROW (5.1.5) and SUB-TRANS.

Case: $\tau \equiv \forall\alpha{\leq}\tau_1.\ \tau_2$

Similar.

Case: $\tau \equiv \bigwedge[\tau_1..\tau_n]$

By the induction hypothesis, $\Gamma \vdash \tau_i^\flat \sim \tau_i$ for each i. By D-CONG-INTER (5.1.5), $\Gamma \vdash \bigwedge[\tau_1^\flat \mathinner{..} \tau_n^\flat] \sim \bigwedge[\tau_1..\tau_n]$. The result then follows by D-ABSORB (5.1.1) and SUB-TRANS. □

Proof of Lemma 5.3.1.2:

1 By induction on the definition of $arrowbasis_\Gamma$.

2 By the completeness of the subtyping algorithm (5.2.6.8), $\Gamma \vdash \sigma \leq \tau_1 \to \tau_2$ implies $\Gamma \vdash \sigma \leq [\tau_1, []] \Rightarrow \tau_2$. We show, by induction on derivations, that $\Gamma \vdash \sigma \leq [\tau_1, X_a] \Rightarrow \tau_b$ implies $\Gamma \vdash \bigwedge(arrowbasis_\Gamma(\sigma)) \leq [\tau_1, X_a] \Rightarrow \tau_b$, from which the desired result follows as a special case, since $\tau_1 \to \tau_2$ can always be written in the form $[\tau_1, X_a] \Rightarrow \tau_b$, where the outermost constructor of τ_b is $\wedge$ or a variable.

Proceed by cases on the final step of a derivation of $\Gamma \vdash \sigma \leq [\tau_1, X_a] \Rightarrow \tau_b$.

Case ASUBR-INTER: $\tau_b \equiv \bigwedge[\tau_{b1}..\tau_{bn}]$

By assumption, $\Gamma \vdash \sigma \leq [\tau_1, X_a] \Rightarrow \tau_{bi}$ for each i; by the induction hypothesis, $\Gamma \vdash \bigwedge(arrowbasis_\Gamma(\sigma)) \leq [\tau_1, X_a] \Rightarrow \tau_{bi}$. By derived rule D-CONG-INTER (5.1.5), $\Gamma \vdash \bigwedge[\bigwedge(arrowbasis_\Gamma(\sigma)) .. \bigwedge(arrowbasis_\Gamma(\sigma))] \leq \bigwedge[([\tau_1, X_a] \Rightarrow \tau_{b1}) .. ([\tau_1, X_a] \Rightarrow \tau_{bn})]$. By D-ABSORB and D-REINDEX (5.1.1), followed by SUB-TRANS, we know that $\Gamma \vdash \bigwedge(arrowbasis_\Gamma(\sigma)) \leq \bigwedge[([\tau_1, X_a] \Rightarrow \tau_{b1}) .. ([\tau_1, X_a] \Rightarrow \tau_{bn})]$. By $len([\tau_1, X_a])$ applications of SUB-DIST-IA and SUB-DIST-IQ (as appropriate) and SUB-TRANS, $\Gamma \vdash \bigwedge(arrowbasis_\Gamma(\sigma)) \leq [\tau_1, X_a] \Rightarrow \bigwedge[\tau_{b1}..\tau_{bn}]$.

Case ASUBL-INTER: $\sigma \equiv \bigwedge[\sigma_1..\sigma_n] \qquad \tau_b \equiv \alpha$

By assumption, $\Gamma \vdash \sigma_i \leq [\tau_1, X_a] \Rightarrow \alpha$ for some i. By the induction hypothesis, $\Gamma \vdash \bigwedge(arrowbasis_\Gamma(\sigma_i)) \leq [\tau_1, X_a] \Rightarrow \alpha$. Since $arrowbasis_\Gamma(\sigma_i) \subseteq arrowbasis_\Gamma(\sigma)$, SUB-REFL, D-ALL-SOME (5.1.1), and SUB-TRANS give $\Gamma \vdash \bigwedge(arrowbasis_\Gamma(\sigma)) \leq [\tau_1, X_a] \Rightarrow \alpha$.

Case ASUBL-ARROW: $\sigma \equiv \sigma_1 \to \sigma_2 \qquad \tau_b \equiv \alpha$

By the definition of $arrowbasis_\Gamma$ and the equivalence of ordinary and syntax-directed subtyping (5.2.6.10).

Case ASUBL-ALL: $\sigma \equiv \forall \alpha \leq \sigma_1.\ \sigma_2 \qquad \tau_b \equiv \alpha$

Can't happen ($[\tau_1, X_b]$ has the wrong form).

Case ASUBL-REFL: $\sigma \equiv \beta \qquad \tau_b \equiv \beta \qquad [\tau_1, X_a] \equiv []$

Can't happen.

Case ASUBL-TVAR: $\sigma \equiv \beta \qquad \tau_b \equiv \alpha \qquad \Gamma \vdash \Gamma(\beta) \leq [\tau_1, X_a] \Rightarrow \alpha$

By the induction hypothesis, $\Gamma \vdash \bigwedge(arrowbasis(\Gamma(\beta))) \leq [\tau_1, X_a] \Rightarrow \alpha$. By the definition of $arrowbasis_\Gamma$, $\Gamma \vdash \bigwedge(arrowbasis(\beta)) \leq [\tau_1, X_a] \Rightarrow \alpha$. □

Proof of Lemma 5.3.1.4: By the equivalence of ordinary and canonical subtyping (5.2.5.6), we have $\Gamma^\flat \vdash (\bigwedge M)^\flat \leq (\tau_1 \to \tau_2)^\flat$ and $\Gamma^\flat \vdash \sigma^\flat \leq \tau_1^\flat$, with $V \equiv [\psi_i \mid \Gamma^\flat \vdash \sigma^\flat \leq \phi_i^\flat]$. By the definition of $\flat$ (5.2.5.1), $\Gamma^\flat \vdash \bigcup_i(\bigwedge[\phi_i^\flat \to \kappa \mid \kappa \in \psi_i^\flat]) \leq \bigwedge[\tau_1^\flat \to \iota \mid \iota \in \tau_2^\flat]$. By the syntax-directedness of canonical subtyping (5.2.4.1(1)), for all $\iota \in \tau_2^\flat$ there is some i and some $\kappa \in \psi_i^\flat$ such that $\Gamma^\flat \vdash \phi_i^\flat \to \kappa \leq \tau_1^\flat \to \iota$. By syntax-directedness again (5.2.4.1(2)), for all $\iota \in \tau_2^\flat$ there is some i and some $\kappa \in \psi_i^\flat$ such that $\Gamma^\flat \vdash \tau_1^\flat \leq \phi_i^\flat$ and $\Gamma^\flat \vdash \kappa \leq \iota$, that is, for all $\iota \in \tau_2^\flat$ there is some i such that $\Gamma^\flat \vdash \tau_1^\flat \leq \phi_i^\flat$ and there is some $\kappa \in {\psi_\iota}^\flat$ such that $\Gamma^\flat \vdash^{\flat} \kappa \leq \iota$. By CSub-Trans, for all $\iota \in \tau_2^\flat$ there is some i such that $\Gamma^\flat \vdash^{\flat} \sigma^\flat \leq {\phi_i}^\flat$ and there is some $\kappa \in {\psi_\iota}^\flat$ such that $\Gamma^\flat \vdash^{\flat} \kappa \leq \iota$. By CSub-AE, $\Gamma^\flat \vdash^{\flat} \bigcup_i(\bigwedge[\kappa \mid \kappa \in {\psi_i}^\flat \text{ and } \Gamma^\flat \vdash^{\flat} \sigma^\flat \leq {\phi_i}^\flat]) \leq \tau_2^\flat$, that is, $\Gamma^\flat \vdash^{\flat} (\bigwedge V)^\flat \leq \tau_2^\flat$ By the equivalence of ordinary and canonical subtyping (5.2.5.6), $\Gamma \vdash^{\wedge} \bigwedge V \leq \tau_2$. $\square$

Proof of Lemma 5.3.1.5: By the equivalence of ordinary and canonical subtyping (Theorem 5.2.5.6), we have $\Gamma^\flat \vdash^{\flat} (\bigwedge M)^\flat \leq (\forall\alpha{\leq}\tau_1.\ \tau_2)^\flat$ and $\Gamma^\flat \vdash^{\flat} \sigma^\flat \leq \tau_1^\flat$, with $V \equiv [\{\sigma/\alpha\}\psi_i \mid \Gamma^\flat \vdash^{\flat} \sigma^\flat \leq {\phi_i}^\flat]$. By the definition of $\flat$, $\Gamma^\flat \vdash^{\flat} \bigcup_i(\bigwedge[\forall\alpha{\leq}{\phi_i}^\flat.\ \kappa \mid \kappa \in {\psi_i}^\flat]) \leq \bigwedge[\forall\alpha{\leq}\tau_1^\flat.\ \iota \mid \iota \in \tau_2^\flat]$. By the syntax-directedness of canonical subtyping (5.2.4.1(1)), for all $\iota \in \tau_2^\flat$ there is some i and some $\kappa \in {\psi_i}^\flat$ such that $\Gamma^\flat \vdash^{\flat} (\forall\alpha{\leq}{\phi_i}^\flat.\ \kappa) \leq (\forall\alpha{\leq}\tau_1^\flat.\ \iota)$. By syntax-directedness again (5.2.4.1(3)), for all $\iota \in \tau_2^\flat$ there is some i and some $\kappa \in {\psi_i}^\flat$ such that $\Gamma^\flat \vdash^{\flat} \tau_1^\flat \leq {\phi_i}^\flat$ and $\Gamma^\flat,\ \alpha{\leq}\tau_1^\flat \vdash \kappa \leq \iota$, that is, for all $\iota \in \tau_2^\flat$ there is some i such that $\Gamma^\flat \vdash^{\flat} \tau_1^\flat \leq {\phi_i}^\flat$ and there is some $\kappa \in {\psi_i}^\flat$ such that $\Gamma^\flat,\ \alpha{\leq}\tau_1^\flat \vdash \kappa \leq \iota$. By CSub-Trans, for all $\iota \in \tau_2^\flat$ there is some i such that $\Gamma^\flat \vdash^{\flat} \sigma^\flat \leq {\phi_i}^\flat$ and there is some $\kappa \in {\psi_i}^\flat$ such that $\Gamma^\flat,\ \alpha{\leq}\tau_1^\flat \vdash \kappa \leq \iota$. By CSub-AE, $\Gamma^\flat,\ \alpha{\leq}\tau_1^\flat \vdash^{\flat} \bigcup_i(\bigwedge[\kappa \mid \kappa \in \psi_i \text{ and } \Gamma^\flat \vdash^{\flat} \sigma^\flat \leq {\phi_i}^\flat]) \leq \tau_2^\flat$ that is, $\Gamma^\flat,\ \alpha{\leq}\tau_1^\flat \vdash^{\flat} (\bigwedge[\psi_i \mid \Gamma^\flat \vdash^{\flat} \sigma^\flat \leq {\phi_i}^\flat])^\flat \leq \tau_2^\flat$. By the equivalence of ordinary and canonical subtyping (5.2.5.6), $\Gamma,\ \alpha{\leq}\tau_1 \vdash^{\wedge} \bigwedge[\psi_i \mid \Gamma \vdash \sigma \leq \phi_i] \leq \tau_2$. Then by the substitution property (5.1.6), $\Gamma \vdash^{\wedge} \{\sigma/\alpha\}(\bigwedge[\psi_i \mid \Gamma \vdash \sigma \leq \phi_i]) \leq \{\sigma/\alpha\}\tau_2$, that is, $\Gamma \vdash^{\wedge} \bigwedge V \leq \{\sigma/\alpha\}\tau_2$. $\square$

Proof of Theorem 5.3.3.3:

1 ($\Longleftarrow$) Lemma 3.3.2.3.

($\Longrightarrow$) If $\Gamma_P,\ \Gamma \vdash^{\wedge} \sigma \leq \tau$, then by the completeness of the subtyping algorithm, $\Gamma_P,\ \Gamma \vdash^{\cdot} \sigma \leq \tau$. By the syntax-directedness of the subtyping algorithm and the fact that σ, τ, and Γ contain no quantified types, this derivation will not contain any instances of ASubL-All, the rule that deals with quantified types. It may be therefore be rewritten as a derivation from the $\lambda_\wedge$ rules by a translation similar to the one in the proof of Theorem 5.2.6.6, dropping the bindings Γ_P and translating instances of ASubL-Refl as instances of Sub-Refl and instances of ASubL-TVar as derivations of the following form:

$$\dfrac{\dfrac{\forall\gamma \in P \text{ with } \beta{\leq_P}\gamma. \quad \dfrac{\beta \leq_P \gamma}{\Gamma \vdash^{\lambda\wedge} \beta \leq \gamma}\ \text{(SUB-PRIM)}}{\Gamma \vdash^{\lambda\wedge} \beta \leq \bigwedge[\gamma \in P \mid \beta{\leq_P}\gamma]}\ \text{(SUB-INTER-G)} \qquad \dfrac{\text{(induction hypothesis)}}{\Gamma \vdash^{\lambda\wedge} \bigwedge[\gamma \in P \mid \beta{\leq_P}\gamma] \leq X{\Rightarrow}\alpha}}{\Gamma \vdash^{\lambda\wedge} \beta \leq X{\Rightarrow}\alpha.}\ \text{(SUB-TRANS)}$$

2 ($\Longleftarrow$) Fact 3.3.2.4.

($\Longrightarrow$) If $\Gamma,\ \Gamma_P \vdash^{\wedge} e \in \tau$, then by the completeness of the subtyping algorithm, $\Gamma,\ \Gamma_P \vdash^{\cdot} e \in \tau$. By the syntax-directedness of the subtyping algorithm and the fact that e, τ, and Γ contain no type abstractions, type applications, or quantified types, this derivation will not contain any instances of A-All-I or A-All-E. It may be therefore be rewritten straightforwardly as a derivation from the $\lambda_\wedge$ rules, using the previous case to handle the translation of subtyping derivations. $\square$

Proof of Lemma 6.1.1.3: By induction on the form of C.

Case: $x \notin FV(C)$

$$
\begin{aligned}
[\![\lambda^\star x.\ C]\!]_\eta \cdot m &= [\![K\,C]\!]_\eta \cdot m \\
&= k \cdot [\![C]\!]_\eta \cdot m \\
&= [\![C]\!]_\eta \\
&= [\![C]\!]_{\eta[x\leftarrow m]} \quad \text{by Lemma 6.1.1.2.}
\end{aligned}
$$

Case: $C \equiv x$

$$
\begin{aligned}
[\![\lambda^\star x.\ C]\!]_\eta \cdot m &= [\![S\,K\,K]\!]_\eta \cdot m \\
&= s \cdot k \cdot k \cdot m \\
&= m \\
&= (\eta[x\leftarrow m])(x) \\
&= [\![C]\!]_{\eta[x\leftarrow m]}.
\end{aligned}
$$

Case: $C \equiv C_1\,C_2 \qquad x \in FV(C_1\,C_2)$

$$
\begin{aligned}
[\![\lambda^\star x.\ C]\!]_\eta \cdot m &= [\![S\,(\lambda^\star x.\ C_1)\,(\lambda^\star x.\ C_2)]\!]_\eta \cdot m \\
&= s \cdot [\![\lambda^\star x.\ C_1]\!]_\eta \cdot [\![\lambda^\star x.\ C_2]\!]_\eta \cdot m \\
&= ([\![\lambda^\star x.\ C_1]\!]_\eta \cdot m) \cdot ([\![\lambda^\star x.\ C_2]\!]_\eta \cdot m) \\
&= ([\![C_1]\!]_{\eta[x\leftarrow m]}) \cdot ([\![C_2]\!]_{\eta[x\leftarrow m]}) \quad \text{by the induction hypothesis} \\
&= ([\![C_1\,C_2]\!]_{\eta[x\leftarrow m]}).
\end{aligned}
$$

□

Proof of Lemma 6.1.2.7: By induction on the structure of a derivation of $\Gamma \vdash \sigma \leq \tau$.

Case SUB-REFL*:* $\sigma \equiv \tau$

Immediate.

Case SUB-TRANS*:* $\Gamma \vdash \sigma \leq \theta \qquad \Gamma \vdash \theta \leq \tau$

By the induction hypothesis.

Case SUB-TVAR*:* $\sigma \equiv \alpha \qquad \tau \equiv \Gamma(\alpha)$

Immediate from 6.1.2.5.

Case SUB-ARROW*:* $\sigma \equiv \sigma_1 \to \sigma_2 \qquad \tau \equiv \tau_1 \to \tau_2 \qquad \Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma \vdash \sigma_2 \leq \tau_2$

$$
\begin{aligned}
& m\,\{[\![\sigma_1 \to \sigma_2]\!]_\eta\}\,n \\
\Leftrightarrow\ & m\,\{[\![\sigma_1]\!]_\eta \to [\![\sigma_2]\!]_\eta\}\,n \\
\Leftrightarrow\ & \forall p, q.\quad p\,\{[\![\sigma_1]\!]_\eta\}\,q \text{ implies } m\cdot p\,\{[\![\sigma_2]\!]_\eta\}\,n\cdot q \\
\Rightarrow\ & \forall p, q.\quad p\,\{[\![\tau_1]\!]_\eta\}\,q \text{ implies } m\cdot p\,\{[\![\tau_2]\!]_\eta\}\,n\cdot q \quad \text{by the induction hypothesis} \\
\Leftrightarrow\ & m\,\{[\![\tau_1 \to \tau_2]\!]_\eta\}\,n.
\end{aligned}
$$

Case SUB-ALL*:* $\sigma \equiv \forall\alpha{\leq}\sigma_1.\ \sigma_2 \qquad \tau \equiv \forall\alpha{\leq}\tau_1.\ \tau_2 \qquad \Gamma \vdash \tau_1 \leq \sigma_1$
$\Gamma,\ \alpha{\leq}\tau_1 \vdash \sigma_2 \leq \tau_2$

$$
\begin{aligned}
& m\,\{[\![\forall\alpha{\leq}\sigma_1.\ \sigma_2]\!]_\eta\}\,n \\
\Leftrightarrow\ & m\,\{\textstyle\bigcap_{A \subseteq [\![\sigma_1]\!]_\eta} [\![\sigma_2]\!]_{\eta[\alpha\leftarrow A]}\}\,n \\
\Leftrightarrow\ & \forall A \subseteq [\![\sigma_1]\!]_\eta.\quad m\,\{[\![\sigma_2]\!]_{\eta[\alpha\leftarrow A]}\}\,n \\
\Rightarrow\ & \forall A \subseteq [\![\tau_1]\!]_\eta.\quad m\,\{[\![\tau_2]\!]_{\eta[\alpha\leftarrow A]}\}\,n \quad \text{by the induction hypothesis} \\
\Leftrightarrow\ & m\,\{[\![\forall\alpha{\leq}\tau_1.\ \tau_2]\!]_\eta\}\,n.
\end{aligned}
$$

Case SUB-INTER-G*:* $\tau \equiv \bigwedge[\tau_1..\tau_n]$ for all i, $\Gamma \vdash \sigma \leq \tau_i$

By the induction hypothesis, $[\![\sigma]\!]_\eta \subseteq [\![\tau_i]\!]_\eta$ for each i; so $[\![\sigma]\!]_\eta \subseteq \bigcap_{1\leq i\leq n}[\![\tau_i]\!]_\eta = [\![\bigwedge[\tau_1..\tau_n]]\!]_\eta$.

Case SUB-INTER-LB*:* $\sigma \equiv \bigwedge[\tau_1..\tau_n]$ $\tau \equiv \tau_i$

Immediate from the definition of $\bigwedge$.

Case SUB-DIST-IA*:* $\sigma \equiv \bigwedge[\sigma'\!\to\!\tau_1 \;..\; \sigma'\!\to\!\tau_n]$ $\tau \equiv \sigma'\!\to\!\bigwedge[\tau_1..\tau_n]$

$$\begin{array}{ll}
& m\;\{[\![\bigwedge[\sigma'\!\to\!\tau_1 \;..\; \sigma'\!\to\!\tau_n]]\!]_\eta\}\;m' \\
\Leftrightarrow & m\;\{\bigcap_{1\leq i\leq n}[\![\sigma'\!\to\!\tau_i]\!]_\eta\}\;m' \\
\Leftrightarrow & \forall i.\quad m\;\{[\![\sigma']\!]_\eta\!\to\![\![\tau_i]\!]_\eta\}\;m' \\
\Leftrightarrow & \forall i.\quad \forall p,q.\quad p\;\{[\![\sigma']\!]_\eta\}\;q \text{ implies } m{\cdot}p\;\{[\![\tau_i]\!]_\eta\}\;m'{\cdot}q \\
\Leftrightarrow & \forall p,q.\quad p\;\{[\![\sigma']\!]_\eta\}\;q \text{ implies } (\forall i.\; m{\cdot}p\;\{[\![\tau_i]\!]_\eta\}\;m'{\cdot}q) \\
\Leftrightarrow & \forall p,q.\quad p\;\{[\![\sigma']\!]_\eta\}\;q \text{ implies } m{\cdot}p\;\{[\![\bigwedge[\tau_1..\tau_n]]\!]_\eta\}\;m'{\cdot}q \\
\Leftrightarrow & m\;\{[\![\sigma'\!\to\!\bigwedge[\tau_1..\tau_n]]\!]_\eta\}\;m'.
\end{array}$$

Case SUB-DIST-IQ*:* $\sigma \equiv \bigwedge[\forall\alpha{\leq}\sigma'.\;\tau_1 \;..\; \forall\alpha{\leq}\sigma'.\;\tau_n]$ $\tau \equiv \forall\alpha{\leq}\sigma'.\;\bigwedge[\tau_1..\tau_n]$

$$\begin{array}{ll}
& [\![\bigwedge[\forall\alpha{\leq}\sigma'.\;\tau_1 \;..\; \forall\alpha{\leq}\sigma'.\;\tau_n]]\!]_\eta \\
= & \bigcap_{1\leq i\leq n}\bigcap_{A\subseteq[\![\sigma']\!]_\eta}\tau_i \\
= & \bigcap_{A\subseteq[\![\sigma']\!]_\eta}\bigcap_{1\leq i\leq n}\tau_i \\
= & [\![\forall\alpha{\leq}\sigma.\;\bigwedge[\tau_1..\tau_n]]\!]_\eta.
\end{array}$$

□

Proof of Lemma 6.1.2.9: By induction on a derivation of $\Gamma \vdash e \in \tau$.

Case VAR*:* $e \equiv x$ $\tau \equiv \Gamma(x)$

Immediate.

Case ARROW-I*:* $e \equiv \lambda x{:}\tau_1.\;e'$ $\Gamma,\,x{:}\tau_1 \vdash e' \in \tau_2$ $\tau \equiv \tau_1\!\to\!\tau_2$

Choose m and n such that $m\;\{[\![\tau_1]\!]_\eta\}\;n$. Then $m\;\{[\![\tau_1]\!]_\eta\}\;m$ and $n\;\{[\![\tau_1]\!]_\eta\}\;n$, so $\eta_1[x{\leftarrow}m] \models \Gamma,\;x{:}\tau_1$ and $\eta_2[x{\leftarrow}n] \models \Gamma,\;x{:}\tau_1$. The induction hypothesis gives $[\![e']\!]_{\eta_1[x\leftarrow m]}\;\{[\![\tau_2]\!]_\eta\}\;[\![e']\!]_{\eta_2[x\leftarrow n]}$. But

$$\begin{array}{lcll}
[\![e]\!]_{\eta_1}\cdot m & = & [\![\lambda^\star x.\;|erase(e')|]\!]_{\eta_1}\cdot m & \text{by definition} \\
& = & [\![|erase(e')|]\!]_{\eta_1[x\leftarrow m]} & \text{by Lemma 6.1.1.3} \\
& = & [\![e']\!]_{\eta_1[x\leftarrow m]} & \text{by definition,}
\end{array}$$

and similarly $[\![e]\!]_{\eta_2}\cdot n = [\![e']\!]_{\eta_2[x\leftarrow n]}$. So $[\![e]\!]_{\eta_1}\cdot m\;\{[\![\tau_2]\!]_\eta\}\;[\![e]\!]_{\eta_2}\cdot n$. Since this holds for all m and n such that $m\;\{[\![\tau_1]\!]_\eta\}\;n$, the definition of $\to$ gives $[\![e]\!]_{\eta_1}\;\{[\![\tau_1]\!]_\eta\!\to\![\![\tau_2]\!]_\eta\}\;[\![e]\!]_{\eta_2}$, i.e., $[\![e]\!]_{\eta_1}\;\{[\![\tau_1\!\to\!\tau_2]\!]_\eta\}\;[\![e]\!]_{\eta_2}$.

Case ARROW-E*:* $e \equiv e_1\,e_2$ $\Gamma \vdash e_1 \in \tau_1\!\to\!\tau_2$ $\Gamma \vdash e_2 \in \tau_1$ $\tau \equiv \tau_2$

By the induction hypothesis, $[\![e_1]\!]_{\eta_1}\;\{[\![\tau_1\!\to\!\tau_2]\!]_\eta\}\;[\![e_1]\!]_{\eta_2}$, i.e. $[\![e_1]\!]_{\eta_1}\;\{[\![\tau_1]\!]_\eta\!\to\![\![\tau_2]\!]_\eta\}\;[\![e_1]\!]_{\eta_2}$, and $[\![e_2]\!]_{\eta_1}\;\{[\![\tau_1]\!]_\eta\}\;[\![e_2]\!]_{\eta_2}$. So, by the definition of $\to$, $[\![e_1]\!]_{\eta_1}\cdot[\![e_2]\!]_{\eta_1}\;\{[\![\tau_2]\!]_\eta\}\;[\![e_1]\!]_{\eta_2}\cdot[\![e_2]\!]_{\eta_2}$, i.e. (by Lemma 6.1.2.8(1)), $[\![e_1\,e_2]\!]_{\eta_1}\;\{[\![\tau_2]\!]_\eta\}\;[\![e_1\,e_2]\!]_{\eta_2}$.

Case ALL-I*:* $e \equiv \Lambda\alpha{\leq}\tau.\ e' \qquad \Gamma, \alpha{\leq}\tau_1 \vdash e' \in \tau_2 \qquad \tau \equiv \forall\alpha{\leq}\tau_1.\ \tau_2$

Choose an arbitrary PER $A \subseteq \llbracket\tau_1\rrbracket_\eta$. By the induction hypothesis, we know that $\llbracket e'\rrbracket_{\eta_1[\alpha\leftarrow A]}\ \{\llbracket\tau_2\rrbracket_{\eta[\alpha\leftarrow A]}\}\ \llbracket e'\rrbracket_{\eta_2[\alpha\leftarrow A]}$. By Lemma 6.1.1.2, $\llbracket e'\rrbracket_{\eta_1}\ \{\llbracket\tau_2\rrbracket_{\eta[\alpha\leftarrow A]}\}\ \llbracket e'\rrbracket_{\eta_2}$. This is so for every $A \subseteq \llbracket\tau_1\rrbracket_\eta$, so the definition of $\bigcap$ gives $\llbracket e'\rrbracket_{\eta_1}\ \{\bigcap_{A\subseteq\llbracket\tau_1\rrbracket_\eta} \llbracket\tau_2\rrbracket_{\eta[\alpha\leftarrow A]}\}\ \llbracket e'\rrbracket_{\eta_2}$, that is, $\llbracket e'\rrbracket_{\eta_1}\ \{\llbracket\forall\alpha{\leq}\tau_1.\ \tau_2\rrbracket_\eta\}\ \llbracket e'\rrbracket_{\eta_2}$, that is (by the definition of *erase*), $\llbracket e\rrbracket_{\eta_1}\ \{\llbracket\forall\alpha{\leq}\tau_1.\ \tau_2\rrbracket_\eta\}\ \llbracket e\rrbracket_{\eta_2}$.

Case ALL-E*:* $e \equiv e'\,[\sigma] \qquad \Gamma \vdash e' \in \forall\alpha{\leq}\tau_1.\ \tau_2 \qquad \Gamma \vdash \sigma \leq \tau_1 \qquad \tau \equiv \tau_2$

By the induction hypothesis, we know that $\llbracket e'\rrbracket_{\eta_1}\ \{\llbracket\forall\alpha{\leq}\tau_1.\ \tau_2\rrbracket_\eta\}\ \llbracket e'\rrbracket_{\eta_2}$, that is, $\llbracket e'\rrbracket_{\eta_1}\ \{\bigcap_{A\subseteq\llbracket\tau_1\rrbracket_\eta} \llbracket\tau_2\rrbracket_{\eta[\alpha\leftarrow A]}\}\ \llbracket e'\rrbracket_{\eta_2}$. Since, by Lemma 6.1.2.7, $\llbracket\sigma\rrbracket_\eta \subseteq \llbracket\tau_1\rrbracket_\eta$, we know that $\llbracket e'\rrbracket_{\eta_1}\ \{\llbracket\tau_2\rrbracket_{\eta[\alpha\leftarrow\llbracket\sigma\rrbracket_\eta]}\}\ \llbracket e'\rrbracket_{\eta_2}$, i.e. (by the definition of *erase*), $\llbracket e\rrbracket_{\eta_1}\ \{\llbracket\tau_2\rrbracket_{\eta[\alpha\leftarrow\llbracket\sigma\rrbracket_\eta]}\}\ \llbracket e\rrbracket_{\eta_2}$, i.e. (by Lemma 6.1.2.8(3) $\llbracket e\rrbracket_{\eta_1}\ \{\llbracket\{\sigma/\alpha\}\tau_2\rrbracket_\eta\}\ \llbracket e\rrbracket_{\eta_2}$.

Case FOR*:* $e \equiv \mathit{for}\ \alpha\ \mathit{in}\ \sigma_1..\sigma_n.\ .e' \qquad \Gamma \vdash \{\sigma_i/\alpha\}e' \in \tau_i \qquad \tau \equiv \tau_i$

By the induction hypothesis, $\llbracket\{\sigma_i/\alpha\}e'\rrbracket_{\eta_1}\ \{\llbracket\tau_i\rrbracket_\eta\}\ \llbracket\{\sigma_i/\alpha\}e'\rrbracket_{\eta_2}$. By Lemma 6.1.2.8(2), $\llbracket e'\rrbracket_{\eta_1}\ \{\llbracket\tau_i\rrbracket_\eta\}\ \llbracket e'\rrbracket_{\eta_2}$. By the definition of *erase*, $\llbracket e\rrbracket_{\eta_1}\ \{\llbracket\tau_i\rrbracket_\eta\}\ \llbracket e\rrbracket_{\eta_2}$.

Case INTER-I*:* $\Gamma \vdash e \in \tau_i$ for each $i \qquad \tau \equiv \bigwedge[\tau_1..\tau_n]$

By the induction hypothesis, $\llbracket e\rrbracket_{\eta_1}\ \{\llbracket\tau_i\rrbracket_\eta\}\ \llbracket e\rrbracket_{\eta_2}$ for each i; hence, $\llbracket e\rrbracket_{\eta_1}\ \{\bigcap_{1\leq i\leq n}\llbracket\tau_i\rrbracket_\eta\}\ \llbracket e\rrbracket_{\eta_2}$, i.e., $\llbracket e\rrbracket_{\eta_1}\ \{\llbracket\bigwedge[\tau_1..\tau_n]\rrbracket_\eta\}\ \llbracket e\rrbracket_{\eta_2}$.

Case SUB*:* $\Gamma \vdash e \in \sigma \qquad \Gamma \vdash \sigma \leq \tau$

By the induction hypothesis, $\llbracket e\rrbracket_{\eta_1}\ \{\llbracket\sigma\rrbracket_\eta\}\ \llbracket e\rrbracket_{\eta_2}$, hence (by Lemma 6.1.2.7) $\llbracket e\rrbracket_{\eta_1}\ \{\llbracket\tau\rrbracket_\eta\}\ \llbracket e\rrbracket_{\eta_2}$. □

B. Summary of Major Definitions

B.1. $F_\wedge$

B.1.1. *Subtyping*

$$\Gamma \vdash \tau \leq \tau \qquad \text{(SUB-REFL)}$$

$$\frac{\Gamma \vdash \tau_1 \leq \tau_2 \qquad \Gamma \vdash \tau_2 \leq \tau_3}{\Gamma \vdash \tau_1 \leq \tau_3} \qquad \text{(SUB-TRANS)}$$

$$\Gamma \vdash \alpha \leq \Gamma(\alpha) \qquad \text{(SUB-TVAR)}$$

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \sigma_1{\rightarrow}\sigma_2 \leq \tau_1{\rightarrow}\tau_2} \qquad \text{(SUB-ARROW)}$$

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma, \alpha{\leq}\tau_1 \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \forall\alpha{\leq}\sigma_1.\ \sigma_2 \leq \forall\alpha{\leq}\tau_1.\ \tau_2} \qquad \text{(SUB-ALL)}$$

$$\frac{\text{for all } i,\ \Gamma \vdash \sigma \leq \tau_i}{\Gamma \vdash \sigma \leq \bigwedge[\tau_1..\tau_n]} \qquad \text{(SUB-INTER-G)}$$

$$\Gamma \vdash \bigwedge[\tau_1..\tau_n] \leq \tau_i \qquad \text{(SUB-INTER-LB)}$$

$$\Gamma \vdash \bigwedge[\sigma{\rightarrow}\tau_1\ ..\ \sigma{\rightarrow}\tau_n] \leq \sigma \rightarrow \bigwedge[\tau_1..\tau_n] \qquad \text{(SUB-DIST-IA)}$$

$$\Gamma \vdash \bigwedge[\forall\alpha{\leq}\sigma.\ \tau_1 \mathinner{..} \forall\alpha{\leq}\sigma.\ \tau_n] \ \leq\ \forall\alpha{\leq}\sigma.\ \bigwedge[\tau_1..\tau_n] \qquad \text{(Sub-Dist-IQ)}$$

B.1.2. *Typing*

$$\Gamma \vdash x \in \Gamma(x) \qquad \text{(Var)}$$

$$\frac{\Gamma,\, x{:}\tau_1 \vdash e \in \tau_2}{\Gamma \vdash \lambda x{:}\tau_1.\ e \in \tau_1 \rightarrow \tau_2} \qquad \text{(Arrow-I)}$$

$$\frac{\Gamma \vdash e_1 \in \tau_1 \rightarrow \tau_2 \qquad \Gamma \vdash e_2 \in \tau_1}{\Gamma \vdash e_1\ e_2 \in \tau_2} \qquad \text{(Arrow-E)}$$

$$\frac{\Gamma,\, \alpha{\leq}\tau_1 \vdash e \in \tau_2}{\Gamma \vdash \Lambda\alpha{\leq}\tau_1.\ e \in \forall\alpha{\leq}\tau_1.\ \tau_2} \qquad \text{(All-I)}$$

$$\frac{\Gamma \vdash e \in \forall\alpha{\leq}\tau_1.\ \tau_2 \qquad \Gamma \vdash \tau \leq \tau_1}{\Gamma \vdash e[\tau] \in \{\tau/\alpha\}\tau_2} \qquad \text{(All-E)}$$

$$\frac{\Gamma \vdash \{\sigma_i/\alpha\}e \in \tau_i}{\Gamma \vdash \mathit{for}\,\alpha\ \mathit{in}\,\sigma_1..\sigma_n.\ e \in \tau_i} \qquad \text{(For)}$$

$$\frac{\text{for all } i,\ \Gamma \vdash e \in \tau_i}{\Gamma \vdash e \in \bigwedge[\tau_1..\tau_n]} \qquad \text{(Inter-I)}$$

$$\frac{\Gamma \vdash e \in \tau_1 \qquad \Gamma \vdash \tau_1 \leq \tau_2}{\Gamma \vdash e \in \tau_2} \qquad \text{(Sub)}$$

B.1.3. *Syntax-Directed Subtyping*

$$\frac{\text{for all } i,\ \Gamma \vdash \sigma \leq X \Rightarrow \tau_i}{\Gamma \vdash \sigma \leq X \Rightarrow \bigwedge[\tau_1..\tau_n]} \qquad \text{(ASubR-Inter)}$$

$$\frac{\text{for some } i,\ \Gamma \vdash \sigma_i \leq X \Rightarrow \alpha}{\Gamma \vdash \bigwedge[\sigma_1..\sigma_n] \leq X \Rightarrow \alpha} \qquad \text{(ASubL-Inter)}$$

$$\frac{\Gamma \vdash \tau_1 \leq [\,] \Rightarrow \sigma_1 \qquad \Gamma \vdash \sigma_2 \leq X_2 \Rightarrow \alpha}{\Gamma \vdash \sigma_1 \rightarrow \sigma_2 \leq [\tau_1, X_2] \Rightarrow \alpha} \qquad \text{(ASubL-Arrow)}$$

$$\frac{\Gamma \vdash \tau_1 \leq [\,] \Rightarrow \sigma_1 \qquad \Gamma,\, \beta{\leq}\tau_1 \vdash \sigma_2 \leq X_2 \Rightarrow \alpha}{\Gamma \vdash \forall\beta{\leq}\sigma_1.\ \sigma_2 \leq [\beta{\leq}\tau_1, X_2] \Rightarrow \alpha} \qquad \text{(ASubL-All)}$$

$$\Gamma \vdash \alpha \leq [\,] \Rightarrow \alpha \qquad \text{(ASubL-Refl)}$$

$$\frac{\Gamma \vdash \Gamma(\beta) \leq X \Rightarrow \alpha}{\Gamma \vdash \beta \leq X \Rightarrow \alpha} \qquad \text{(ASubL-TVar)}$$

B.1.4. *Type Synthesis*

$$\Gamma \vdash x \in \Gamma(x) \tag{A-Var}$$

$$\frac{\Gamma, x{:}\tau_1 \vdash e \in \tau_2}{\Gamma \vdash \lambda x{:}\tau_1.\ e \in \tau_1 \to \tau_2} \tag{A-Arrow-I}$$

$$\frac{\Gamma \vdash e_1 \in \sigma_1 \qquad \Gamma \vdash e_2 \in \sigma_2}{\Gamma \vdash e_1\ e_2 \in \bigwedge[\psi_i \mid (\phi_i \to \psi_i) \in \mathit{arrowbasis}_\Gamma(\sigma_1) \text{ and } \Gamma \vdash \sigma_2 \leq \phi_i]} \tag{A-Arrow-E}$$

$$\frac{\Gamma, \alpha{\leq}\tau_1 \vdash e \in \tau_2}{\Gamma \vdash \Lambda\alpha{\leq}\tau_1.\ e \in \forall\alpha{\leq}\tau_1.\ \tau_2} \tag{A-All-I}$$

$$\frac{\Gamma \vdash e \in \sigma}{\Gamma \vdash e[\tau] \in \bigwedge[\{\tau/\alpha\}\psi_i \mid (\forall\alpha{\leq}\phi_i.\ \psi_i) \in \mathit{allbasis}_\Gamma(\sigma) \text{ and } \Gamma \vdash \tau \leq \phi_i]} \tag{A-All-E}$$

$$\frac{\text{for all } i,\ \Gamma \vdash \{\sigma_i/\alpha\}e \in \tau_i}{\Gamma \vdash \mathit{for}\ \alpha\ \mathit{in}\ \sigma_1..\sigma_n.\ e \in \bigwedge[\tau_1..\tau_n]} \tag{A-For}$$

B.2. $F_{\leq}$

B.2.1. *Subtyping*

$$\Gamma \vdash \sigma \leq \mathit{Top} \tag{Sub-Top}$$

$$\Gamma \vdash \tau \leq \tau \tag{Sub-Refl}$$

$$\frac{\Gamma \vdash \tau_1 \leq \tau_2 \qquad \Gamma \vdash \tau_2 \leq \tau_3}{\Gamma \vdash \tau_1 \leq \tau_3} \tag{Sub-Trans}$$

$$\Gamma \vdash \alpha \leq \Gamma(\alpha) \tag{Sub-TVar}$$

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \sigma_1 \to \sigma_2 \leq \tau_1 \to \tau_2} \tag{Sub-Arrow}$$

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma, \alpha{\leq}\tau_1 \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \forall\alpha{\leq}\sigma_1.\ \sigma_2 \leq \forall\alpha{\leq}\tau_1.\ \tau_2} \tag{Sub-All}$$

B.2.2. *Typing*

$$\Gamma \vdash x \in \Gamma(x) \tag{Var}$$

$$\frac{\Gamma, x{:}\tau_1 \vdash e \in \tau_2}{\Gamma \vdash \lambda x{:}\tau_1.\ e \in \tau_1 \to \tau_2} \tag{Arrow-I}$$

$$\frac{\Gamma \vdash e_1 \in \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 \in \tau_1}{\Gamma \vdash e_1\ e_2 \in \tau_2} \tag{Arrow-E}$$

$$\frac{\Gamma, \alpha{\leq}\tau_1 \vdash e \in \tau_2}{\Gamma \vdash \Lambda\alpha{\leq}\tau_1.\ e \in \forall\alpha{\leq}\tau_1.\ \tau_2} \tag{All-I}$$

$$\frac{\Gamma \vdash e \in \forall\alpha{\leq}\tau_1.\ \tau_2 \qquad \Gamma \vdash \tau \leq \tau_1}{\Gamma \vdash e[\tau] \in \{\tau/\alpha\}\tau_2} \tag{All-E}$$

$$\frac{\Gamma \vdash e \in \tau_1 \qquad \Gamma \vdash \tau_1 \leq \tau_2}{\Gamma \vdash e \in \tau_2} \tag{Sub}$$

B.3. $\lambda_\wedge$

B.3.1. *Subtyping*

$$\frac{\Gamma \vdash \rho_1 \leq_P \rho_2}{\Gamma \vdash \rho_1 \leq \rho_2} \tag{Sub-Prim}$$

$$\Gamma \vdash \tau \leq \tau \tag{Sub-Refl}$$

$$\frac{\Gamma \vdash \tau_1 \leq \tau_2 \qquad \Gamma \vdash \tau_2 \leq \tau_3}{\Gamma \vdash \tau_1 \leq \tau_3} \tag{Sub-Trans}$$

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \sigma_1 \to \sigma_2 \leq \tau_1 \to \tau_2} \tag{Sub-Arrow}$$

$$\frac{\text{for all } i,\ \Gamma \vdash \sigma \leq \tau_i}{\Gamma \vdash \sigma \leq \bigwedge[\tau_1..\tau_n]} \tag{Sub-Inter-G}$$

$$\Gamma \vdash \bigwedge[\tau_1..\tau_n] \leq \tau_i \tag{Sub-Inter-LB}$$

$$\Gamma \vdash \bigwedge[\sigma \to \tau_1 .. \sigma \to \tau_n] \leq \sigma \to \bigwedge[\tau_1..\tau_n] \tag{Sub-Dist-IA}$$

B.3.2. *Typing*

$$\Gamma \vdash x \in \Gamma(x) \tag{Var}$$

$$\frac{\Gamma, x{:}\sigma_i \vdash e \in \tau_i}{\Gamma \vdash \lambda x{:}\sigma_1..\sigma_n.\ e \in \sigma_i \to \tau_i} \tag{Arrow-I'}$$

$$\frac{\Gamma \vdash e_1 \in \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 \in \tau_1}{\Gamma \vdash e_1\, e_2 \in \tau_2} \tag{Arrow-E}$$

$$\frac{\text{for all } i,\ \Gamma \vdash e \in \tau_i}{\Gamma \vdash e \in \bigwedge[\tau_1..\tau_n]} \tag{Inter-I}$$

$$\frac{\Gamma \vdash e \in \tau_1 \qquad \Gamma \vdash \tau_1 \leq \tau_2}{\Gamma \vdash e \in \tau_2} \tag{Sub}$$

Appendix. References

Abadi, M., Cardelli, L., & Viswanathan, R. 1996. An Interpretation of Objects and Object Types. *Pages 396–409 of: Principles of Programming Languages.*

Barbanera, F., & Dezani-Ciancaglini, M. 1991. Intersection and Union Types. *In:* (Ito & Meyer, 1991).

Barbanera, F., Dezani-Ciancaglini, M., & de'Liguoro, U. 1995. Intersection and Union Types: Syntax and Semantics. *Information and Computation*, **119**(2), 202–230.

Barendregt, H., Coppo, M., & Dezani-Ciancaglini, M. 1983. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, **48**(4), 931–940.

Barendregt, H. P. 1984. *The Lambda Calculus.* Revised edn. North Holland.

Böhm, C., & Berarducci, A. 1985. Automatic Synthesis of Typed Λ-Programs on Term Algebras. *Theoretical Computer Science*, **39**, 135–154.

Breazu-Tannen, V., Coquand, T., Gunter, C., & Scedrov, A. 1991. Inheritance as Implicit Coercion. *Information and Computation*, **93**, 172–221. Also in Carl A. Gunter and John C. Mitchell, editors, *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design* (MIT Press, 1994).

Bruce, K., & Mitchell, J. 1992 (Jan.). PER models of subtyping, recursive types and higher-order polymorphism. *In: Proceedings of the Nineteenth ACM Symposium on Principles of Programming Languages.*

Bruce, K. B. 1991 (Mar.). The Equivalence of Two Semantic Definitions for Inheritance in Object-Oriented Languages. *In: Proceedings of Mathematical Foundations of Programming Semantics.*

Bruce, K. B. 1994. A Paradigmatic Object-Oriented Programming Language: Design, Static Typing and Semantics. *Journal of Functional Programming*, **4**(2). A preliminary version appeared in POPL 1993 under the title "Safe Type Checking in a Statically Typed Object-Oriented Programming Language".

Bruce, K. B., & Longo, G. 1990. A Modest Model of Records, Inheritance, and Bounded Quantification. *Information and Computation*, **87**, 196–240. Also in Carl A. Gunter and John C. Mitchell, editors, *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design* (MIT Press, 1994). An earlier version appeared in the proceedings of the IEEE Symposium on Logic in Computer Science, 1988.

Bruce, K. B., Meyer, A. R., & Mitchell, J. C. 1990. The Semantics of Second-Order Lambda Calculus. *In:* (Huet, 1990). Also appeared in *Information and Computation* 84, 1 (January 1990).

Bruce, K. B., Schuett, A., & van Gent, R. 1995. PolyTOIL: A Type-Safe Polymorphic Object-Oriented Language. *Pages 27–51 of:* Olthoff, W. (ed), *Proceedings of* ECOOP '95. LNCS 952. Aarhus, Denmark: Springer-Verlag.

Canning, P., Hill, W., & Olthoff, W. 1988. *A Kernel Language for Object-Oriented Programming.* Tech. rept. STL-88-21. Hewlett-Packard Labs.

Canning, P., Cook, W., Hill, W., Olthoff, W., & Mitchell, J. 1989a (Sept.). F-Bounded Quantification for Object-Oriented Programming. *Pages 273–280 of: Fourth International Conference on Functional Programming Languages and Computer Architecture.*

Canning, P., Cook, W., Hill, W., & Olthoff, W. 1989b. Interfaces for Strongly-Typed Object-Oriented Programming. *Pages 457–467 of: Object Oriented Programing: Systems, Languages, and Applications (OOPSLA).*

Cardelli, L. 1984. A semantics of multiple inheritance. *Pages 51–67 of:* Kahn, G., MacQueen, D., & Plotkin, G. (eds), *Semantics of Data Types.* Lecture Notes in Computer Science, vol. 173. Springer-Verlag. Full version in *Information and Computation* 76(2/3):138–164, 1988.

Cardelli, L. 1988 (Jan.). Structural Subtyping and the Notion of Power Type. *Pages 70–79 of: Proceedings of the 15th ACM Symposium on Principles of Programming Languages.*

Cardelli, L. 1991. Typeful Programming. *In:* Neuhold, E. J., & Paul, M. (eds), *Formal Description of Programming Concepts.* Springer-Verlag. An earlier version appeared as DEC Systems Research Center Research Report #45, February 1989.

Cardelli, L. 1992 (Jan.). *Extensible Records in a Pure Calculus of Subtyping.* Research report 81. DEC Systems Research Center. Also in Carl A. Gunter and John C. Mitchell, editors, *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design* (MIT Press, 1994).

Cardelli, L. 1993 (Feb.). *An Implementation of $F_{<:}$.* Research report 97. DEC Systems Research Center.

Cardelli, L., & Longo, G. 1991. A semantic basis for Quest. *Journal of Functional Programming*, **1**(4), 417–458. Preliminary version in ACM Conference on Lisp and Functional Programming, June 1990. Also available as DEC SRC Research Report 55, Feb. 1990.

Cardelli, L., & Mitchell, J. 1991. Operations on Records. *Mathematical Structures in Computer Science*, **1**, 3–48. Also in Carl A. Gunter and John C. Mitchell, editors, *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design* (MIT Press, 1994); available as DEC Systems Research Center Research Report #48, August, 1989, and in the proceedings of MFPS '89, Springer LNCS volume 442.

Cardelli, L., & Wegner, P. 1985. On Understanding Types, Data Abstraction, and Polymorphism. *Computing Surveys*, **17**(4).

Cardelli, L., Martini, S., Mitchell, J. C., & Scedrov, A. 1994. An Extension of System F with Subtyping. *Information and Computation*, **109**(1–2), 4–56. A preliminary version appeared in TACS '91 (Sendai, Japan, pp. 750–770).

Cardone, F., Dezani-Ciancaglini, M., & de' Liguoro, U. 1994. Combining type disciplines. *Annals of Pure and Applied Logic*, **66**, 197–230.

Cardone, F. 1989. Relational Semantics for Recursive Types and Bounded Quantification. *Pages 164–178 of: Proceedings of the Sixteenth International Colloquium on Automata, Languages, and Programming.* Lecture Notes in Computer Science, vol. 372. Stresa, Italy: Springer-Verlag.

Cardone, F., & Coppo, M. 1990. Two Extensions of Curry's Type Inference System. *Pages 19–76 of:* Odifreddi, P. (ed), *Logic and Computer Science.* APIC Studies in Data Processing, no. 31. Academic Press.

Cartwright, R., & Fagan, M. 1991 (June). Soft typing. *Pages 278–292 of: Proceedings of the SIGPLAN '91 Conference on Programming Language Design and Implementation.* Also available as Sigplan Notices 26(6) June 1991.

Church, A. 1940. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, **5**, 56–68.

Compagnoni, A. B. 1994 (Sept.). Decidability of Higher-Order Subtyping with Intersection Types. *In: Computer Science Logic.* Kazimierz, Poland. Springer *Lecture Notes in Computer Science* 933, June 1995. Also available as University of Edinburgh, LFCS technical report ECS-LFCS-94-281, titled "Subtyping in $F^{\omega}_{\wedge}$ is decidable".

Compagnoni, A. B. 1995 (Jan.). *Higher-Order Subtyping with Intersection Types.* Ph.D. thesis, Catholic University, Nigmegen.

Compagnoni, A. B., & Pierce, B. C. 1996. Multiple Inheritance via Intersection Types. *Mathematical Structures in Computer Science.* To appear. Preliminary version available as University of Edinburgh technical report ECS-LFCS-93-275 and Catholic University Nijmegen computer science technical report 93-18, Aug. 1993.

Cook, W. R., Hill, W. L., & Canning, P. S. 1990 (Jan.). Inheritance is not Subtyping. *Pages 125–135 of: Seventeenth Annual ACM Symposium on Principles of Programming Languages.*

Also in Carl A. Gunter and John C. Mitchell, editors, *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design* (MIT Press, 1994).

Coppo, M., & Dezani-Ciancaglini, M. 1978. A New Type-Assignment for λ-Terms. *Archiv. Math. Logik*, **19**, 139–156.

Coppo, M., & Dezani-Ciancaglini, M. 1980. An Extension of the Basic Functionality Theory for the λ-Calculus. *Notre-Dame Journal of Formal Logic*, **21**(4), 685–693.

Coppo, M., Dezani, M., & Sallé, P. 1979. Functional characterization of some semantic equalities inside λ-calculus. Lecture Notes in Computer Science, no. 81. Springer-Verlag.

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

Coppo, M., Dezani-Ciancaglini, M., & Venneri, B. 1981. Functional Characters of Solvable Terms. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, **27**, 45–58.

Coppo, M., Dezani-Ciancaglini, M., Honsell, F., & Longo, G. 1983. Extended Type Structures and Filter Lambda Models. *Pages 241–262 of:* Lolli, G., Longo, G., & Marja, A. (eds), *Logic Colloquium 82*. Amsterdam: North-Holland.

Coppo, M., Dezani-Ciancaglini, M., & Zacchi, M. 1987. Type Theories, Normal Forms and D_∞-Lambda-Models. *Information and Computation*, **72**, 85–116.

Coppo, M., & Giannini, P. 1995. Principal Types and Unification for a Simple Intersection Type Systems. *Information and Computation*, **122**(1), 70–96.

Curien, P.-L., & Ghelli, G. 1991. Subtyping + extensionality: Confluence of $\beta\eta$-reductions in $F_\leq$. *In:* (Ito & Meyer, 1991).

Curien, P.-L., & Ghelli, G. 1992. Coherence of Subsumption: Minimum typing and type-checking in $F_\leq$. *Mathematical Structures in Computer Science*, **2**, 55–91. Also in Carl A. Gunter and John C. Mitchell, editors, *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design* (MIT Press, 1994).

Curry, H. B., & Feys, R. 1958. *Combinatory Logic.* Vol. 1. North Holland.

de Bruijn, N. G. 1972. Lambda-Calculus Notation with Nameless Dummies: a Tool for Automatic Formula Manipulation with Application to the Church-Rosser Theorem. *Indag. Math.*, **34**(5), 381–392.

Dezani-Ciancaglini, M., & Margaria, I. 1984. F-Semantics for Intersection Type Discipline. *Pages 279–300 of:* Kahn, G., MacQueen, D. B., & Plotkin, G. (eds), *Semantics of Data Types*. Lecture Notes in Computer Science, vol. 173. Springer-Verlag.

Dezani-Ciancaglini, M., & Margaria, I. 1986. A Characterisation of F-Complete Type Assignments. *Theoretical Computer Science*, **45**, 121–157.

Fagan, M. 1990 (Dec.). *Soft Typing: An Approach to Type Checking for Dynamically Typed Languages.* Ph.D. thesis, Rice University.

Freeman, T., & Pfenning, F. 1991. Refinement Types for ML. *In: Proceedings of the SIGPLAN '91 Symposium on Language Design and Implementation, Toronto, Ontario.* ACM Press.

Freyd, P., Mulry, P., Rosolini, G., & Scott, D. 1990. Extensional PERs. *Pages 346–354 of: Fifth Annual Symposium on Logic in Computer Science (Philadelphia, PA).* IEEE Computer Society Press.

Ghelli, G. 1993. Recursive types are not conservative over $F_{\leq}$. *Pages 146–162 of:* Bezen, M., & Groote, J. (eds), *Proceedings of thethe International Conference on Typed Lambda Calculi and Applications (TLCA), Utrecht, The Netherlands.* Lecture Notes in Computer Science, no. 664. Berlin: Springer Verlag.

Ghelli, G. 1990 (Mar.). *Proof Theoretic Studies about a Minimal Type System Integrating Inclusion and Parametric Polymorphism.* Ph.D. thesis, Università di Pisa. Technical report TD–6/90, Dipartimento di Informatica, Università di Pisa.

Ghelli, G. 1995. Divergence of $F_{\leq}$ Type Checking. *Theoretical Computer Science*, **139**(1,2), 131–162.

Girard, J.-Y. 1972. *Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur.* Ph.D. thesis, Université Paris VII.

Gunter, C. A. 1992. *Semantics of Programming Languages: Structures and Techniques.* Cambridge, MA: The MIT Press.

Hayashi, S. 1991. Singleton, Union and Intersection Types for Program Extraction. *In:* (Ito & Meyer, 1991). Full version in *Information and Computation*, 109(1/2):174-210, 1994.

Hindley, J. R. 1982. The Simple Semantics for Coppo-Dezani-Sallé Types. *Pages 212–226 of:* Dezani-Ciancaglini, & Montanari (eds), *Proceedings of the International Symposium on Programming.* Springer-Verlag. Lecture Notes in Computer Science No. 137.

Hindley, J. R., & Seldin, J. P. 1986. *Introduction to Combinators and λ-Calculus.* London Mathematical Society Student Texts, vol. 1. Cambridge University Press.

Hofmann, M., & Pierce, B. 1995. A Unifying Type-Theoretic Framework for Objects. *Journal of Functional Programming.* Previous versions appeared in the Symposium on Theoretical Aspects of Computer Science, 1994, (pages 251–262) and, under the title "An Abstract View of Objects and Subtyping (Preliminary Report)," as University of Edinburgh, LFCS technical report ECS-LFCS-92-226, 1992.

Huet, G. (ed). 1990. *Logical Foundations of Functional Programming.* University of Texas at Austin Year of Programming Series. Addison-Wesley.

Ito, T., & Meyer, A. R. (eds). 1991. *Theoretical Aspects of Computer Software (Sendai, Japan).* Lecture Notes in Computer Science, no. 526. Springer-Verlag.

Jim, T. 1996. What are principal typings and what are they good for? *Pages 42–53 of:* Principles of Programming Languages.

Katiyar, D., Luckham, D., & Mitchell, J. 1994 (Jan.). A Type System for Prototyping Languages. *Pages 138–150 of: Conference Record of POPL '94: 21st ACM SIGPLAN–SIGACT Symposium of Principles of Programming Languages, Portland, Oregon.* Association for Computing Machinery.

Leivant, D. 1990. Discrete Polymorphism (Summary). *Pages 288–297 of: Proceedings of the 1990 ACM Conference on Lisp and Functional Programming.*

Martini, S. 1988. Bounded Quantifiers have Interval Models. *Pages 174–183 of: Proceedings of the ACM Conference on Lisp and Functional Programming.* Snowbird, Utah: ACM.

Mitchell, J., Meldal, S., & Madhav, N. 1991 (Jan.). An extension of Standard ML modules with subtyping and inheritance. *Pages 270–278 of: Proceedings of the Eighteenth ACM Symposium on Principles of Programming Languages.*

Mitchell, J. C. 1984. Type inference and type containment. *Pages 257–278 of: Proc. Int. Symp. on Semantics of Data Types, Sophia-Antipolis (France).* Berlin: Springer LNCS 173. Full version in *Information and Computation*, vol. 76, no. 2/3, 1988, pp. 211–249. Reprinted in *Logical Foundations of Functional Programming,* ed. G. Huet, Addison-Wesley (1990) 153–194.

Mitchell, J. C. 1990. A Type-Inference Approach to Reduction Properties and Semantics of Polymorphic Expressions. *In:* (Huet, 1990).

Pfenning, F. 1993 (May). Refinement Types for Logical Frameworks. *Pages 315–328 of: Informal Proceedings of the 1993 Workshop on Types for Proofs and Programs.*

Pierce, B., & Steffen, M. 1996. Higher-Order Subtyping. *Theoretical Computer Science.* To appear. A preliminary version appeared in IFIP Working Conference on Programming Concepts, Methods and Calculi (PROCOMET), June 1994, and as University of Edinburgh technical report ECS-LFCS-94-280 and Universität Erlangen-Nürnberg Interner Bericht IMMD7-01/94, January 1994.

Pierce, B. C. 1991a (December). *Programming with Intersection Types and Bounded Polymorphism.* Ph.D. thesis, Carnegie Mellon University. Available as School of Computer Science technical report CMU-CS-91-205.

Pierce, B. C. 1991b (Feb.). *Programming with Intersection Types, Union Types, and Polymorphism.* Technical Report CMU-CS-91-106. Carnegie Mellon University.

Pierce, B. C. 1994. Bounded Quantification is Undecidable. *Information and Computation*, **112**(1), 131–165. Also in Carl A. Gunter and John C. Mitchell, editors, *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design* (MIT Press, 1994). A preliminary version appeared in POPL '92.

Pierce, B. C., & Turner, D. N. 1994. Simple Type-Theoretic Foundations for Object-Oriented Programming. *Journal of Functional Programming*, **4**(2), 207–247. A preliminary version appeared in Principles of Programming Languages, 1993, and as University of Edinburgh technical report ECS-LFCS-92-225, under the title "Object-Oriented Programming Without Recursive Types".

Pottinger, G. 1980. A Type Assignment for the Strongly Normalizable λ-Terms. *Pages 561–577 of: To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism.* New York: Academic Press.

Reynolds, J. C. 1981. The Essence of Algol. *Pages 345–372 of:* de Bakker, J. W., & van Vliet, J. C. (eds), *Algorithmic Languages.* Amsterdam: North-Holland.

Reynolds, J. 1974. Towards a Theory of Type Structure. *Pages 408–425 of: Proc. Colloque sur la Programmation.* New York: Springer-Verlag LNCS 19.

Reynolds, J. C. 1988 (June). *Preliminary Design of the Programming Language Forsythe.* Tech. rept. CMU-CS-88-159. Carnegie Mellon University.

Reynolds, J. C. 1991. The Coherence of Languages with Intersection Types. *In:* (Ito & Meyer, 1991).

Ronchi della Rocca, S., & Venneri, B. 1984. Principal Type Schemes for an Extended Type Theory. *Theoretical Computer Science*, **28**, 151–169.

Ronchi della Rocca, S. 1988. Principal Type Scheme and Unification for Intersection Type Discipline. *Theoretical Computer Science*, **59**, 181–209.

Sallé, P. 1982. Une extension de la theorie des types en λ-calcul. Springer-Verlag. Lecture Notes in Computer Science No. 62.

Scott, D. 1976. Data Types as Lattices. *SIAM Journal on Computing*, **5**(3), 522–587.

van Bakel, S. 1991. *Principal type schemes for the strict type assignment system.* Technical report 91-6. University of Nijmegen.

van Bakel, S. 1992. Complete restrictions of the intersection type discipline. *Theoretical Computer Science*, **99**.

Wright, A. K., & Cartwright, R. 1994. A Practical Soft Type System for Scheme. *In: Proceedings of the 1994 ACM Conference on Lisp and Functional Programming.* ACM. Also available as LISP Pointers VII(3) July-September 1994.