Greedy Bidirectional Polymorphism

Jana Dunfield
McGill University, Montréal, Canada

Abstract

Bidirectional typechecking has become popular in advanced type systems because it works in many situations where inference is undecidable. In this paper, I show how to cleanly handle parametric polymorphism in a bidirectional setting. The key contribution is a bidirectional type system for a subset of ML that supports first-class (higher-rank and even impredicative) polymorphism, and is complete for predicative polymorphism (including ML-style polymorphism and higher-rank polymorphism). The system's power comes from bidirectionality combined with a "greedy" method of finding polymorphic instances inspired by Cardelli's early work on System $F_{<:}$. This work demonstrates that bidirectionality is a good foundation for traditionally vexing features like first-class polymorphism.

Categories and Subject Descriptors D.3.3 [*Programming Languages*]: Language Constructs and Features—Polymorphism

General Terms languages

1. Introduction

To check programs in advanced type systems, it is often useful to split the traditional typing judgment $e : A$ into two forms, $e \Uparrow A$ read "e synthesizes type A" and $e \Downarrow A$ read "e checks against type A". This technique has been used for dependent types (Coquand 1996; Norell 2007; Abel et al. 2008; Löh et al. 2008); subtyping (Pierce and Turner 2000; Odersky et al. 2001); intersection, union, indexed and refinement types (Xi 1998; Davies and Pfenning 2000; Dunfield and Pfenning 2004); termination checking (Abel 2004); higher-rank polymorphism (Peyton Jones et al. 2007); refinement types for LF (Lovas and Pfenning 2007); contextual modal types (Pientka 2008; Pientka and Dunfield 2008); and compiler intermediate representations (Chlipala et al. 2005).

Bidirectional typechecking is *necessary* because annotation-free type inference, which works well for the lambda calculus with prenex polymorphism, becomes difficult (if not undecidable) when we add first-class polymorphism, subtyping, intersection types, and so forth. Bidirectional typechecking is *nice* because reports of type errors are better localized, which is useful even when type inference is feasible.

In earlier work, we gave a concise recipe for bidirectional typechecking (Dunfield and Pfenning 2004), in which annotations are needed exactly where redexes appear. But we left out a vital feature: parametric polymorphism. So what are the proper *bidirectional* introduction and elimination rules for parametric polymorphism? It turns out that the introduction rule is easy, but the elimination rule is hard. For example, if we have a polymorphic function *choose* : $\forall\alpha.\, \alpha \to \alpha \to \alpha$, to find the right instantiation of α in the application *choose* $x\ y$ we must look at x's type (and, for certain mixes of type system features, y's type as well). Clearly, we do not know how to instantiate α from the term *choose* alone.

How can we find polymorphic instances in a bidirectional type system that is simple to formulate and use—without a heavy type annotation burden? I adapt an idea of Cardelli (1993), *greed*: the *first* constraint on a type variable determines the instantiation. For *choose* $x\ y$, this means α is determined by the type of x.

In this paper, I show how to use greed to find polymorphic instances in System F (Girard 1986; Reynolds 1974), where polymorphism is first-class (higher-rank and impredicative). This yields a remarkably simple algorithm that is complete for predicative polymorphism (including ML-style prenex polymorphism). That is, if a typing derivation exists that instantiates type variables at monomorphic types, the user gives no more information than the annotations already present (on redexes) if there were no polymorphism. The algorithm handles some uses of impredicative polymorphism, where type variables are instantiated with polymorphic types, without extra help; for the rest, I provide a "hint" mechanism. Using intersection and union types, the approach can even handle subtyping, as described elsewhere (Dunfield 2009).

This paper shows that first-class polymorphism, while often tricky with type *inference*, is manageable in bidirectional type*checking*. Rather than starting with Damas-Milner inference, perhaps eventually trying to glue on some bidirectionality for the season's latest type features, we get simplicity and power by making things bidirectional from the ground up.

I will begin by giving a point of reference: a bidirectional type system that assumes polymorphic instances are found magically (Section 2). Section 3 develops a decidable version of that system and shows that it is complete, with respect to the Section 2 system, for typing derivations that use only predicative polymorphism. Section 4 adds datatypes, Section 5 briefly sketches subtyping, and Section 6 explains the implementation.

2. System Bi

System Bi is a very simple bidirectional type system with first-class polymorphism. It does not touch the problem of finding polymorphic instances; that is left to System $\text{Bi}^{\hat\alpha}$ ("bi ex"), described in the next section. But it is a good reference point for proving things about System $\text{Bi}^{\hat\alpha}$.

Figure 1 gives the syntax of terms, types, etc. For simplicity, I omit some constructs such as fixed point recursion $\mathbf{fix}\, u.\, e$, which is easy to handle as in previous work (Dunfield and Pfenning 2004). We'll also gloss over datatypes $\vec{A}\ \delta$ where δ is the name of an n-argument inductive datatype and $\vec{A}$ is a sequence of n types. For example, given a base type int and the one-argument datatype list, we can write int list. Term-level data constructors have *constructor*

ML '09, August 30, 2009, Edinburgh, Scotland, UK.

www.cs.queensu.ca/~jana/papers/poly/

$$
\begin{array}{rrl}
\text{Type variables} & & \alpha, \beta \\
\text{Atomic types} & A^{\textit{atomic}} & ::= \mathbf{1} \mid \alpha \mid \forall\alpha.\, A \\
\text{Types} & A, B, C & ::= A^{\textit{atomic}} \mid A \to B \\
\text{Contexts} & \Gamma & ::= \cdot \mid \Gamma, x{:}A \mid \Gamma, \alpha \\
\text{Matches} & ms & ::= \cdot \mid c(x) \Rightarrow e \,\|\, ms \\
\text{Annotations} & N & ::= (\Gamma \vdash A) \\
\text{Terms} & e & ::= x \mid () \mid \lambda x.\, e \mid e_1\, e_2 \\
 & & \mid c(e) \mid \textbf{case}\ e\ \textbf{of}\ ms \mid (e : N) \\
\text{Values} & v & ::= x \mid () \mid \lambda x.\, e \mid c(v) \mid (v : N) \\
\text{Evaluation contexts} & \mathcal{E} & ::= [\,] \mid \mathcal{E}\, e \mid v\, \mathcal{E} \mid c(\mathcal{E}) \mid \textbf{case}\ \mathcal{E}\ \textbf{of}\ ms
\end{array}
$$

$$
\frac{e' \ \mapsto_R \ e''}{\mathcal{E}[e'] \ \mapsto \ \mathcal{E}[e'']}
\qquad
\begin{array}{rcl}
(\lambda x.\, e)\, v & \mapsto_R & [v/x]e \\
\textbf{case}\ c(v)\ \textbf{of}\ \ldots c(x) \Rightarrow e \ldots & \mapsto_R & [v/x]e
\end{array}
$$

Figure 1: Grammar and operational semantics for System Bi

type $B \to \vec{\alpha}\ \delta$—no GADTs here. Datatypes are not particularly interesting in System Bi; while we give the syntax of case arms (matches ms) and constructors $c(e)$, we omit the typing rules (but see Section 4).

The SML-like operational semantics (defined under type erasure) is straightforward, making use of evaluation contexts; $\mathcal{E}[e']$ is a term with e' in evaluation position.

Figure 2 has the rules for well-formedness of types and contexts. In general, we assume every context we write is well-formed, but tend to explicitly say when individual types are well-formed.

The bidirectional typing judgments are $\Gamma \vdash e \Uparrow A$, read "$e$ synthesizes A", and $\Gamma \vdash e \Downarrow B$, read "$e$ checks against B". (The arrows correspond to the flow of type information in an abstract syntax tree representation of e.) Figure 3 gives the typing rules. Introduction and elimination rules follow the pattern we introduced (Dunfield and Pfenning 2004): the conclusion of an introduction rule is checked against a given type, and the premise of an elimination rule—where the type being eliminated appears—synthesizes a type. This yields the smallest sensible set of rules, and means that annotations are needed only on redexes (including declarations of recursive functions), but not on function calls (except calls of the form $(\lambda x.\, e)e'$, which are redexes).

The rule sub expresses a subsumption principle: if e synthesizes a type A that is at least as polymorphic as B—the known type that e is being checked against—then an A can be used where a B is expected. For example, a function of type $\forall\alpha.\, \alpha \to \alpha$ can be passed to a function expecting $\mathsf{int} \to \mathsf{int}$. We write this limited form of subtyping as $\Gamma \vdash A \leqq B$.

The rule anno is read as "if $N = (\Gamma' \vdash A')$ matches the typing $(\Gamma \vdash A)$ and e checks against A, then $(e : (\Gamma' \vdash A'))$ checks against A". The relation $\lesssim$ handles the renaming between Γ and Γ', and between A and A'. These *contextual annotations* are discussed below.

1I introduces the unit constructor $()$, so it is a checking rule. A philosophical point lurks: $()$ is the introduction form for $\mathbf{1}$, so the rule for it should check. But $()$ is also like a constant or predefined function, so it would be reasonable to make $\mathbf{1}$ a synthesis rule.

$\forall$I introduces a universal quantifier—with a value restriction, since I'm interested primarily in call-by-value languages with side effects. $\forall$E is an "oracular" elimination rule; it assumes someone has revealed to us the instance A'. Of course this is not practical—indeed, it begs the question this paper is supposed to answer—and we will address this in System Bi$^{\hat{\alpha}}$.

Figure 3 gives the rules for the limited subtyping used in sub. Again, we defer the rules for datatypes to Section 4. In the rule $\forall$L$\leqq$, we write $[A'/\alpha]A$ to mean the substitution of A' for α in the type A. Following Dunfield and Pfenning (2003), reflexivity and transitivity are admissible and so need no explicit rules. For

$$
\boxed{\Gamma \vdash A\ \mathsf{wf}}
\qquad
\boxed{\Gamma\ \mathsf{wf}}
$$

$$
\frac{FV(A) \subseteq \mathsf{dom}(\Gamma)}{\Gamma \vdash A\ \mathsf{wf}}
\qquad
\frac{}{\cdot\ \mathsf{wf}}
\qquad
\frac{\Gamma\ \mathsf{wf} \qquad x \notin \mathsf{dom}(\Gamma) \qquad \Gamma \vdash A\ \mathsf{wf}}{\Gamma, x{:}A\ \mathsf{wf}}
$$

Figure 2: Well-formedness of types and contexts

$\boxed{\Gamma \vdash e \Downarrow A}$ e checks against type A

$\boxed{\Gamma \vdash e \Uparrow A}$ e synthesizes type A

$$
\frac{\Gamma(x) = A}{\Gamma \vdash x \Uparrow A}\ \mathsf{var}
\qquad
\frac{\Gamma \vdash e \Uparrow A \qquad \Gamma \vdash A \leqq B}{\Gamma \vdash e \Downarrow B}\ \mathsf{sub}
$$

$$
\frac{N \lesssim (\Gamma \vdash A) \qquad \Gamma \vdash e \Downarrow A}{\Gamma \vdash (e : N) \Uparrow A}\ \mathsf{anno}
\qquad
\frac{}{\Gamma \vdash () \Downarrow \mathbf{1}}\ \mathbf{1}\mathrm{I}
$$

$$
\frac{\Gamma, x{:}A \vdash e \Downarrow B}{\Gamma \vdash \lambda x.\, e \Downarrow A \to B} \to\mathrm{I}
\qquad
\frac{\Gamma \vdash e_1 \Uparrow A \to B \qquad \Gamma \vdash e_2 \Downarrow A}{\Gamma \vdash e_1\, e_2 \Uparrow B} \to\mathrm{E}
$$

$$
\frac{\Gamma, \alpha \vdash v \Downarrow A}{\Gamma \vdash v \Downarrow \forall\alpha.\, A}\ \forall\mathrm{I}
\qquad
\frac{\Gamma \vdash e \Uparrow \forall\alpha.\, A \qquad \Gamma \vdash A'\ \mathsf{wf}}{\Gamma \vdash e \Uparrow [A'/\alpha]\, A}\ \forall\mathrm{E}
$$

$\boxed{\Gamma \vdash A \leqq B}$ A is at least as polymorphic as B

$$
\frac{}{\Gamma \vdash \mathbf{1} \leqq \mathbf{1}}\ \mathbf{1}{\leqq}
\qquad
\frac{\Gamma \vdash B_1 \leqq A_1 \qquad \Gamma \vdash A_2 \leqq B_2}{\Gamma \vdash A_1 \to A_2 \leqq B_1 \to B_2} \to{\leqq}
$$

$$
\frac{}{\Gamma \vdash \alpha \leqq \alpha}\ \alpha\mathrm{Refl}{\leqq}
$$

$$
\frac{\Gamma \vdash [A'/\alpha]A \leqq B \qquad \Gamma \vdash A'\ \mathsf{wf}}{\Gamma \vdash \forall\alpha.\, A \leqq B}\ \forall\mathrm{L}{\leqq}
\qquad
\frac{\Gamma, \beta \vdash A \leqq B}{\Gamma \vdash A \leqq \forall\beta.\, B}\ \forall\mathrm{R}{\leqq}
$$

Figure 3: Typing and subtyping in System Bi

$$
\frac{}{(\cdot \vdash A) \lesssim (\Gamma \vdash A)}\ {\lesssim}\text{-empty}
$$

$$
\frac{\Gamma(x) \equiv B_0 \qquad (\Gamma_0 \vdash A_0) \lesssim (\Gamma \vdash A)}{(x{:}B_0, \Gamma_0 \vdash A_0) \lesssim (\Gamma \vdash A)}\ {\lesssim}\text{-pvar}
$$

$$
\frac{\Gamma \vdash \alpha'\ \mathsf{wf} \qquad ([\alpha'/\alpha]\Gamma_0 \vdash [\alpha'/\alpha]A_0) \lesssim (\Gamma \vdash A)}{(\alpha, \Gamma_0 \vdash A_0) \lesssim (\Gamma \vdash A)}\ {\lesssim}\text{-tyvar}
$$

Figure 4: Contextual matching, used in annotations and hints

example, $\Gamma \vdash \forall\alpha.\, A \leqq \forall\beta.\, [\beta/\alpha]A$—which is the same as $\Gamma \vdash \forall\alpha.\, A \leqq \forall\alpha.\, A$—is derivable by (1) deriving $\Gamma, \beta \vdash [\beta/\alpha]A \leqq [\beta/\alpha]A$; (2) applying $\forall$L$\leqq$, giving $\Gamma, \beta \vdash \forall\alpha.\, A \leqq [\beta/\alpha]A$; (3) applying $\forall$R$\leqq$. (To prove transitivity, measure the derivations by the lexicographic ordering of (1) the number of $\forall$L$\leqq$ applications in the *second* derivation, with (2) the height of both derivations; this makes the $\forall$R$\leqq$/$\forall$L$\leqq$ case work.)

2.1 Contextual annotations

Annotations are *contextual* (Dunfield and Pfenning 2004): when checking $(e : (\Gamma' \vdash A'))$ under the context Γ, the context Γ' establishes the relationship between type variables declared in Γ and type variables used in A', the annotated type of e. For example, the following fragment uses the type of x to establish that the α in the inner annotation (on $\lambda y.\, \mathsf{Cons}(y, \mathsf{Nil})$) is the same as the α used in the outer annotation. The type variable α is bound by Γ', and its scope is $x{:}\alpha \vdash \ldots$, but the program variable x in $x{:}\alpha$ is in the scope of λx.

$$\Big(\lambda x.\, \lambda n.\, \ldots \\ \big((\lambda y.\, \mathsf{Cons}(y, \mathsf{Nil})) : (\alpha, x{:}\alpha \vdash \alpha \to \alpha\ \mathsf{list})\big) \ldots\Big) \\ : \forall \alpha.\, \alpha \to \mathsf{int} \to \alpha\ \mathsf{list}$$

This avoids having a term-level binder for type variables. Allowing something like

$$(\lambda x.\, \lambda n.\, \ldots (e : \underline{\underline{\alpha}}) \ldots) \ : \ \forall \alpha.\, \alpha \to \mathsf{int} \to \alpha\ \mathsf{list}$$

does not sit well: the underlined α is not within the most natural scope of α, which is simply $\alpha \to \mathsf{int} \to \alpha\ \mathsf{list}$. Letting α be in scope within the body of the annotated term breaks down if we add intersection types (which aren't in this paper, but we want a general mechanism).

Figure 4 gives the rules for deriving $(\Gamma' \vdash A') \lesssim (\Gamma \vdash A)$, where $(\Gamma' \vdash A')$ is the user's typing from an annotation, Γ is the "ambient" context under which the annotated term $(e : N)$ is being typed, and A is A', renamed as needed to match Γ. The only output is A. Rule $\lesssim$-empty allows closed types with an empty context, e.g. $(e : (\cdot \vdash (\forall \beta.\, \beta \to \beta) \to \mathbf{1}))$; in practice, the "$\cdot \vdash$" can be omitted. Rule $\lesssim$-pvar is used when the typing mentions a program variable, as $x{:}\alpha$ in the example; the premise $\Gamma(x) \equiv B_0$ denotes equality, modulo renaming of type variables[1]. Rule $\lesssim$-tyvar allows α-varying (no pun intended) of type variables. Note that as the rules traverse the left-hand context from left to right, the left-hand context in the judgment can become ill-formed, but the output (right-hand) context is always well-formed.

Contextual annotations' major virtue is robustness: they work with or without intersection types, index refinements, and other features. The formalism can be simplified in practice—since we don't regard type variables in the ambient context as being in scope in Γ_0, and the notation (and implementation) syntactically distinguish type variables from other things, the type declarations α could be omitted. Or, as long as we don't have intersection types, we could declare that α is within the scope of its annotation, cutting out the nondeterministic choice of α' in $\lesssim$-tyvar.

Contextual annotations also set the stage for System Bi$^{\widehat{\alpha}}$, where we'll add *hint declarations* **hint** $(\Gamma_A \vdash A)$ **in** e. These are suggestions from the user to the typechecker: under a context Γ, when examining e, the typechecker can try A when instantiating a quantifier $\forall \beta.\, B$—with the context Γ_A establishing the map from type variables in A to type variables in Γ.

2.2 The metatheory of System Bi

Type safety can be proved in a three-step process:

1. Define a type assignment version of System Bi.
2. Show that every derivation in System Bi has a corresponding derivation in the type assignment system.
3. Prove a type safety theorem for the type assignment system, with respect to the operational semantics in Figure 1.

Step 1 is very easy: drop the rule anno and replace "$\Uparrow$" and "$\Downarrow$" symbols in the typing judgments with ":". For example, $\Gamma \vdash e_1\, e_2 \Uparrow B$ in rule $\to$E becomes $\Gamma \vdash e_1\, e_2 : B$.

For Step 2, we must show that given a derivation of $\Gamma \vdash e \Downarrow A$ (or of $\Gamma \vdash e \Uparrow A$) in System Bi, we can construct a derivation of $\Gamma \vdash e' : A$, where e' is e with annotations erased. This is an easy proof by induction on the derivation, and I proved it in my dissertation Dunfield (2007b, Ch. 2) for a similar (though richer) system. The only novelty here is parametric polymorphism, which presents no difficulties: the cases for $\forall$I and $\forall$E almost exactly follow the cases for ΠI and ΠE (the rules for universal index quantification, a type system feature omitted from this paper for simplicity).

Step 3 is not trivial, but it *is* an easy extension of the proof in my dissertation (Dunfield 2007b, Ch. 2). As in Step 2, the reasoning for $\forall$I and $\forall$E follows the reasoning for ΠI and ΠE. In particular, there is no need to extend *derivation rank* and *value definiteness* (Dunfield 2007b, pp. 36–38), concepts needed for union types—which are not even present in System Bi.

[1] This is more restricted than the rule in Dunfield and Pfenning (2004), which allowed $\Gamma(x)$ to be a *subtype* of B_0; this restriction will simplify System Bi$^{\widehat{\alpha}}$.

$$\frac{FV(A) \subseteq \mathsf{dom}(\Gamma)}{\Gamma \vdash A\ \mathsf{wf}} \qquad \frac{\widehat{\alpha} \notin \mathsf{dom}(\Gamma_1) \qquad \Gamma_1, \widehat{\alpha} \vdash \Gamma_2\ \mathsf{wf}}{\Gamma_1 \vdash \widehat{\alpha}, \Gamma_2\ \mathsf{wf}}$$

$$\frac{}{\Gamma \vdash \cdot\ \mathsf{wf}} \qquad \frac{\widehat{\alpha} \notin \mathsf{dom}(\Gamma_1) \qquad \Gamma_1 \vdash A\ \mathsf{wf} \qquad \Gamma_1, \widehat{\alpha}{=}A \vdash \Gamma_2\ \mathsf{wf}}{\Gamma_1 \vdash \widehat{\alpha}{=}A, \Gamma_2\ \mathsf{wf}}$$

Figure 5: Well-formedness of existential contexts and types

3. System Bi$^{\widehat{\alpha}}$: Explicit Existential Variables

Now let's transform the declarative System Bi into an algorithmic System Bi$^{\widehat{\alpha}}$ ("bi ex") by adding existential variables for unsolved polymorphic instances. After extending the syntax, we explain the typing and subtyping rules, discuss the **hint** construct, and then prove (with respect to System Bi) soundness and a limited form of completeness.

$$\begin{array}{lrcl} \text{Types} & A & ::= & \ldots \mid \widehat{\alpha} \\ \text{Contexts} & \Gamma, \Omega & ::= & \ldots \mid \Gamma, \widehat{\alpha} \mid \Gamma, \widehat{\alpha}{=}A \mid \Gamma, \blacktriangleleft_{\widehat{\alpha}} \mid \Gamma, \mathsf{hint}(\Gamma' \vdash A') \\ \text{Terms} & e & ::= & \ldots \mid \mathbf{hint}\ (\Gamma' \vdash A')\ \mathbf{in}\ e \end{array}$$

We write $\widehat{\alpha}$, $\widehat{\beta}$, and so on for existential type variables, created in situations corresponding to the $\forall$E and $\forall$L$\leqq$ rules of System Bi. We create $\widehat{\alpha}$ by adding $\widehat{\alpha}$ to the context Γ. When the system finds a solution (e.g. when trying to derive $\widehat{\alpha} \leqq \mathbf{1}$) the declaration $\widehat{\alpha}$ is replaced by $\widehat{\alpha}{=}\mathbf{1}$, indicating that the solution of $\widehat{\alpha}$ is $\mathbf{1}$. Contexts are ordered: the position of the declaration $\widehat{\alpha}$ determines which variables can appear in a solution: in the context $\Gamma_1, \widehat{\alpha}{=}A, \Gamma_2$ the solution type A must be well-formed under Γ_1, without using anything declared in Γ_2. This prevents circularity, and allows rules like $\forall$I that add non-existential declarations to remove them without making dangling references. Similarly, $\widehat{\alpha}, x{:}\widehat{\alpha}$ is well-formed because $\widehat{\alpha}$ is declared before $x{:}\widehat{\alpha}$.

Since the rules need to add and replace things in Γ, we modify judgment forms like $\Gamma \vdash e \Downarrow C$:

$$\begin{array}{lcl} \Gamma \vdash e \Downarrow C & \text{becomes} & \Gamma \vdash e \Downarrow C \dashv \Gamma' \\ \Gamma \vdash e \Uparrow C & \text{becomes} & \Gamma \vdash e \Uparrow C \dashv \Gamma' \\ \Gamma \vdash A \leqq B & \text{becomes} & \Gamma \vdash A \leqq B \dashv \Gamma' \end{array}$$

The *output context* Γ' is like Γ but may have more information, containing new $\widehat{\alpha}$ and $\widehat{\alpha}{=}A$ elements, and various $\widehat{\beta}$ elements replaced by $\widehat{\beta}{=}B$ elements. (I chose $\vdash$ and $\dashv$ to suggest the fact that Γ and Γ' are equivalent in a declarative sense: if all the $\widehat{\alpha}$, $\widehat{\alpha}{=}A$, $\blacktriangleleft_{\widehat{\alpha}}$, $\mathsf{hint}(\ldots)$ declarations are dropped from Γ and Γ', those contexts are equal.)

For the *marker* $\blacktriangleleft_{\widehat{\alpha}}$, we can thank the proof of predicative completeness: one typing rule ($\forall$L$\widehat{\alpha}\leqq$) needs this marker to remove junk—$\widehat{\alpha}$-variables that have gone out of scope—from the output context. Junk is harmless but would complicate the proof. Markers are ignored otherwise (and need not be implemented).

A context Γ is well-formed, $\cdot \vdash \Gamma$ wf, if each variable occurs once in its domain (defined below) and each type in Γ is well-formed under the declarations to its left.

Definition 1 (Domain of Γ)**.** The domain $\mathsf{dom}(\Gamma)$ of a context Γ is:

$$\begin{array}{lcl}
\mathsf{dom}(\cdot) & = & \emptyset \\
\mathsf{dom}(\Gamma, x{:}A) & = & \mathsf{dom}(\Gamma) \cup \{x\} \\
\mathsf{dom}(\Gamma, \alpha) & = & \mathsf{dom}(\Gamma) \cup \{\alpha\} \\
\mathsf{dom}(\Gamma, \widehat{\alpha}) & = & \mathsf{dom}(\Gamma) \cup \{\widehat{\alpha}\} \\
\mathsf{dom}(\Gamma, \widehat{\alpha}{=}A) & = & \mathsf{dom}(\Gamma) \cup \{\widehat{\alpha}\} \\
\mathsf{dom}(\Gamma, \mathsf{hint}(\Gamma' \vdash A')) & = & \mathsf{dom}(\Gamma) \\
\mathsf{dom}(\Gamma, \blacktriangleleft_{\widehat{\alpha}}) & = & \mathsf{dom}(\Gamma)
\end{array}$$

To prove properties of System Bi$^{\widehat{\alpha}}$, it's useful to view existential contexts as iterated substitutions, so that

$$[\widehat{\alpha}{=}A, \widehat{\beta}{=}\widehat{\alpha}](\widehat{\alpha} \to \widehat{\beta}) = A \to A$$

The context is applied from the right, so first $\widehat{\alpha}$ replaces $\widehat{\beta}$, giving $\widehat{\alpha} \to \widehat{\alpha}$, and then A replaces $\widehat{\alpha}$, resulting in $A \to A$.

We only apply contexts that complete the context in which the type lives, so all existential variables disappear: given $\widehat{\alpha} \to \widehat{\beta}$, well-formed in the context $(\widehat{\beta}{=}\widehat{\alpha}, \widehat{\alpha})$, applying $[\widehat{\beta}{=}\widehat{\alpha}, \widehat{\alpha}{=}\mathbf{1}]\ \widehat{\alpha} \to \widehat{\beta}$ yields $\mathbf{1} \to \mathbf{1}$. To apply a context Ω to another context Γ, the contexts must be the same except for Γ having more unsolved variables (and ignoring hints and markers), and Ω having solutions for variables not even mentioned in Γ:

$$\begin{array}{llcll}
 & [\cdot] \quad \cdot & = & \cdot & \\
 & [\Omega, x{:}A] \quad (\Gamma, x{:}A) & = & [\Omega]\Gamma, x{:}[\Omega]A & \\
 & [\Omega, \alpha] \quad (\Gamma, \alpha) & = & [\Omega]\Gamma, \alpha & \\
 & [\Omega, \widehat{\alpha}{=}A] \quad (\Gamma, \widehat{\alpha}) & = & [\Omega]\,([A/\widehat{\alpha}]\Gamma) & \\
* & [\Omega, \widehat{\alpha}{=}A_\Omega] \quad (\Gamma, \widehat{\alpha}{=}A_\Gamma) & = & [\Omega]\,([A_\Gamma/\widehat{\alpha}]\Gamma) & \text{if } [\Omega]A_\Omega = [\Omega]A_\Gamma \\
 & [\Omega, \widehat{\alpha}{=}A] \quad \Gamma & = & [\Omega]\Gamma & \text{if } \widehat{\alpha} \notin \mathsf{dom}(\Gamma) \\
 & [\Omega]\ (\Gamma, \mathsf{hint}(\Gamma' \vdash A')) & = & [\Omega]\Gamma & \\
 & [\Omega, \mathsf{hint}(\Gamma' \vdash A')]\ \Gamma & = & [\Omega]\Gamma & \\
 & [\Omega]\ (\Gamma, \blacktriangleleft_{\widehat{\alpha}}) & = & [\Omega]\Gamma &
\end{array}$$

The line marked * allows an Ω to complete a context with solutions written in a different but equivalent way: for example, $[\widehat{\alpha}{=}\mathbf{1}, \widehat{\beta}{=}\mathbf{1}](\widehat{\alpha}{=}\mathbf{1}, \widehat{\beta}{=}\widehat{\alpha})$ because $[\widehat{\alpha}{=}\mathbf{1}]\mathbf{1} = \mathbf{1} = [\widehat{\alpha}{=}\mathbf{1}]\widehat{\alpha}$.

Definition 2 (Solved contexts)**.** A context Γ' is *solved* if it contains no unsolved existentials $\widehat{\alpha}$.

Definition 3. We write $\Gamma \subseteq \Omega$ if (1) for all $\widehat{\alpha}$ in Γ there is a solution $\widehat{\alpha}{=}A$ in Ω, and for all $\widehat{\alpha}{=}A_\Gamma$ in Γ there is $\widehat{\alpha}{=}A_\Omega$ such that $[\Omega]A_\Omega = [\Omega]A_\Gamma$, and (2) declarations present in Γ appear in the same order in Ω.

Definition 4 (Completion of contexts)**.** A context Ω *completes* a context Γ iff $\Gamma \subseteq \Omega$ and Ω is solved.

How these existential contexts behave is best shown with an example. Suppose that Γ has $f{:}\mathsf{int}{\to}\mathsf{bool}$. At the top of Figure 6 is a derivation in System Bi, which "guesses" $\alpha = \mathsf{int}$.

At the bottom of the figure is a derivation in System Bi$^{\widehat{\alpha}}$. It has three interesting parts; the names of the involved rules are shaded, along with changes in the existential context. Towards the left we apply $\forall\mathrm{E}\widehat{\alpha}$, adding an unsolved existential $\widehat{\alpha}$ to the output context. Along the upper right is a use of $\widehat{\alpha}^{=}\mathrm{L}\leqq$, which expresses the essence of the greedy method: if we need to satisfy $\widehat{\alpha} \leqq B$, take B as the solution. In this example, B is int. The premise of $\widehat{\alpha}^{=}\mathrm{L}\leqq$ checks that the solution is well-formed in the context to the left of $\widehat{\alpha}$ in $\Gamma, \widehat{\alpha}$.

Existential contexts flow "in-order", starting in the conclusion on the left of the $\vdash$, up to the first premise (left of the $\vdash$), into the first premise's derivation, then back into the first premise itself (right of the $\dashv$), over to the second premise (left of the $\vdash$), etc., and finally back to the conclusion on the right of the $\dashv$.

Finally, while omitted from the figure, within the subderivation of $\Gamma, \widehat{\alpha}{=}\mathsf{int} \vdash xs \Downarrow \widehat{\alpha}\ \mathsf{list} \dashv \Gamma, \widehat{\alpha}{=}\mathsf{int}$ we apply a rule to replace $\widehat{\alpha}$ with int; this is not done implicitly.

3.1 Hints

We could have an explicit instantiation construct $e\,[A']$, such that if $e \Uparrow \forall\alpha.\ A$, then $e\,[A'] \Uparrow [A'/\alpha]A$. In effect, this gives an explicit version of $\forall$E. But we also have the subtyping rule $\forall\mathrm{L}\leqq$, which can be used on a deeply nested quantifier—and then where would we put the $[A']$? We might write a type annotation $(e : [A'/\alpha]A)$, but this is verbose when A is long.

So, instead of a construct that only works with $\forall$E, we add one that lets the user suggest an instance for $\forall$E or $\forall\mathrm{L}\leqq$. The syntax is

$$\textbf{hint}\ (\Gamma' \vdash A')\ \textbf{in}\ e$$

When encountered, the typing $(\Gamma' \vdash A')$ is put in Γ:

$$\frac{\Gamma, \mathsf{hint}(\Gamma' \vdash A') \vdash e \Downarrow C}{\Gamma \vdash \textbf{hint}\ (\Gamma' \vdash A')\ \textbf{in}\ e \Downarrow C}\ \mathsf{hint}$$

The type is then available to the rules $\forall$E-hint and $\forall$L-hint$\leqq$. As with contextual annotations, the context Γ' guides the interpretation of A'. For example, $\textbf{hint}\ (\alpha, x{:}\alpha \vdash \forall\beta.\ \alpha \to \beta)\ \textbf{in}\ e$ constrains α to be the type variable that is the type of x. On the other hand, $\textbf{hint}\ (\alpha \vdash \forall\beta.\ \alpha \to \beta)\ \textbf{in}\ e$ is unconstrained; α could be replaced by any available type variable. This is managed through the contextual subtyping rules in Figure 4. One new contextual subtyping rule is needed, to ignore hint declarations:

$$\frac{(\Gamma_0 \vdash A_0) \lesssim (\Gamma \vdash A)}{(\mathsf{hint}(\Gamma' \vdash A'), \Gamma_0 \vdash A_0) \lesssim (\Gamma \vdash A)}\ \lesssim\text{-hint}$$

To ensure decidability, rules hint and $\forall$L-hint$\leqq$ remove hints as they use them. With no restriction, writing $\textbf{hint}\ (\cdot \vdash \forall\beta.\ \beta)\ \textbf{in}\ f\ x$ where f has type $\forall\alpha.\ \alpha$ is fatal: using the hint, we replace α with $\forall\beta.\ \beta$, resulting in $\forall\beta.\ \beta$, on which we can use the hint again, and again…[2]

3.2 Typing and subtyping rules

Many of the typing and subtyping rules of System Bi$^{\widehat{\alpha}}$ (Figure 7) are the same as System Bi, overlaid with existential contexts. We'll look at typing first.

From the top, var, sub, anno, $\to$I, $\to$E and $\forall$I clearly correspond to the rules in Figure 3. Note that $\to$I and $\forall$I add declarations $x{:}A$ and α, respectively, and in their conclusions drop some existential declarations Γ_Z. Those declarations are out of scope, and since they appear on the right, nothing else refers to them. $\forall\mathrm{E}\widehat{\alpha}$ adds a fresh $\widehat{\alpha}$ to the existential context and synthesizes $[\widehat{\alpha}/\alpha]A$. The rules ExSubst$\Downarrow$ and ExSubst$\Uparrow$ apply the solution to $\widehat{\alpha}$ in the checking and synthesizing direction, respectively. ExSubst$\Downarrow$ does not apply $[A/\widehat{\alpha}]$ to Γ, because if we have, say, $y{:}\widehat{\alpha}$ in Γ, we can apply ExSubst$\Uparrow$ after applying var. The rule $\to\mathrm{I}\widehat{\alpha}$ is syntax-directed: if checking a λ against $\widehat{\alpha}$, then $\widehat{\alpha} = \widehat{\alpha_1} \to \widehat{\alpha_2}$ for some new "articulation" variables $\widehat{\alpha_1}, \widehat{\alpha_2}$. Rule $\to\mathrm{E}\widehat{\alpha}$ is dual.

In the subtyping rules, we change $\forall\mathrm{L}\leqq$ as we changed $\forall$E, to add an $\widehat{\alpha}$:

$$\frac{\Gamma, \blacktriangleleft_{\widehat{\alpha}}, \widehat{\alpha} \vdash [\widehat{\alpha}/\alpha]A \leqq B \dashv \Gamma', \blacktriangleleft_{\widehat{\alpha}}, \Gamma_Z}{\Gamma \vdash \forall\alpha.\ A \leqq B \dashv \Gamma'}\ \forall\mathrm{L}\widehat{\alpha}\leqq$$

As in $\to$I and $\forall$I, the declarations following the added $\widehat{\alpha}$ declaration are dropped. Because $\to\widehat{\alpha}\mathrm{L}\leqq$ and $\to\widehat{\alpha}\mathrm{R}\leqq$ (below) can insert

[2] My implementation imposes a looser restriction: an $\forall$ that came from a hint cannot be instantiated with another $\forall$ from a hint, but can be used more than once.

$$
\dfrac{
  \dfrac{\Gamma \vdash \mathit{filter} \Uparrow \forall\alpha.\,(\alpha\to\mathsf{bool})\to\alpha\ \mathsf{list}\to\alpha\ \mathsf{list} \qquad \Gamma \vdash \mathsf{int}\ \mathsf{wf}}{\Gamma \vdash \mathit{filter} \Uparrow (\mathsf{int}\to\mathsf{bool})\to\mathsf{int}\ \mathsf{list}\to\mathsf{int}\ \mathsf{list}}\;\forall\mathrm{E}
  \qquad
  \dfrac{\Gamma \vdash \mathsf{f} \Uparrow \mathsf{int}\to\mathsf{bool} \qquad \dfrac{\Gamma \vdash \mathsf{int} \leqq \mathsf{int} \qquad \Gamma \vdash \mathsf{bool} \leqq \mathsf{bool}}{\Gamma \vdash \mathsf{int}\to\mathsf{bool} \leqq \mathsf{int}\to\mathsf{bool}}\;{\to}{\leqq}}{\Gamma \vdash \mathsf{f} \Downarrow \mathsf{int}\to\mathsf{bool}}\;\mathrm{sub}
}{
  \dfrac{\Gamma \vdash \mathit{filter}\ \mathsf{f} \Uparrow \mathsf{int}\ \mathsf{list}\to\mathsf{int}\ \mathsf{list} \qquad \Gamma \vdash \mathsf{xs} \Downarrow \mathsf{int}\ \mathsf{list}}{\Gamma \vdash \mathit{filter}\ \mathsf{f}\ \mathsf{xs} \Uparrow \mathsf{int}\ \mathsf{list}}\;{\to}\mathrm{E}
}\;{\to}\mathrm{E}
$$

$$
\dfrac{
  \dfrac{\Gamma \vdash \mathit{filter} \Uparrow \forall\alpha.\,(\alpha\to\mathsf{bool})\to\alpha\ \mathsf{list}\to\alpha\ \mathsf{list} \dashv \Gamma}{\Gamma \vdash \mathit{filter} \Uparrow (\widehat{\alpha}\to\mathsf{bool})\to\widehat{\alpha}\ \mathsf{list}\to\widehat{\alpha}\ \mathsf{list} \dashv \Gamma, \widehat{\alpha}}\;\forall\mathrm{E}\widehat{\alpha}
  \qquad
  \dfrac{\Gamma,\widehat{\alpha} \vdash \mathsf{f} \Uparrow \mathsf{int}\to\mathsf{bool} \dashv \Gamma,\widehat{\alpha} \qquad \dfrac{\dfrac{\Gamma \vdash \mathsf{int}\ \mathsf{wf}}{\Gamma,\widehat{\alpha} \vdash \widehat{\alpha} \leqq \mathsf{int} \dashv \Gamma,\widehat{\alpha}{=}\mathsf{int}}\;\widehat{\alpha}^{=}\mathrm{L}{\leqq} \qquad \Gamma,\widehat{\alpha}{=}\mathsf{int} \vdash \mathsf{bool} \leqq \mathsf{bool} \dashv \Gamma,\widehat{\alpha}{=}\mathsf{int}}{\Gamma,\widehat{\alpha} \vdash \mathsf{int}\to\mathsf{bool} \leqq \widehat{\alpha}\to\mathsf{bool} \dashv \Gamma,\widehat{\alpha}{=}\mathsf{int}}\;{\to}{\leqq}}{\Gamma,\widehat{\alpha} \vdash \mathsf{f} \Downarrow \widehat{\alpha}\to\mathsf{bool} \dashv \Gamma,\widehat{\alpha}{=}\mathsf{int}}\;\mathrm{sub}
}{
  \dfrac{\Gamma \vdash \mathit{filter}\ \mathsf{f} \Uparrow \widehat{\alpha}\ \mathsf{list}\to\widehat{\alpha}\ \mathsf{list} \dashv \Gamma,\widehat{\alpha}{=}\mathsf{int} \qquad \Gamma,\widehat{\alpha}{=}\mathsf{int} \vdash \mathsf{xs} \Downarrow \widehat{\alpha}\ \mathsf{list} \dashv \Gamma,\widehat{\alpha}{=}\mathsf{int}}{\Gamma \vdash \mathit{filter}\ \mathsf{f}\ \mathsf{xs} \Uparrow \widehat{\alpha}\ \mathsf{list} \dashv \Gamma,\widehat{\alpha}{=}\mathsf{int}}\;{\to}\mathrm{E}
}\;{\to}\mathrm{E}
$$

Figure 6: Typing derivations for *filter* f xs in System Bi, above, and System Bi$^{\widehat{\alpha}}$, below

existential "articulation" variables just before $\widehat{\alpha}$, however, an explicit marker is needed to drop those declarations. The marker $\blacktriangleleft_{\widehat{\alpha}}$ separates the context it follows from $\widehat{\alpha}$'s articulation variables, $\widehat{\alpha}$, and anything else (Γ_Z) created after it. This bookkeeping prevents existential variables that won't subsequently be used from building up in the context, making the completeness proof easier to manage. (Implementing junk is harmless, and mine doesn't try to remove it.)

The subtyping rules ExSubst{L,R}$\leqq$ correspond to the typing rule ExSubst$\Uparrow$. When there is an arrow on one side and an existential variable on the other, $\to\widehat{\alpha}\mathrm{L}{\leqq}/\to\widehat{\alpha}\mathrm{R}{\leqq}$ split the existential (similar to $\to\mathrm{I}\widehat{\alpha}$). Eventually an "atomic" type is reached, and $\widehat{\alpha}^{=}\mathrm{L}{\leqq}/\widehat{\alpha}^{=}\mathrm{R}{\leqq}$ can be applied. These rules greedily instantiate the existential to the atomic type on the other side of $\leqq$. "Atomic" is a misnomer here: it could be a polytype $\forall\alpha.\ A$; the point is to keep it from being an arrow, which would complicate the proof of predicative completeness. The premises of $\widehat{\alpha}^{=}\mathrm{L}{\leqq}$ and $\widehat{\alpha}^{=}\mathrm{R}{\leqq}$ check that the solution is well-formed under the declarations that precede the variable.

3.3 Contextual subtyping rules

Because $\lesssim$-pvar uses equality, modulo renaming, instead of the full $\leqq$ relation, the contextual subtyping rules from System Bi do not change (apart from the $\lesssim$-hint rule).

3.4 Preliminaries

For the metatheory, we will use a function $\overline{\Gamma}$ that drops existential variable information and hints from Γ, yielding an "ordinary" Γ consisting only of variable declarations x:A and type variables α.

$$
\begin{array}{rcl}
\overline{\cdot} &=& \cdot \\
\overline{\Gamma, x{:}A} &=& \overline{\Gamma}, x{:}A \\
\overline{\Gamma, \alpha} &=& \overline{\Gamma}, \alpha
\end{array}
\qquad
\begin{array}{rcl}
\overline{\Gamma, \widehat{\alpha}} &=& \overline{\Gamma} \\
\overline{\Gamma, \widehat{\alpha}{=}A} &=& \overline{\Gamma}
\end{array}
\qquad
\begin{array}{rcl}
\overline{\Gamma, \blacktriangleleft_{\widehat{\alpha}}} &=& \overline{\Gamma} \\
\overline{\Gamma, \mathsf{hint}(\Gamma' \vdash A')} &=& \overline{\Gamma}
\end{array}
$$

The proof of Lemma 5 is by induction on the given derivation; Lemmas 6 and 7, by induction on Γ_2.

Lemma 5. *If* $\Gamma_1 \vdash \mathcal{J} \dashv \Gamma_2$ *then* $\overline{\Gamma_1} = \overline{\Gamma_2}$.

Lemma 6. *If* $\overline{\Gamma_1, \Gamma_Z} = \overline{\Gamma_2}$ *where* Γ_Z *has the form* x:A *or* α *then* $\Gamma_2 = \Gamma_{21}, \Gamma_Z, \Gamma_{22}$ *where* $\overline{\Gamma_{22}} = \cdot$.

Lemma 7. $(\Gamma_2 \vdash A) \lesssim (\Gamma_1, \Gamma_2 \vdash A)$.

Corollary 8 (Reflexivity)**.** $(\Gamma \vdash A) \lesssim (\Gamma \vdash A)$.

Lemma 9. *If* Ω *completes* Γ *then* $\mathsf{dom}([\Omega]\Gamma) \subseteq \mathsf{dom}(\Gamma)$.

Proof. By induction on Ω. Since Ω completes Γ, the contexts are the same modulo hints and existential variables that are declared in both but only solved in Ω, or declared in Ω only. In the case when $\Omega = \Omega', \widehat{\alpha}{=}A$ and $\Gamma = \Gamma', \widehat{\alpha}$: from the definition, $[\Omega]\Gamma = [\Omega']([A/\widehat{\alpha}]\Gamma')$. By IH, $\mathsf{dom}([\Omega']([A/\widehat{\alpha}]\Gamma') \subseteq \mathsf{dom}([A/\widehat{\alpha}]\Gamma')$, and substituting for $\widehat{\alpha}$ in Γ' does not change its domain at all. ☐

Lemma 10. *Given a context* Ω *that completes* Γ, *if* $\Gamma \vdash A\ \mathsf{wf}$ *then* $[\Omega]\Gamma \vdash [\Omega]A\ \mathsf{wf}$.

Proof. By inversion on $\Gamma \vdash A\ \mathsf{wf}$, $FV(A) \subseteq \mathsf{dom}(\Gamma)$. Since Ω completes Γ and is (implicitly) well-formed, all free variables of $[\Omega]A$ are α-variables, and $\mathsf{dom}(\Gamma)$ and $\mathsf{dom}([\Omega]\Gamma)$ have the same α-variables. So $FV([\Omega]A) \subseteq \mathsf{dom}([\Omega]\Gamma)$. ☐

Lemma 11 (Well-Formedness)**.** *If* $\mathcal{D} :: \Gamma \vdash \ldots \dashv \Gamma'$ *then for any solved* $\widehat{\alpha} \in \mathsf{dom}(\Gamma)$, *it is the case that* $\Gamma = \Gamma_1, \widehat{\alpha}{=}A, \Gamma_2$ *and* $\Gamma_1 \vdash A\ \mathsf{wf}$, *and likewise for any solved* $\widehat{\alpha} \in \mathsf{dom}(\Gamma')$.

Lemma 12 (Monotonicity)**.** *If* $\Gamma \vdash \ldots \dashv \Gamma'$ *then for any* $\widehat{\alpha} \in \mathsf{dom}(\Gamma')$, *one of the following holds:*

(1) $\widehat{\alpha}$ *is unsolved in both* Γ *and* Γ'; *or*
(2) there exists A' *such that* $\widehat{\alpha}$ *is unsolved in* Γ *and* $\Gamma' = \Gamma_1', \widehat{\alpha}{=}A', \Gamma_2'$; *or*
(3) there exists A' *such that* $\Gamma = \Gamma_1, \widehat{\alpha}{=}A', \Gamma_2$ *and* $\Gamma' = \Gamma_1', \widehat{\alpha}{=}A', \Gamma_2'$.

Also, markers are preserved:
if $\Gamma = \Gamma_1, \blacktriangleleft_{\widehat{\alpha}}, \Gamma_2$ *then* $\Gamma' = \Gamma_1', \blacktriangleleft_{\widehat{\alpha}}, \Gamma_2'$.

3.5 Decidability

System Bi$^{\widehat{\alpha}}$ is decidable. To concisely define an ordering on judgments such that the premises of each rule are smaller than its conclusion, we need several definitions:

(i) $A_1 \prec A_2$ iff A_1 is a proper subexpression of A_2, or if, by replacing one or more $\widehat{\alpha}$s with αs in A_1, we get a proper subexpression of A_2.

(ii) $\{C_1, C_2\} \prec \{D_1, D_2\}$ iff $C_k \not\succ D_\ell$ for all $k, \ell \in \{1, 2\}$, and there exist k, ℓ such that $C_k \prec D_\ell$.

$$\boxed{\Gamma \vdash e \Downarrow A \dashv \Gamma' \qquad \Gamma \vdash e \Uparrow A \dashv \Gamma'}$$

$$\frac{\Gamma(x) = A}{\Gamma \vdash x \Uparrow A \dashv \Gamma}\ \mathsf{var} \qquad \frac{\Gamma_1 \vdash e \Uparrow A \dashv \Gamma_2 \qquad \Gamma_2 \vdash A \leqq B \dashv \Gamma_3}{\Gamma_1 \vdash e \Downarrow B \dashv \Gamma_3}\ \mathsf{sub} \qquad \frac{N \lesssim (\Gamma \vdash A) \qquad \Gamma \vdash e \Downarrow A \dashv \Gamma'}{\Gamma \vdash (e : N) \Uparrow A \dashv \Gamma'}\ \mathsf{anno}$$

$$\frac{}{\Gamma \vdash () \Downarrow \mathbf{1} \dashv \Gamma}\ \mathbf{1}\mathrm{I} \qquad \frac{\Gamma, x{:}A \vdash e \Downarrow B \dashv \Gamma', x{:}A, \Gamma_Z}{\Gamma \vdash \lambda x.\, e \Downarrow A \to B \dashv \Gamma'}\ {\to}\mathrm{I} \qquad \frac{\Gamma_1 \vdash e_1 \Uparrow A \to B \dashv \Gamma_2 \qquad \Gamma_2 \vdash e_2 \Downarrow A \dashv \Gamma_3}{\Gamma_1 \vdash e_1\, e_2 \Uparrow B \dashv \Gamma_3}\ {\to}\mathrm{E}$$

$$\frac{\Gamma, \alpha \vdash v \Downarrow A \dashv \Gamma', \alpha, \Gamma_Z}{\Gamma \vdash v \Downarrow \forall\alpha.\, A \dashv \Gamma'}\ \forall\mathrm{I} \qquad \frac{\Gamma \vdash e \Uparrow \forall\alpha.\, A \dashv \Gamma'}{\Gamma \vdash e \Uparrow [\widehat{\alpha}/\alpha]\, A \dashv \Gamma', \widehat{\alpha}}\ \forall\mathrm{E}\widehat{\alpha}$$

$$\frac{\Gamma, \mathsf{hint}(\Gamma' \vdash A') \vdash e \Downarrow C \dashv \Gamma'}{\Gamma \vdash \textbf{hint}\,(\Gamma' \vdash A')\ \textbf{in}\ e \Downarrow C \dashv \Gamma'}\ \mathsf{hint} \qquad \frac{\Gamma_1 \vdash e \Uparrow \forall\alpha.\, A \dashv \Gamma_2 \qquad \begin{array}{l} \Gamma_2 = \Gamma_L, \mathsf{hint}(\Gamma_0 \vdash A_0), \Gamma_R \\ (\Gamma_0 \vdash A_0) \lesssim (\Gamma_L, \Gamma_R \vdash A') \end{array}}{\Gamma_1 \vdash e \Uparrow [A'/\alpha]\, A \dashv \Gamma_L, \Gamma_R}\ \forall\mathrm{E}\text{-hint}$$

$$\frac{\Gamma \vdash e \Downarrow \Gamma(\widehat{\alpha}) \dashv \Gamma'}{\Gamma \vdash e \Downarrow \widehat{\alpha} \dashv \Gamma'}\ \mathrm{ExSubst}{\Downarrow} \qquad \frac{\Gamma \vdash e \Uparrow \widehat{\alpha} \dashv \Gamma'}{\Gamma \vdash e \Uparrow \Gamma'(\widehat{\alpha}) \dashv \Gamma'}\ \mathrm{ExSubst}{\Uparrow}$$

$$\frac{\Gamma_1, \widehat{\alpha_1}, \widehat{\alpha_2}, \widehat{\alpha}{=}\widehat{\alpha_1}{\to}\widehat{\alpha_2}, \Gamma_2 \vdash \lambda x.\, e \Downarrow \widehat{\alpha} \dashv \Gamma'}{\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash \lambda x.\, e \Downarrow \widehat{\alpha} \dashv \Gamma'}\ {\to}\mathrm{I}\widehat{\alpha} \qquad \frac{\Gamma \vdash e_1 \Uparrow \widehat{\alpha} \dashv \Gamma_1', \widehat{\alpha}, \Gamma_2' \qquad \Gamma_1', \widehat{\alpha_1}, \widehat{\alpha_2}, \widehat{\alpha}{=}\widehat{\alpha_1}{\to}\widehat{\alpha_2}, \Gamma_2' \vdash e_2 \Downarrow \widehat{\alpha_1} \dashv \Gamma'}{\Gamma \vdash e_1\, e_2 \Uparrow \widehat{\alpha_2} \dashv \Gamma'}\ {\to}\mathrm{E}\widehat{\alpha}$$

$$\boxed{\Gamma \vdash A \leqq B \dashv \Gamma'}$$

$$\frac{}{\Gamma \vdash \mathbf{1} \leqq \mathbf{1} \dashv \Gamma}\ \mathbf{1}{\leqq} \qquad \frac{\Gamma_1 \vdash B_1 \leqq A_1 \dashv \Gamma_2 \qquad \Gamma_2 \vdash A_2 \leqq B_2 \dashv \Gamma_3}{\Gamma_1 \vdash A_1 \to A_2 \leqq B_1 \to B_2 \dashv \Gamma_3}\ {\to}{\leqq} \qquad \frac{}{\Gamma \vdash \alpha \leqq \alpha \dashv \Gamma}\ \alpha\mathrm{Refl}{\leqq}$$

$$\frac{\Gamma, \blacktriangleleft_{\widehat{\alpha}}, \widehat{\alpha} \vdash [\widehat{\alpha}/\alpha]A \leqq B \dashv \Gamma', \blacktriangleleft_{\widehat{\alpha}}, \Gamma_Z}{\Gamma \vdash \forall\alpha.\, A \leqq B \dashv \Gamma'}\ \forall\mathrm{L}\widehat{\alpha}{\leqq} \qquad \frac{\Gamma, \beta \vdash A \leqq B \dashv \Gamma', \beta, \Gamma_Z}{\Gamma \vdash A \leqq \forall\beta.\, B \dashv \Gamma'}\ \forall\mathrm{R}{\leqq}$$

$$\frac{\begin{array}{l} \Gamma_1 = \Gamma_L, \mathsf{hint}(\Gamma_0 \vdash A_0), \Gamma_R \\ (\Gamma_0 \vdash A_0) \lesssim (\Gamma_L, \Gamma_R \vdash A') \qquad \Gamma_L, \Gamma_R \vdash [A'/\alpha]A \leqq B \dashv \Gamma_2 \end{array}}{\Gamma_1 \vdash \forall\alpha.\, A \leqq B \dashv \Gamma_2}\ \forall\mathrm{L}\text{-hint}{\leqq}$$

$$\frac{}{\Gamma \vdash \widehat{\alpha} \leqq \widehat{\alpha} \dashv \Gamma}\ \widehat{\alpha}\mathrm{Refl}{\leqq} \qquad \frac{\Gamma \vdash \Gamma(\widehat{\alpha}) \leqq B \dashv \Gamma'}{\Gamma \vdash \widehat{\alpha} \leqq B \dashv \Gamma'}\ \mathrm{ExSubstL}{\leqq} \qquad \frac{\Gamma \vdash A \leqq \Gamma(\widehat{\beta}) \dashv \Gamma'}{\Gamma \vdash A \leqq \widehat{\beta} \dashv \Gamma'}\ \mathrm{ExSubstR}{\leqq}$$

$$\frac{\Gamma_1, \widehat{\alpha_1}, \widehat{\alpha_2}, \widehat{\alpha}{=}\widehat{\alpha_1}{\to}\widehat{\alpha_2}, \Gamma_2 \vdash \widehat{\alpha} \leqq B_1 \to B_2 \dashv \Gamma'}{\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash \widehat{\alpha} \leqq B_1 \to B_2 \dashv \Gamma'}\ {\to}\widehat{\alpha}\mathrm{L}{\leqq} \qquad \frac{\Gamma_1, \widehat{\beta_1}, \widehat{\beta_2}, \widehat{\beta}{=}\widehat{\beta_1}{\to}\widehat{\beta_2}, \Gamma_2 \vdash A_1 \to A_2 \leqq \widehat{\beta} \dashv \Gamma'}{\Gamma_1, \widehat{\beta}, \Gamma_2 \vdash A_1 \to A_2 \leqq \widehat{\beta} \dashv \Gamma'}\ {\to}\widehat{\alpha}\mathrm{R}{\leqq}$$

$$\frac{\Gamma_1 \vdash B^{\mathit{atomic}}\ \mathsf{wf}}{\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash \widehat{\alpha} \leqq B^{\mathit{atomic}} \dashv \Gamma_1, \widehat{\alpha}{=}B^{\mathit{atomic}}, \Gamma_2}\ \widehat{\alpha}^{=}\mathrm{L}{\leqq} \qquad \frac{\Gamma_1 \vdash A^{\mathit{atomic}}\ \mathsf{wf}}{\Gamma_1, \widehat{\beta}, \Gamma_2 \vdash A^{\mathit{atomic}} \leqq \widehat{\beta} \dashv \Gamma_1, \widehat{\beta}{=}A^{\mathit{atomic}}, \Gamma_2}\ \widehat{\alpha}^{=}\mathrm{R}{\leqq}$$

Figure 7: Typing and subtyping rules of System Bi$^{\widehat{\alpha}}$

(iii) The *weight* of an existential variable $\widehat{\alpha}$ in Γ is the number of existential variables in Γ_L where $\Gamma = \Gamma_L, \widehat{\alpha}[\ldots], \cdots$, plus itself. For example, the weight of $\widehat{\beta}$ in $\widehat{\alpha}, \widehat{\beta}{=}\mathbf{1}$ is 2. Solved and unsolved variables are counted alike. Weights are natural numbers, ordered by $<$.

(iv) A type's *angst* with respect to Γ is the weight of the type's heaviest existential variable, again ordered by $<$.

The last two criteria are motivated by ExSubst⇓, ExSubst⇑, and ExSubst{L,R}≦. For example, the type in ExSubst⇓'s premise is $\Gamma(\widehat{\alpha})$ while its conclusion has $\widehat{\alpha}$. In the sense of part (i), $\Gamma(\widehat{\alpha})$ could be much larger than $\widehat{\alpha}$. Counting the number of free existentials in the type doesn't work, because $\widehat{\alpha}$'s solution could be $\widehat{\alpha_1} \to \widehat{\alpha_2}$, which has two existential variables. But $\widehat{\alpha_1} \to \widehat{\alpha_2}$ *does* have less angst than $\widehat{\alpha}$, because $\widehat{\alpha_1}$ and $\widehat{\alpha_2}$ must be declared before $\widehat{\alpha}$ in Γ—otherwise they could not appear in $\widehat{\alpha}$'s solution.

In each rule, a term gets smaller, a type gets smaller (in the ordinary sense, e.g. A smaller than $A \to B$, or in the sense of becoming less angstful), the set of available hints gets smaller, or we introduce a solution for an existential variable. When comparing two synthesis judgments we flip the ordering of types because the types are output rather than input. The appendix has the definitions of the orderings on subtyping and typing judgments, and proofs of decidability.

3.6 Soundness of System Bi$^{\widehat{\alpha}}$

Each System Bi$^{\widehat{\alpha}}$ derivation corresponds to a Bi one. In combination with type safety for a type assignment version of System Bi, this means that a well-typed-in-System Bi$^{\widehat{\alpha}}$ program won't go wrong:

Theorem 13 (Soundness of System Bi$^{\widehat{\alpha}}$)**.** *If* $\Gamma \vdash \mathcal{J} \dashv \Gamma'$ *and* Ω *completes* Γ' *then* $[\Omega]\Gamma' \vdash [\Omega]\mathcal{J}'$*, where* $\mathcal{J}'$ *is* $\mathcal{J}$ *with any* **hint** $\ldots$ **in** e *subterms replaced by* e *and hints in annotations removed.*

Note that Ω is an input to the theorem. Consider the System Bi$^{\widehat{\alpha}}$ derivation of $\cdot \vdash (\lambda x.\, x : (\forall\alpha.\, (\alpha{\to}\alpha) \to \alpha \to \alpha)\lambda y.\, y \dashv \widehat{\alpha}$. To create the corresponding System Bi derivation, we must create an Ω that instantiates $\widehat{\alpha}$. Fortunately, we can instantiate it to anything, including $\mathbf{1}$.

3.7 Completeness of System Bi$^{\widehat{\alpha}}$

We will show that, with respect to System Bi, System Bi$^{\widehat{\alpha}}$ is incomplete for impredicative polymorphism, complete when **hint**s are added to the term, and complete for predicative polymorphism.

3.7.1 Impredicative incompleteness

A small example shows that System Bi$^{\widehat{\alpha}}$ is incomplete for impredicative polymorphism. We abbreviate $\forall\beta.\ \beta \to \beta$ as **ID**. Let $\Gamma = f{:}\forall\alpha.\ \alpha \to \alpha \to \mathbf{1}, x{:}(\mathsf{int}{\to}\mathsf{int}) \to \mathbf{1}, y{:}\mathbf{ID} \to \mathbf{1}$. The derivation in System Bi, shown at the top of Figure 8, has no hint-free analogue in System Bi$^{\widehat{\alpha}}$. Below it, the failed derivation in System Bi$^{\widehat{\alpha}}$ makes the problem clear: the first constraint on $\widehat{\alpha}$ is that it be a supertype of x's type, $(\mathsf{int} \to \mathsf{int}) \to \mathbf{1}$, so that type is used, greedily, as the solution of $\widehat{\alpha}$. (For clarity, we substitute for $\widehat{\alpha}$ in the rest of the derivation.) But the second constraint (shaded) requires that $\mathsf{int} \to \mathsf{int}$ be a subtype of **ID**, which is false. All the choices of rules are fully determined, so no derivation exists.

Note that System Bi$^{\widehat{\alpha}}$ *can* synthesize a type if we swap x and y. Rewriting code to match the vagaries of an algorithm is unpleasant, even when possible; the only way to guarantee that System Bi$^{\widehat{\alpha}}$ finds some impredicative instance is to declare a hint.

3.7.2 Hinted completeness

A weak completeness result says that for every System Bi derivation typing e, there exists a System Bi$^{\widehat{\alpha}}$ derivation typing e^+, where e^+ is e enclosed in **hint** declarations. The proof is in the appendix.

Theorem 14. *If $\Gamma \vdash \mathcal{J}$ in System Bi and Γ_H consists of hints, then $\Gamma_H, \Gamma'_H, \Gamma \vdash \mathcal{J} \dashv \Gamma_H, \Gamma$ in System Bi$^{\widehat{\alpha}}$ where Γ'_H consists of hints.*

Corollary 15 (Hinted Completeness)**.** *If $\cdot \vdash e \Downarrow A$ in System Bi then $\cdot \vdash e^+ \Downarrow A \dashv \cdot$ in System Bi$^{\widehat{\alpha}}$, where $e^+ =$ **hint** $(\Gamma_1 \vdash A_1)$ **in** … **hint** $(\Gamma_n \vdash A_n)$ **in** e.*

3.7.3 Predicative completeness

In this section, we show that System Bi$^{\widehat{\alpha}}$ is *predicatively complete*: given a derivation in System Bi in which all polymorphic instances A' used in $\forall$E and $\forall$L$\leqq$ are monotypes (contain no $\forall$), we can derive the same judgment in System Bi$^{\widehat{\alpha}}$. Consequently, System Bi$^{\widehat{\alpha}}$ is complete for prenex or ML-style polymorphism, in which instances are monotypes *and* $\forall$s appear only on the outside of types.

We show completeness by building a System Bi$^{\widehat{\alpha}}$ derivation from any System Bi one. Where we have a derivation in System Bi of $\Gamma \vdash \mathcal{J}'$, we create a derivation of $\Gamma'_1 \vdash \mathcal{J} \dashv \Gamma'_2$, where $\mathcal{J}$ is like $\mathcal{J}'$ but may have more existential variables. Specifically, $\mathcal{J}' = [\Omega]\mathcal{J}$ for some Ω representing solutions embedded in the System Bi derivation.

Moreover, Γ'_1 must correspond to Ω, and Γ'_2 to Ω, Ω' for some (often empty) Ω'. The example derivations from Figure 6 give some intuition for this correspondence.

When every arrow appearing in Ω has the form $\widehat{\alpha_1} \to \widehat{\alpha_2}$, we say that Ω is *articulated*. System Bi$^{\widehat{\alpha}}$ keeps contexts articulated by restricting $\widehat{\alpha}^{=}$L$\leqq$ and $\widehat{\alpha}^{=}$R$\leqq$, which instantiate existential variables, to non-arrows.

Let's define the articulation of $\widehat{\alpha}{=}A'$ as follows:

$$\begin{aligned}
\mathit{Artic}(\widehat{\alpha}{=}\mathbf{1}) &= \widehat{\alpha}{=}\mathbf{1}\\
\mathit{Artic}(\widehat{\alpha}{=}\beta) &= \widehat{\alpha}{=}\beta\\
\mathit{Artic}(\widehat{\alpha}{=}B_1{\to}B_2) &= \widehat{\alpha}{=}\widehat{\beta_1}{\to}\widehat{\beta_2}, \mathit{Artic}(\widehat{\beta_1}{=}B_1), \mathit{Artic}(\widehat{\beta_2}{=}B_2)
\end{aligned}$$

Since the given System Bi$^{\widehat{\alpha}}$ derivation is predicative, there is no need to define the articulation of $\forall\alpha.\ A$. The proof of the next theorem is in the appendix. Some of the proof cases use a lemma saying that if A is at least as polymorphic as B (in System Bi), but A and B are actually monotypes, then $A = B$.

Lemma 16. *If $\Gamma \vdash A \leqq B$ where A and B contain no $\forall$, then $A = B$.*

Proof. By induction on the given System Bi derivation. ☐

Theorem 17 (Predicative Completeness)**.** *For any Ω and Γ'_1 and predicative derivation $\mathcal{D}$ of $\Gamma \vdash [\Omega]\mathcal{J}$ in System Bi, provided that*

(1) Ω is predicative (for any $\widehat{\alpha}$, the type $\Omega(\widehat{\alpha})$ is monomorphic) and articulated, and
(2) Ω completes Γ'_1, and $[\Omega]\Gamma'_1 = \Gamma$, then

$$\begin{array}{lcl}
[\Omega]\Gamma'_1 \vdash [\Omega]A' \leq [\Omega]B' & & \\
\textit{where } \Gamma'_1 \vdash A' \textsf{ wf} \textit{ and } \Gamma'_1 \vdash B' \textsf{ wf} & \Longrightarrow & \Gamma'_1 \vdash A' \leqq B' \dashv \Gamma'_2\\
[\Omega]\Gamma'_1 \vdash e \Downarrow [\Omega]A' & & \\
\textit{where } \Gamma'_1 \vdash A' \textsf{ wf} & \Longrightarrow & \Gamma'_1 \vdash e \Downarrow A' \dashv \Gamma'_2\\
[\Omega]\Gamma'_1 \vdash e \Uparrow C & \Longrightarrow & \Gamma'_1 \vdash e \Uparrow C' \dashv \Gamma'_2\\
 & & \textit{for some } C' \textit{ such that}\\
 & & C = [\Omega, \Omega']C'
\end{array}$$

for some Ω' such that (Ω, Ω') completes Γ'_2 and $[\Omega, \Omega']\Gamma'_2 = \Gamma$.

Proof. See the appendix. ☐

4. Datatypes

Supporting datatypes in System Bi is straightforward: we need only add two typing rules and one subtyping rule. The existential-context versions of those rules, in System Bi$^{\widehat{\alpha}}$, follow the pattern of $\to$. We articulate as with $\to$; the analogy is clear if we think of $\to$ as just a two-argument datatype "$(\alpha, \beta) \to$". So we have $\delta\widehat{\alpha}$L$\leqq$ and $\delta\widehat{\alpha}$R$\leqq$ following $\to\widehat{\alpha}$L$\leqq$ and $\to\widehat{\alpha}$R$\leqq$, and a rule δI$\widehat{\alpha}$ following $\to$I$\widehat{\alpha}$. But I have no δE$\widehat{\alpha}$, which would solve $\widehat{\alpha}$ appropriately when checking **case** e **of** ms where $e \Uparrow \widehat{\alpha}$; such a rule could work in some cases, such as **case** e **of** $c(x) \Rightarrow \ldots$, because δ can be inferred from the pattern $c(x)$, but not in general. See Figure 9.

I assume covariant type arguments, but handling contravariant and nonvariant type arguments is easy: the rule $\delta\alpha$ just needs to flip subtyping judgments (for contravariance), or add flipped judgments (for nonvariance). Actually determining the variance of type arguments is outside this paper's scope.

5. Subtyping with Intersection and Union Types

By adding intersection and union types, we can extend System Bi and, more importantly, System Bi$^{\widehat{\alpha}}$ to subtyping, replacing the weak "at least as polymorphic as" relation $\leqq$ with a richer $\leq$.

A value has intersection type $A \wedge B$ if it has both type A and type B. Intersection types can express combinations of properties of functions and data constructors (Reynolds 1996; Davies 2005; Dunfield and Pfenning 2004). Union types (Pierce 1991; Dunfield and Pfenning 2004; Dunfield 2007b) are dual to intersection types; a value has type $A \vee B$ if it has type A or type B (or possibly both). Given an atomic subtyping relation on datatypes $\delta_1 \preceq \delta_2$,

$$
\dfrac{
  \dfrac{f \Uparrow \forall\alpha.\,\alpha \to \alpha \to \mathbf{1}}{f \Uparrow [(\mathbf{ID}{\to}\mathbf{1})/\alpha](\alpha{\to}\alpha{\to}\mathbf{1})}\;\forall\mathrm{E}
  \qquad
  \dfrac{x \Uparrow \ldots \qquad \dfrac{\mathbf{ID} \leqq \mathsf{int}{\to}\mathsf{int} \qquad \mathbf{1} \leqq \mathbf{1}}{(\mathsf{int}{\to}\mathsf{int}) \to \mathbf{1} \leqq \mathbf{ID}{\to}\mathbf{1}}\;{\to}{\leqq}}{x \Downarrow \mathbf{ID}{\to}\mathbf{1}}\;\mathsf{sub}
}{
  \dfrac{f\,x \Uparrow (\mathbf{ID}{\to}\mathbf{1}) \to \mathbf{1}}{}\;{\to}\mathrm{E}
}
$$

$$
\dfrac{f\,x \Uparrow (\mathbf{ID}{\to}\mathbf{1}) \to \mathbf{1} \qquad \dfrac{y \Uparrow \mathbf{ID}{\to}\mathbf{1} \qquad \mathbf{ID}{\to}\mathbf{1} \leqq \mathbf{ID}{\to}\mathbf{1}}{y \Downarrow \mathbf{ID}{\to}\mathbf{1}}\;\mathsf{sub}}{f\,x\,y \Uparrow \mathbf{1}}\;{\to}\mathrm{E}
$$

$$
\dfrac{
  \dfrac{
    \dfrac{f \Uparrow \forall\alpha.\,\alpha{\to}\alpha{\to}\mathbf{1}}{f \Uparrow \widehat{\alpha}{\to}\widehat{\alpha}{\to}\mathbf{1}}\;\forall\mathrm{E}\widehat{\alpha}
    \qquad
    \dfrac{x \Uparrow \ldots \qquad (\mathsf{int}{\to}\mathsf{int}) \to \mathbf{1} \leqq \widehat{\alpha}}{x \Downarrow \widehat{\alpha}}\;\mathsf{sub}
  }{f\,x \Uparrow ((\mathsf{int}{\to}\mathsf{int}){\to}\mathbf{1}) \to \mathbf{1}}\;{\to}\mathrm{E} + \mathrm{ExSubst}{\Uparrow}
  \qquad
  \dfrac{y \Uparrow \mathbf{ID}{\to}\mathbf{1} \qquad \dfrac{\mathsf{int}{\to}\mathsf{int} \nleqq \mathbf{ID} \qquad \mathbf{1} \leqq \mathbf{1}}{\mathbf{ID}{\to}\mathbf{1} \nleqq ((\mathsf{int}{\to}\mathsf{int}){\to}\mathbf{1})}\;{\to}{\leqq}}{y \not\Downarrow ((\mathsf{int}{\to}\mathsf{int}){\to}\mathbf{1})}\;\mathsf{sub}
}{f\,x\,y \not\Uparrow}\;{\to}\mathrm{E}
$$

Figure 8: Derivation in System Bi using impredicative polymorphism (top), and a failed derivation in System Bi$^{\widehat{\alpha}}$ (bottom)

Type variable sequences $\vec{\alpha}, \vec{\beta} \;::=\; \cdot \mid \alpha \mid (\alpha_1, \ldots, \alpha_n)$ Types $A, B, C \;::=\; A \to B \mid \alpha \mid \forall\alpha.\,A \mid \vec{B}\,\delta$

Add to the System Bi rules in Figure 3:

$$
\dfrac{\overline{\Gamma} \vdash c : A \to \vec{B}\,\delta \qquad \Gamma \vdash e \Downarrow A}{\Gamma \vdash c(e) \Downarrow \vec{B}\,\delta}\;\delta\mathrm{I}
\qquad
\dfrac{\Gamma \vdash e \Uparrow \vec{B}\,\delta \qquad \Gamma \vdash ms \Downarrow_{\vec{B}\,\delta} C}{\Gamma \vdash \textbf{case}\; e \;\textbf{of}\; ms \Downarrow C}\;\delta\mathrm{E}
\qquad
\dfrac{\Gamma \vdash A_1 \leq B_1 \quad \cdots \quad \Gamma \vdash A_n \leq B_n \qquad \delta_1 \preceq \delta_2}{\Gamma \vdash (A_1, \ldots, A_n)\,\delta_1 \leq (B_1, \ldots, B_n)\,\delta_2}\;\delta\alpha
$$

Add to the System Bi$^{\widehat{\alpha}}$ rules in Figure 7:

$$
\dfrac{\Gamma_1 \vdash c : A \to \vec{B}\,\delta \dashv \Gamma_2 \qquad \Gamma_2 \vdash e \Downarrow A \dashv \Gamma_3}{\Gamma_1 \vdash c(e) \Downarrow \vec{B}\,\delta \dashv \Gamma_3}\;\delta\mathrm{I}
\qquad
\dfrac{\Gamma_1 \vdash e \Uparrow \vec{B}\,\delta \dashv \Gamma_2 \qquad \Gamma_2 \vdash ms \Downarrow_{\vec{B}\,\delta} C \dashv \Gamma_3}{\Gamma_1 \vdash \textbf{case}\; e \;\textbf{of}\; ms \Downarrow C \dashv \Gamma_3}\;\delta\mathrm{E}
$$

$$
\dfrac{c \text{ constructs } \delta \qquad \begin{array}{l}\Gamma_1, \widehat{\alpha_1}, \ldots, \widehat{\alpha_2},\\ \quad \widehat{\alpha}{=}(\widehat{\alpha_1}, \ldots, \widehat{\alpha_n})\,\delta, \Gamma_2 \vdash c(e) \Downarrow \widehat{\alpha} \dashv \Gamma'\end{array}}{\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash c(e) \Downarrow \widehat{\alpha} \dashv \Gamma'}\;\delta\mathrm{I}\widehat{\alpha}
\qquad
\dfrac{\Gamma_1 \vdash A_1 \leq B_1 \dashv \Gamma_2 \quad \cdots \quad \Gamma_n \vdash A_n \leq B_n \dashv \Gamma_{n+1}}{\Gamma_1 \vdash (A_1, \ldots, A_n)\,\delta \leq (B_1, \ldots, B_n)\,\delta \dashv \Gamma_{n+1}}\;\delta\alpha
$$

$$
\dfrac{\begin{array}{l}\Gamma_1, \widehat{\alpha_1}, \ldots, \widehat{\alpha_n},\\ \quad \widehat{\alpha}{=}(\widehat{\alpha_1}, \ldots, \widehat{\alpha_n})\,\delta, \Gamma_2 \vdash \widehat{\alpha} \leqq (B_1, \ldots, B_n)\,\delta \dashv \Gamma'\end{array}}{\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash \widehat{\alpha} \leqq (B_1, \ldots, B_n)\,\delta \dashv \Gamma'}\;\delta\widehat{\alpha}\mathrm{L}{\leqq}
\qquad
\dfrac{\begin{array}{l}\Gamma_1, \widehat{\beta_1}, \ldots, \widehat{\beta_n},\\ \quad \widehat{\beta}{=}(\widehat{\beta_1}, \ldots, \widehat{\beta_n})\,\delta, \Gamma_2 \vdash (A_1, \ldots, A_n)\,\delta \leqq \widehat{\beta} \dashv \Gamma'\end{array}}{\Gamma_1, \widehat{\beta}, \Gamma_2 \vdash (A_1, \ldots, A_n)\,\delta \leqq \widehat{\beta} \dashv \Gamma'}\;\delta\widehat{\alpha}\mathrm{R}{\leqq}
$$

Figure 9: Extending System Bi and System Bi$^{\widehat{\alpha}}$ with datatypes

simply adding to System Bi$^{\widehat{\alpha}}$ typing and subtyping rules for intersections and unions (as we might in a setting without parametric polymorphism) delivers a reasonable system.

A further enhancement uses intersection and union types to refine the greedy approach itself. The idea is that, when the system tries to derive $\widehat{\alpha} \leq \mathsf{nat}$ (with $\widehat{\alpha}$ as yet unknown), don't instantiate $\widehat{\alpha}$ to nat permanently as above; instead, instantiate it provisionally. So, if we then see $\widehat{\alpha} \leq \mathsf{int}$, we *add* int, yielding $\widehat{\alpha}{=}\mathsf{nat} \wedge \mathsf{int}$: the type $\mathsf{nat} \wedge \mathsf{int}$, being the intersection of nat and int, is included in both, so $\mathsf{nat} \wedge \mathsf{int} \leq \mathsf{nat}$ and $\mathsf{nat} \wedge \mathsf{int} \leq \mathsf{int}$. Dually, if we need to derive $\mathsf{nat} \leq \widehat{\beta}$ and then $\mathsf{int} \leq \widehat{\beta}$, we end up with $\widehat{\beta}{=}\mathsf{nat} \vee \mathsf{int}$, because $\mathsf{nat} \vee \mathsf{int}$ is a *super*type of both nat and int.

For a fuller account of these systems, see Dunfield (2009) (which I expect to revise). I mention them here to suggest that my approach is robust. It does not break when a supposedly tricky feature, subtyping, is added, and there is an actual synergy between the greedy method and intersection/union types.

6. Implementation

I implemented a version of System Bi$^{\widehat{\alpha}}$ (with the datatype rules in Figure 9) as an extension of Stardust (Dunfield 2007a), a type-checker for a subset of Standard ML with intersection types, union types, datasort refinements, and index refinements. The implemented system is much richer than System Bi$^{\widehat{\alpha}}$, but the examples here don't use the extra features; unlike the extension mentioned in Section 5, the implementation does *not* automatically create intersections and unions.

The example in Figure 10 begins with a simple application of higher-rank predicative polymorphism, used in short-cut deforestation (Gill et al. 1993). Types are quantified explicitly in the function type annotations `(*[ ... ]*)`. `foldr` uses only prenex polymorphism and can of course be written in SML, but `build` uses rank-2 polymorphism. The rest is adapted from Leijen (2009), showing impredicative polymorphism.

6.1 Complexity of typechecking

If hints are used, typechecking a function can be exponential in the number of hints: at each opportunity to apply $\forall\mathrm{E}\widehat{\alpha}$ or $\forall$E-hint, there is a choice between applying $\forall\mathrm{E}\widehat{\alpha}$, applying $\forall$E-hint with the first available hint, with the second, etc. However, we can show the complexity is exponential even if $\forall\mathrm{E}\widehat{\alpha}$ is never used: As formulated, $\forall$E-hint drops a hint after use. First there are n hints and n choices; at the next opportunity to apply $\forall$E-hint there are $n-1$ hints and $n-1$ choices; and so on. If the last sequence of hints chosen is the only one to yield a valid derivation, we have done work proportional to $n \cdot (n-1) \cdot \ldots \cdot 2$, or roughly n^n.

```
datatype 'a list = Nil | Cons of 'a * 'a list ;

(*[ val foldr : -all 'a,'b- ('a*'b → 'b)
                       → 'b → 'a list → 'b   ]*)
fun foldr f u xs = case xs of
    Nil ⇒ u  |  Cons(x, xs) ⇒ f (x, foldr f u xs)

(*[ val build : -all 'a- (-all 'b- ('a*'b→'b)→'b→'b)
                      → 'a list                    ]*)
fun build f = f Cons Nil

(*[ val map : -all 'a,'b- ('a→'b)→'a list→'b list ]*)
fun map f xs = build (fn c ⇒ fn n ⇒ foldr
                        (fn (x,ys) ⇒ c (f x, ys)) n xs)


(*[ val id : -all 'a- 'a → 'a  ]*)   fun id x = x
(*[ val inc : int → int ]*)         fun inc x = x + 1

(*[ val poly : (-all 'a- 'a→'a) → int * bool  ]*)
fun poly f = (f 1, f true)

(*[ val single : -all 'a- 'a → 'a list ]*)
fun single x = Cons(x, Nil)

(*[ val append : -all 'a- 'a list→'a list→'a list ]*)
fun append xs ys = ...

val  _ = poly id
val  _ = poly (fn x ⇒ x)
val  ids = single id           (* (-all 'a- 'a→'a) list *)
val  _ = map poly ids             (* (int * bool) list *)
val  _ = append (single inc) ids  (* (int → int) list *)
```

Figure 10: Example of first-class polymorphism

I have not analyzed the complexity of typechecking, but consider the function **fun** *nonlinear2* $a\ b\ a'\ b' = ()$ with type annotation $\forall\alpha,\beta.\ \alpha \to \beta \to \alpha \to \beta \to \mathbf{1}$. Given the context $id : \forall\delta.\ \delta{\to}\delta, uf : \mathbf{1} \to \mathbf{1}$, synthesizing a type for the application *nonlinear2 uf uf id id* involves several nondeterministic choices of when to instantiate each of the *id* types. Still, this can be checked with only a few calls to the function that attempts to derive a subtyping judgment, and this continues to hold as we add arguments according to the same pattern. But if we introduce a type error, even an obvious one like an extra argument *nonlinear2 uf uf id id id*, then by the time we reach the 14-argument function *nonlinear7* it takes 87,000 subtyping ($\leqq$) calls and 49 seconds to reject the program. This is a contrived example, and I have not yet found a real example that makes typechecking unacceptably slow.

Note that intersection types make these systems PSPACE-hard (Reynolds 1996), even if parametric polymorphism is never used, and typechecking can be very slow when intersections and unions are used extensively (Dunfield 2007a).

7. Related Work

For impredicative System F without annotations, type inference is undecidable (Wells 1999); it becomes decidable if quantifiers are restricted to rank 2 or less (Kfoury and Wells 1994).

Peyton Jones et al. (2007) developed a bidirectional system that supports arbitrary-rank, but predicative, polymorphism (quantifiers can appear anywhere in types, but polymorphic instances must be monotypes). Their system does not support subtyping, except for "at least as polymorphic as" subtyping (which we write as $\leqq$).

My bidirectional systems are strictly weaker than Damas-Milner: they require annotations on redexes (though that requirement could be weakened by adding synthesis rules for some syntactic forms), and don't manufacture quantifiers by generalizing type variables. $\mathsf{ML^F}$ (Le Botlan and Rémy 2003), a type inference system in the Damas-Milner tradition, supports impredicative polymorphism, with annotations needed only for impredicative instantiations (similar to my predicative completeness). $\mathsf{ML^F}$ is more powerful than my systems, but appears substantially more complicated, even in its revised form (Rémy and Yakobowski 2008).

HML (Leijen 2009) extends Damas-Milner and has similar goals to $\mathsf{ML^F}$. HML infers *flexible types*, polymorphic types that are bounded below, as $\forall(\beta \geq \forall\alpha.\ \alpha{\to}\alpha).\ \beta \to \beta$. HML requires annotations only on polymorphic arguments, and is a good deal simpler than $\mathsf{ML^F}$. It is robust under many simple transformations, such as *revapp* $e_2\ e_1$ in place of $e_1\ e_2$ (where *revapp* has type $\forall\alpha,\beta.\ \alpha \to (\alpha{\to}\beta) \to \beta$). In contrast, System $\mathrm{Bi}^{\widehat{\alpha}}$ is sensitive to the ordering of terms when impredicative polymorphism is used; in the failed derivation in Figure 8, swapping the arguments x and y results in success.

In systems with subtyping, several approaches to inferring polymorphic instances have been presented.

In *local type inference* (Pierce and Turner 2000), instances are found by computing upper and lower bounds on types, using information propagated *locally* within the program.

Colored local type inference (Odersky et al. 2001) is akin to Pierce and Turner's approach, but also allows different *parts* of type expressions to be propagated in different directions. My approach gets a similar effect by manipulating type expressions with $\widehat{\alpha}$-variables, which allows us to fix part of the type expression (the part that is not $\widehat{\alpha}$) while $\widehat{\alpha}$ remains flexible.

Davies' Refinement ML (Davies 2005), an extension of Standard ML with intersection types and subtyping, has a *refinement restriction*: the intersection $A \wedge B$ can be formed only if A and B are refinements (subtypes) of the same simple type. It is thus possible in his setting to do ordinary Damas-Milner SML type inference to find simple-type instances of polymorphic variables. In his system, there are only finitely many subtypes of a given simple type, so the one that will make typechecking succeed can be found (in theory, and often in practice), by exhaustive search.

8. Conclusion

I have presented a new approach to inferring polymorphic instances in a bidirectional setting. This paper applies this approach to first-class polymorphism, without subtyping. At first, my goal was simply to add parametric polymorphism to the type systems described in my dissertation—the application to first-class polymorphism was a pleasant surprise. System $\mathrm{Bi}^{\widehat{\alpha}}$ is a "light" version of a rich System $\mathrm{Bi}^{\leq\widehat{\alpha}}$ with subtyping, intersection, and union types. As describing both systems would be (or, shall we say, *was*) on the long side for a conference, I intend to write a journal article; in the meantime, see Dunfield (2009).

The type systems in this paper might seem odd at first. System $\mathrm{Bi}^{\widehat{\alpha}}$, which is not inherently exotic—it lacks intersections and unions—looks quite different from previous approaches to first-class polymorphism. Even those that use bidirectionality, such as Peyton Jones et al. (2007), are rooted in the Damas-Milner inference tradition. My work here is rooted elsewhere (Dunfield and Pfenning 2004). I attribute the virtues of my work to the essential simplicity of bidirectional typechecking.

The systems in this paper, like those in its immediate ancestors (my dissertation and the works of Xi, Davies, Pfenning), are meant for typechecking, not elaboration/compilation: they do not insert explicit polymorphic abstractions and applications. Reformulating System $\mathrm{Bi}^{\widehat{\alpha}}$ in an elaboration style looks straightforward, though.

In addition to investigating elaboration and compilation, I plan to extend this work to GADTs. With bidirectionality and existential type variables, I expect this to be relatively easy.

To designers of languages and type systems, consider bidirectional typechecking; as your type system becomes more powerful, you will likely outgrow Damas-Milner inference, and making it bidirectional from the beginning should lead to a cleaner and more logical system than what you get after retrofitting bidirectionality. If you don't need subtyping, polymorphism is nearly free with your purchase of bidirectionality; if you do need subtyping, polymorphism is nearly free with your purchase of intersections and unions (stay tuned).

Acknowledgments. Thanks to Frank Pfenning, Sungwoo Park, Brigitte Pientka, an anonymous ICFP 2009 reviewer, and the ML Workshop reviewers for (variously) discussions, encouragement and useful comments on several drafts of this paper. A few parts of Stardust are based on code by Tolmach and Oliva (1998).

References

Andreas Abel. Termination checking with types. *RAIRO—Theoretical Informatics and Applications*, 38(4):277–319, 2004. Special Issue: Fixed Points in Computer Science (FICS'03).

Andreas Abel, Thierry Coquand, and Peter Dybjer. Verifying a semantic βη-conversion test for Martin-Löf type theory. In *Mathematics of Program Construction (MPC'08)*, volume 5133 of *LNCS*, pages 29–56, 2008.

Luca Cardelli. An implementation of $F_{<:}$. Research report 97, DEC/Compaq Systems Research Center, February 1993.

Adam Chlipala, Leaf Petersen, and Robert Harper. Strict bidirectional type checking. In *Workshop on Types in Language Design and Impl. (TLDI '05)*, pages 71–78, 2005.

Thierry Coquand. An algorithm for type-checking dependent types. *Science of Computer Programming*, 26(1–3):167–177, 1996.

Rowan Davies. *Practical Refinement-Type Checking*. PhD thesis, Carnegie Mellon University, 2005. CMU-CS-05-110.

Rowan Davies and Frank Pfenning. Intersection types and computational effects. In *ICFP*, pages 198–208, 2000.

Jana Dunfield. Refined typechecking with Stardust. In *Programming Languages meets Programming Verification (PLPV '07)*, 2007a.

Jana Dunfield. *A Unified System of Type Refinements*. PhD thesis, Carnegie Mellon University, 2007b. CMU-CS-07-129.

Jana Dunfield. Bidirectional polymorphism through greed and unions. Unpublished manuscript, April 2009. URL `http://type-refinements.info/bipoly/`.

Jana Dunfield and Frank Pfenning. Type assignment for intersections and unions in call-by-value languages. In *Found. Software Science and Computation Structures (FOSSACS '03)*, pages 250–266, 2003.

Jana Dunfield and Frank Pfenning. Tridirectional typechecking. In *POPL*, pages 281–292, January 2004.

Andrew Gill, John Launchbury, and Simon L Peyton Jones. A short cut to deforestation. In *Functional Programming and Computer Architecture*, pages 223–232, 1993.

Jean-Yves Girard. The system *F* of variable types, fifteen years later. *Theoretical Computer Science*, 45(2):159–192, 1986.

A. J. Kfoury and J. B. Wells. A direct algorithm for type inference in the rank-2 fragment of the second-order λ-calculus. In *LISP and Functional Programming*, pages 196–207, 1994.

Didier Le Botlan and Didier Rémy. ML^F: raising ML to the power of System F. In *ICFP*, pages 27–38, 2003.

Daan Leijen. Flexible types: robust type inference for first-class polymorphism. In *POPL*, pages 66–77, January 2009.

Andres Löh, Conor McBride, and Wouter Swierstra. A tutorial implementation of a dependently typed lambda calculus. Unpublished draft, `http://people.cs.uu.nl/andres/LambdaPi/index.html`, 2008.

William Lovas and Frank Pfenning. A bidirectional refinement type system for LF. In *Int'l Workshop on Logical Frameworks and Meta-languages*, Electronic Notes in Theoretical Computer Science, pages 113–128. Elsevier, July 2007.

Ulf Norell. *Towards a Practical Programming Language Based on Dependent Type Theory*. PhD thesis, Dept. of Computer Science and Engineering, Chalmers University of Technology, 2007.

Martin Odersky, Matthias Zenger, and Christoph Zenger. Colored local type inference. In *POPL*, pages 41–53, 2001.

Simon Peyton Jones, Dimitrios Vytiniotis, Stephanie Weirich, and Mark Shields. Practical type inference for arbitrary-rank types. *J. Functional Programming*, 17(1):1–82, 2007.

Brigitte Pientka. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *POPL*, pages 371–382, 2008.

Brigitte Pientka and Jana Dunfield. Programming with proofs and explicit contexts. In *Principles and Practice of Declarative Programming (PPDP'08)*, pages 163–173, 2008.

Benjamin C. Pierce. Programming with intersection types, union types, and polymorphism. Technical Report CMU-CS-91-106, Carnegie Mellon University, 1991.

Benjamin C. Pierce and David N. Turner. Local type inference. *ACM Trans. Programming Languages and Systems*, 22:1–44, 2000.

Didier Rémy and Boris Yakobowski. From ML to ML^F: graphic type constraints with efficient type inference. In *ICFP*, pages 63–74, 2008.

John C. Reynolds. Towards a theory of type structure. In *Colloque sur la Programmation*, volume 19 of *LNCS*, pages 408–425. Springer, 1974. `http://www.cs.cmu.edu/afs/cs/user/jcr/ftp/theotypestr.pdf`.

John C. Reynolds. Design of the programming language Forsythe. Technical Report CMU-CS-96-146, Carnegie Mellon University, 1996.

Andrew Tolmach and Dino P. Oliva. From ML to Ada: Strongly-typed language interoperability via source translation. *J. Functional Programming*, 8(4):367–412, 1998.

J. B. Wells. Typability and type checking in System F are equivalent and undecidable. *Annals of Pure and Applied Logic*, 98:111–156, 1999.

Hongwei Xi. *Dependent Types in Practical Programming*. PhD thesis, Carnegie Mellon University, 1998.

Appendix: Decidability, Soundness, Completeness

Lemma 11 (Well-Formedness) *Proof.* By induction on $\mathcal{D}$. In the 6 rules that introduce existential solutions, the well-formedness of the solution is either explicit ($\widehat{\alpha}^{=}$L$\leqq$, $\widehat{\alpha}^{=}$R$\leqq$) or is evident from the context ($\rightarrow$I$\widehat{\alpha}$, $\rightarrow$E$\widehat{\alpha}$, $\rightarrow\widehat{\alpha}L\leqq$, $\rightarrow\widehat{\alpha}R\leqq$). $\square$

Theorem 14 *Proof.* By induction on the given derivation. We show the $\forall$E case. Let $\Gamma_{HH} = (\Gamma_H, \mathsf{hint}(\Gamma \vdash A'))$. By IH, $\Gamma_{HH}, \Gamma_H', \Gamma \vdash e \Uparrow \forall\alpha.\, A \dashv \Gamma_{HH}, \Gamma$. By Corollary 8, $(\Gamma \vdash A') \lesssim (\Gamma_H, \Gamma \vdash A')$. Finally, by $\forall$E-hint, $\Gamma_{HH}, \Gamma_H', \Gamma \vdash e \Uparrow [A'/\alpha]A \dashv \Gamma_H, \Gamma$, which is $\Gamma_H, \mathsf{hint}(\Gamma \vdash A'), \Gamma_H', \Gamma \vdash e \Uparrow [A'/\alpha]A \dashv \Gamma_H, \Gamma$, which was to be shown. $\square$

Corollary 15 *Proof.* By Theorem 14, $\Gamma_H \vdash e \Downarrow A \dashv \cdot$ where Γ_H consists of n hints. The result follows by applying the hint rule n times. $\square$

Definition 18 (Ordering of subtyping judgments)**.**
Given $\mathcal{J}_1 = \Gamma_1 \vdash A_1 \leqq B_1 \dashv \dots$
and $\mathcal{J}_2 = \Gamma_2 \vdash A_2 \leqq B_2 \dashv \dots$,
the order $\prec$ is defined lexicographically by

(1) the numbers of hints in Γ_1 and in Γ_2, under $<$;
(2) if $B_1 = B_2$ and $\Gamma_1 = \Gamma_2$, the angst of A_1 versus A_2; or, if $A_1 = A_2$ and $\Gamma_1 = \Gamma_2$, the angst of B_1 versus B_2;
(3) $\{A_1, B_1\} \prec \{A_2, B_2\}$;
(4) $A_1 = A_2$ and $B_1 = B_2$ where all existential variables in $A_1 (= A_2)$ are solved in Γ_1 but not in Γ_2; or, the same, swapping B_1 and B_2 for A_1 and A_2.

Definition 19 (Ordering of typing judgments)**.**
Given $\mathcal{J}_1 = \Gamma_1 \vdash e_1 \Uparrow/\Downarrow C_1 \dashv \Gamma_1'$
and $\mathcal{J}_2 = \Gamma_2 \vdash e_2 \Uparrow/\Downarrow C_2 \dashv \Gamma_2'$,
we define $\mathcal{J}_1 \preceq \mathcal{J}_2$ by the lexicographic ordering of:

(1) e_1 and e_2 (subterm ordering);
(2) the directions, considering $\Uparrow$ smaller than $\Downarrow$;
(3a) If both are checking judgments:
 (i) $C_1 \preceq C_2$;
 (ii) $\Gamma_1 = \Gamma_2$ and C_1 has less angst then C_2; or
 (iii) all $\widehat{\alpha}$-variables in $C_1 (= C_2)$ are solved in Γ_1 but not in Γ_2.
(3b) If both are synthesis judgments:
 (i) the number of hints in Γ_1' versus Γ_2'; if equal,
 (ii) $C_2 \preceq C_1$;
 (iii) C_2 has less angst with respect to Γ_2' than C_1 w.r.t. Γ_1'.

Theorem (Decidability of Subtyping and Contextual Matching)**.** *Given Γ, A, and B, the existence of Γ' such that $\Gamma \vdash A \leqq B \dashv \Gamma'$ in System Bi$^{\widehat{\alpha}}$ is decidable. Moreover, given Γ_0, A_0 and Γ, the existence of A such that $(\Gamma \vdash A_0) \lesssim (\Gamma \vdash A)$ is decidable.*

Proof. By showing that the premises of each rule are smaller, under the defined partial order, than the conclusion.

For contextual matching, the rule $\lesssim$-empty has no premises, and the other rules make Γ_0 shorter. ☐

Theorem (Decidability of Typing)**.** ***1.*** *Given Γ, e, and C, it is decidable whether there exists Γ' such that $\Gamma \vdash e \Downarrow C \dashv \Gamma'$.*
2. *Given Γ and e it is decidable whether there exist Γ' and C such that $\Gamma \vdash e \Uparrow C \dashv \Gamma'$.*

Proof. We show that the premises of each rule are smaller, under the defined partial order, than the conclusion. Note that in each rule, we have enough information to apply the induction hypothesis for each premise. For example, in $\to$E, we have $e = e_1 e_2$, giving us an e_1 for $\to$E's synthesizing premise; applying the IH there gives a type for the second, checking, premise.

For anno, $\to$I, $\to$E, hint, $\to$E$\widehat{\alpha}$, use part (1). For sub, use part (2) and the previous theorem. For $\forall$E-hint, use part (3b)(i). Contextual matching is decidable by the previous theorem.

For $\forall$I, use part (3a)(i); for $\forall$E$\widehat{\alpha}$, part (3b)(ii).

For ExSubst$\Downarrow$ and ExSubst$\Uparrow$, use parts (3a)(ii) and (3b)(iii) respectively. For $\to$I$\widehat{\alpha}$, use part (3a)(iii). ☐

Theorem 13 (Soundness of System Bi$^{\widehat{\alpha}}$).

Proof. Since Ω completes Γ', we have $\Omega \supseteq \Gamma'$: any variable $\widehat{\alpha}$ that is solved in Γ' is also solved *and* $[\Omega]\widehat{\alpha} = [\Omega][\Gamma']\widehat{\alpha}$. Moreover, from Lemma 12, $\Gamma' \supseteq \Gamma$. Since $\supseteq$ is a transitive relation, any $\widehat{\alpha}$ solved in Γ is solved and has the same solution in Ω.

When applying the IH, we must ensure that the Ω and Γ' are in sync. For example, in $\forall$I the output context in the subderivation is $\Gamma', \alpha, \Gamma_Z$ while the output context for the derivation is Γ'. The given Ω completes Γ', not $\Gamma', \alpha, \Gamma_Z$, so it must be extended: Add solutions in Γ_Z to Ω; for unsolved variables $\widehat{\beta}$, choose *any* well-formed type B—**1** is a good choice since it has no free type variables and is thus well-formed in every context—and add $\widehat{\beta}{=}B$ to Ω. This works because $\forall$I strips out all the declarations in Γ_Z, so $\widehat{\beta}$ is about to leave this world unsolved, and therefore unconstrained.

In the $\forall$E$\widehat{\alpha}$ case, the IH gives $[\Omega]\Gamma \vdash e \Uparrow \forall\alpha.\, [\Omega]A$. Since Ω is solved, $\widehat{\alpha}{=}A' \in \Omega$, and by Lemma 11, $\Gamma \vdash A'$ wf. By Lemma 10, $[\Omega]\Gamma \vdash [\Omega]A'$ wf. By $\forall$E, $[\Omega]\Gamma \vdash e \Uparrow [[\Omega]A'/\alpha]([\Omega]A)$. By a property of substitutions, $[[\Omega]A'/\alpha]([\Omega]A) = [\Omega][A'/\alpha]A$, giving the result.

In the ExSubst$\Downarrow$ case, the IH yields $[\Omega]\Gamma \vdash e \Downarrow [\Omega]\,\Gamma(\widehat{\alpha})$; the variable $\widehat{\alpha}$ cannot be free in $\Gamma(\widehat{\alpha})$, and we earlier noted that $[\Omega]\widehat{\alpha} = [\Omega][\Gamma]\widehat{\alpha}$, so in fact $[\Omega]\,\Gamma(\widehat{\alpha}) = [\Omega]\,\Omega(\widehat{\alpha}) = [\Omega]\widehat{\alpha}$, giving the result. ExSubst$\Uparrow$ and ExSubst{L,R}$\leq$ are similar.

In the $\to$I$\widehat{\alpha}$ case, the IH gives $[\Omega]\Gamma, x{:}[\Omega]\widehat{\alpha_1} \vdash e_0 \Downarrow [\Omega]\widehat{\alpha_2}$. By $\to$I, $[\Omega]\Gamma \vdash \lambda x.\, e_0 \Downarrow ([\Omega]\widehat{\alpha_1}) \to ([\Omega]\widehat{\alpha_2})$. The declaration $\widehat{\alpha}{=}\widehat{\alpha_1} \to \widehat{\alpha_2}$ is in Γ, so by Lemma 12 it is also in Ω. Thus, we have $\ldots \Downarrow [\Omega]\widehat{\alpha}$, which was to be shown.

In the $\widehat{\alpha}^{=}$L$\leq$ case, we have $(\widehat{\alpha}{=}B) \in \Gamma'$. By Lemma 12, $(\widehat{\alpha}{=}B) \in \Omega$, so $[\Omega]\widehat{\alpha} = [\Omega]B$. The result follows by reflexivity of $\leqq$. The $\widehat{\alpha}^{=}$R$\leq$ case is symmetric.

The $\to\widehat{\alpha}$L$\leq$, $\to\widehat{\alpha}$R$\leq$ cases use similar reasoning as the $\to$I$\widehat{\alpha}$ case. The remaining cases are straightforward. ☐

Theorem 17 (Predicative Completeness).

Proof. By induction on $\mathcal{D}$. Note that the type C' in the consequent is well-formed under Γ_2'—and not necessarily under Γ_1', as Γ_2' may have existential type variables that Γ_1' does not.

- **Case** $\to\leqq$: $\mathcal{D} :: \dfrac{\Gamma \vdash B_1 \leqq A_1 \qquad \Gamma \vdash A_2 \leqq B_2}{\Gamma \vdash A_1 \to A_2 \leqq B_1 \to B_2}$
 We know that $[\Omega]A' = A_1 \to A_2$. Either $\{\to A'$ case$\}$ $A' = A_1' \to A_2'$ (so $[\Omega]A' = [\Omega]A_1' \to [\Omega]A_2' = A_1 \to A_2$) or $\{\widehat{\alpha}A'$ case$\}$ $A' = \widehat{\alpha}$ (so $[\Omega]A' = [\Omega]\widehat{\alpha}$). Similarly, we distinguish $\{\to B'$ case$\}$ and $\{\widehat{\beta}B'$ case$\}$ depending on whether B' is $B_1' \to B_2'$ or $\widehat{\beta}$. (Note that possibly $\widehat{\beta} = \widehat{\alpha}$.)
 - $\{\to A'$ and $\to B'$ case$\}$:
 By IH, $\Gamma_1' \vdash B_1' \leqq A_1' \dashv \Gamma_2'$, and again: $\Gamma_2' \vdash A_2' \leqq B_2' \dashv \Gamma_3'$. By $\to\leqq$, $\Gamma_1' \vdash A_1' \to A_2' \leqq B_1' \to B_2' \dashv \Gamma_3'$.
 - $\{\widehat{\alpha}A'$ and $\to B'$ case$\}$:
 $\Gamma_1' \vdash A_1' \to A_2' \leqq B_1' \to B_2' \dashv \Gamma_3'$ As preceding case
 If Γ_1' includes a solution for $\widehat{\alpha}$, then:
 ☞ $\Gamma_1' \vdash \widehat{\alpha} \leqq B_1' \to B_2' \dashv \Gamma_3'$ By ExSubstL$\leqq$
 Otherwise, Γ_1' does not include a solution for $\widehat{\alpha}$. $\Omega(\widehat{\alpha}) = [\Omega]A' = A_1 \to A_2$ must have the form $\widehat{\alpha_1} \to \widehat{\alpha_2}$, because Ω is predicative and articulated. We assumed that Γ_1' does not include a solution for $\widehat{\alpha}$, so $\Gamma_1' = \Gamma_L, \widehat{\alpha}, \Gamma_R$. Let $\Gamma_+ = \Gamma_L, \widehat{\alpha_1}, \widehat{\alpha_2}, \widehat{\alpha}{=}\widehat{\alpha_1}{\to}\widehat{\alpha_2}, \Gamma_R$.
 $\Gamma_+ \vdash B_1' \leqq \widehat{\alpha_1} \dashv \Gamma_M$ By IH on $\Gamma \vdash B_1 \leqq A_1$, taking $\Gamma_L, \widehat{\alpha_1}, \widehat{\alpha_2}, \widehat{\alpha}{=}\widehat{\alpha_1}{\to}\widehat{\alpha_2}$ as Γ_1'
 $\Gamma_M \vdash \widehat{\alpha_2} \leqq B_2' \dashv \Gamma_2'$ By IH
 $\Gamma_+ \vdash \widehat{\alpha_1} \to \widehat{\alpha_2} \leqq B_1' \to B_2' \dashv \Gamma_2'$ By $\to\leqq$
 $\Gamma_+ \vdash \widehat{\alpha} \leqq B_1' \to B_2' \dashv \Gamma_2'$ By ExSubstL$\leqq$
 ☞ $\Gamma_1' \vdash \widehat{\alpha} \leqq B_1' \to B_2' \dashv \Gamma_2'$ By $\to\widehat{\alpha}$L$\leqq$
 - $\{\to A'$ and $\widehat{\beta}B'$ case$\}$: Symmetric to the previous.
 - $\{\widehat{\alpha}A'$ and $\widehat{\beta}B'$ case$\}$: If either $\widehat{\alpha}$ or $\widehat{\beta}$ is solved in Γ_1', then the solution in Γ_1' has an $\to$ at its head (since the solution in Ω does). Using suitably articulated contexts, use the IH, then use ExSubst and $\to\widehat{\alpha}$L$\leqq$ or $\to\widehat{\alpha}$R$\leqq$ as needed.
 If neither is solved and $\widehat{\alpha} = \widehat{\beta}$, then the result follows by $\widehat{\alpha}$Refl$\leqq$. Otherwise, neither is solved and $\widehat{\alpha} \neq \widehat{\beta}$. So add a solution for whichever of $\widehat{\alpha}$ and $\widehat{\beta}$ is declared last in Γ_1'. Suppose without loss of generality that $\Gamma_1' = \Gamma_L, \widehat{\alpha}, \Gamma_C, \widehat{\beta}, \Gamma_R$.
 $\Gamma_1' \vdash \widehat{\alpha} \leqq \widehat{\beta} \dashv \Gamma_L, \widehat{\alpha}, \Gamma_C, \widehat{\beta}{=}\widehat{\alpha}, \Gamma_R$ By $\widehat{\alpha}^{=}$R$\leqq$
 To show that applying Ω to the output context yields Ω, note that because Ω is predicative, $[\Omega]\widehat{\alpha}$ and $[\Omega]\widehat{\beta}$ are monomorphic. We have $\Gamma \vdash [\Omega]\widehat{\alpha} \leqq [\Omega]\widehat{\beta}$, so by Lemma 16, $[\Omega]\widehat{\alpha} = [\Omega]\widehat{\beta}$. Thus, $\widehat{\alpha}$ and $\widehat{\beta}$ have the same solution in Ω: the solution $\widehat{\beta}{=}\widehat{\alpha}$ is consistent with Ω, so $[\Omega](\Gamma_L, \widehat{\alpha}, \Gamma_C, \widehat{\beta}{=}\widehat{\alpha}, \Gamma_R) = \Omega$.

- **Case** αRefl$\leqq$: We have $\alpha = [\Omega]A' = [\Omega]B'$. The types A' and B' can each be α or various existential variables.

If $A' = B' = \alpha$, the result follows by αRefl$\leqq$, giving $\Gamma'_1 \vdash \alpha \leqq \alpha \dashv \Gamma'_1$.

If $A' = \alpha$ and B' is some solved $\widehat{\beta}$, the result follows by αRefl$\leqq$, yielding $\Gamma'_1 \vdash \alpha \leqq \alpha \vdash \Gamma'_1$ then ExSubstR$\leqq$ for $\Gamma'_1 \vdash \alpha \leqq \widehat{\beta} \dashv \Gamma'_1$.

If $\widehat{\beta}$ is unsolved: $\widehat{\beta}$ is well-formed in Γ'_1, so $\Gamma'_1 = \Gamma_L, \widehat{\beta}, \Gamma_R$. Applying $\widehat{\alpha}{=}R\leqq$ gives $\Gamma_L, \widehat{\beta}, \Gamma_R \vdash \alpha \leqq \widehat{\beta} \dashv \Gamma_L, \widehat{\beta}{=}\alpha, \Gamma_R$. Let $\Gamma'_2 = \Gamma_L, \widehat{\beta}{=}\alpha, \Gamma_R$. Substituting gives $\Gamma'_1 \vdash \alpha \leqq \widehat{\beta} \dashv \Gamma'_2$, which was to be shown.

The subcases where $B' = \alpha$ and A' is some solved $\widehat{\beta}$ are symmetric to the last two.

If $A' = \widehat{\gamma}$ and $B' = \widehat{\beta}$, first apply αRefl$\leqq$, then:

- If both are solved in Γ'_1, apply ExSubstL$\leqq$ then ExSubstR$\leqq$.
- If only $\widehat{\gamma}$ is solved, apply ExSubstL$\leqq$ then $\widehat{\alpha}{=}R\leqq$.
- If only $\widehat{\beta}$ is solved, apply ExSubstR$\leqq$ then $\widehat{\alpha}{=}L\leqq$.
- If neither is solved: Both $\widehat{\gamma}$ and $\widehat{\beta}$ are well-formed under Γ'_1. Either $\widehat{\gamma}$ comes first or $\widehat{\beta}$ comes first. Suppose $\widehat{\beta}$ comes first. Then $\widehat{\alpha}{=}L\leqq$ gives $\Gamma'_1 \vdash \widehat{\gamma} \leqq \widehat{\beta} \dashv \ldots, \widehat{\alpha}{=}\widehat{\beta}, \ldots$.

- **Case** $\mathbf{1}\leqq$: Similar to the previous case.

- **Case** $\forall L\leqq$: $\mathcal{D} :: \dfrac{\Gamma \vdash [C/\alpha]A_0 \leqq B \qquad \Gamma \vdash C\ \mathsf{wf}}{\Gamma \vdash \forall\alpha.\, A_0 \leqq B}$

We know that $[\Omega]A' = \forall\alpha.\, A_0$. Either $\{\forall A'\text{ case}\}$ $A' = \forall\alpha.\, A'_0$, so $[\Omega]A' = \forall\alpha.\, [\Omega]A'_0$, or $\{\widehat{\gamma}A\text{ case}\}$ $A' = \widehat{\gamma}$ so $[\Omega]\widehat{\gamma} = \forall\alpha.\, \ldots$, which is impossible by the assumption that Ω is predicative.

- $\{\forall A'\text{ case}\}$: Choose a fresh $\widehat{\alpha}$. Let $\Omega' = \mathit{Artic}(\widehat{\alpha}{=}C)$.

$$\begin{array}{rcll}
A_0 &=& [\Omega]A'_0 & \text{Above} \\
[C/\alpha]A_0 &=& [C/\alpha][\Omega]A'_0 & \text{Applying } [C/\alpha] \\
&=& [\Omega]([C/\alpha]A'_0) & \text{Permutation} \\
&=& [\Omega]([C/\widehat{\alpha}][\widehat{\alpha}/\alpha]A'_0) & \widehat{\alpha}\text{ fresh} \\
&=& [\Omega, \mathit{Artic}(\widehat{\alpha}{=}C)][\widehat{\alpha}/\alpha]A'_0 & \text{Defs. of artic., subst.} \\
&=& [\Omega, \Omega'][\widehat{\alpha}/\alpha]A'_0 & \text{Def. of } \Omega' \text{ above}
\end{array}$$

So $[C/\alpha]A_0 = [\Omega, \Omega']([\widehat{\alpha}/\alpha]A'_0)$.

$$\begin{array}{lrl}
& \Gamma'_1, \blacktriangleleft_{\widehat{\alpha}}, \widehat{\alpha} \vdash [\widehat{\alpha}/\alpha]A'_0 \leqq B' \dashv \Gamma_R & \text{By IH w/ } (\Omega, \Omega') \\
& \Gamma_R = \Gamma'_2, \blacktriangleleft_{\widehat{\alpha}}, \Gamma_Z & \text{By Lemma 12} \\
☞ & \Gamma'_1 \vdash \forall\alpha.\, A' \leqq B' \dashv \Gamma'_2 & \text{By } \forall L\widehat{\alpha}\leqq
\end{array}$$

- **Case** $\forall R\leqq$: $\mathcal{D} :: \dfrac{\Gamma, \beta \vdash A \leqq B_0}{\Gamma \vdash A \leqq \forall\beta.\, B_0}$

We know that $[\Omega]B' = \forall\beta.B_0$. Either $\{\forall B'\text{ case}\}$ $B' = \forall\beta.B'_0$ (so $[\Omega]B' = \forall\beta.\, [\Omega]B'_0$) or $\{\widehat{\gamma}B\text{ case}\}$ $B' = \widehat{\gamma}$.

- $\{\forall B'\text{ case}\}$:

$$\begin{array}{lrl}
& \Gamma'_1, \beta \vdash A' \leqq B' \dashv \Gamma''_2 & \text{By IH} \\
& \Gamma''_2 = \Gamma'_2, \beta, \Gamma_Z & \text{By } \overline{\Gamma''_2} = \Gamma \text{ (by Lemma 12)} \\
☞ & \Gamma'_1 \vdash A' \leqq \forall\beta.\, B'_1 \dashv \Gamma'_2 & \text{By } \forall R\leqq
\end{array}$$

- $\{\widehat{\gamma}B'\text{ case}\}$: Applying Ω to $B' = \widehat{\gamma}$ gives $[\Omega]B' = [\Omega]\widehat{\gamma}$, which is equal to $\Omega(\widehat{\gamma})$. But since $[\Omega]B' = \forall\beta.B_0$, we have $\Omega(\widehat{\gamma}) = \forall\beta.\, B_0$, which contradicts our assumption that Ω is predicative: this case is impossible.

- **Case** var: $\Gamma = [\Omega]\Gamma'_1$. Therefore $\Gamma(x) = [\Omega](\Gamma'_1(x))$. So $\Gamma'_1(x) = A'$ where $[\Omega]A' = A$. The result, $\Gamma'_1 \vdash x \Uparrow A' \dashv \Gamma'_1$, follows by var.

- **Case** sub: $\mathcal{D} :: \dfrac{\Gamma \vdash e \Uparrow B \qquad \Gamma \vdash B \leqq A}{\Gamma \vdash e \Downarrow A}$

By IH, $\Gamma'_1 \vdash e \Uparrow B' \dashv \Gamma_M$ where $[\Omega]B' = B$. We have $[\Omega]A' = A$. By IH, $\Gamma_M \vdash B' \leqq A' \dashv \Gamma'_2$. Then use sub.

- **Case** anno: $\mathcal{D} :: \dfrac{N \lesssim (\Gamma \vdash A) \qquad \Gamma \vdash e \Downarrow A}{\Gamma \vdash (e : N) \Uparrow A}$

The result follows by the IH and anno. (The $\lesssim$ premise of anno in System Bi$^{\widehat{\alpha}}$ does not involve existential contexts.)

- **Case** $\to$I: $\mathcal{D} :: \dfrac{\Gamma, x{:}A_1 \vdash e \Downarrow A_2}{\Gamma \vdash \lambda x.\, e \Downarrow A_1 \to A_2}$

If $A' = A'_1 \to A'_2$ (with $[\Omega]A'_1 = A_1$ and $[\Omega]A'_2 = A_2$): The IH gives $\Gamma'_1, x{:}A'_1 \vdash e \Downarrow A'_2 \dashv \Gamma_M$. By Lemma 5, $\overline{\Gamma_M} = \overline{\Gamma'_1}$; then, by Lemma 6, $\Gamma_M = \Gamma'_2, x{:}A'_1, \Gamma_R$. Applying $\to$I gives $\Gamma'_1 \vdash \lambda x.\, e \Downarrow A'_1 \to A'_2 \dashv \Gamma'_2$, which was to be shown.

Otherwise, $A' = \widehat{\alpha}$ and $\Omega(\widehat{\alpha}) = \widehat{\alpha_1} \to \widehat{\alpha_2}$, where $A_1 = [\Omega]\widehat{\alpha_1}$ and $A_2 = [\Omega]\widehat{\alpha_2}$.

- $\{\text{solved case}\}$: $\widehat{\alpha}$ solved in Γ'_1; since Γ'_1 is articulated, $\widehat{\alpha}{=}\widehat{\alpha_1} \to \widehat{\alpha_2} \in \Gamma'_1$.

$$\begin{array}{lrl}
& \Gamma'_1, x{:}\widehat{\alpha_1} \vdash e \Downarrow \widehat{\alpha_2} \dashv \Gamma'_2, x{:}\widehat{\alpha_1}, \Gamma_R & \text{By IH} \\
& \Gamma'_1 \vdash \lambda x.\, e \Downarrow \widehat{\alpha_1} \to \widehat{\alpha_2} \dashv \Gamma'_2 & \text{By} \to\text{I} \\
☞ & \Gamma'_1 \vdash \lambda x.\, e \Downarrow \widehat{\alpha} \dashv \Gamma'_2 & \text{By ExSubst}\Downarrow
\end{array}$$

- $\{\text{not-solved case}\}$: $\widehat{\alpha}$ not solved in Γ'_1: decompose Γ'_1 into $\Gamma_{11}, \widehat{\alpha}, \Gamma_{12}$. Let $\Gamma_\alpha = \widehat{\alpha_1}, \widehat{\alpha_2}, \widehat{\alpha}{=}\widehat{\alpha_1}{\to}\widehat{\alpha_2}$.

$$\begin{array}{lrl}
& \Gamma_{11}, \Gamma_\alpha, \Gamma_{12}, x{:}\widehat{\alpha_1} \vdash e \Downarrow \widehat{\alpha_2} \dashv \Gamma_L, \Gamma_\alpha, \Gamma_{12}, x{:}\widehat{\alpha_1}, \Gamma_R & \text{By IH} \\
& \Gamma_{11}, \Gamma_\alpha, \Gamma_{12} \vdash \lambda x.\, e \Downarrow \widehat{\alpha_1} \to \widehat{\alpha_2} \dashv \Gamma_L, \Gamma_\alpha, \Gamma_{12} & \text{By} \to\text{I} \\
& \Gamma_{11}, \Gamma_\alpha, \Gamma_{12} \vdash \lambda x.\, e \Downarrow \widehat{\alpha} \dashv \Gamma_L, \Gamma_\alpha, \Gamma_{12} & \text{By ExSubst}\Downarrow \\
☞ & \Gamma_{11}, \widehat{\alpha}, \Gamma_{12} \vdash \lambda x.\, e \Downarrow \widehat{\alpha} \dashv \Gamma_L, \Gamma_\alpha & \text{By} \to\text{I}\widehat{\alpha}
\end{array}$$

- **Case** $\to$E: $\mathcal{D} :: \dfrac{\Gamma \vdash e_1 \Uparrow B \to A \qquad \Gamma \vdash e_2 \Downarrow B}{\Gamma \vdash e_1\, e_2 \Uparrow A}$

By IH, $\Gamma'_1 \vdash e_1 \Uparrow C' \dashv \Gamma_M$ where $[\Omega]C' = B \to A$. If $C' = B' \to A'$ then $[\Omega]B' = B$ and $[\Omega]A' = A$. By IH, $\Gamma_M \vdash e_2 \Downarrow B' \dashv \Gamma'_2$. The result is by $\to$E.

Otherwise, $C' = \widehat{\alpha}$ and $\Omega(\widehat{\alpha}) = \widehat{\alpha_1} \to \widehat{\alpha_2}$. Since $[\Omega]C' = B \to A$, we have $[\Omega]\widehat{\alpha_1} = B$ and $[\Omega]\widehat{\alpha_2} = A$. The IH told us that C' is well-formed under Γ_M, so $\widehat{\alpha}$ must be defined within Γ_M, that is, $\Gamma_M = \Gamma_L, \widehat{\alpha}, \Gamma_R$. So the IH really gave us $\Gamma'_1 \vdash e_1 \Uparrow \widehat{\alpha} \dashv \Gamma_L, \widehat{\alpha}, \Gamma_R$. Applying the IH to $\Gamma \vdash e_2 \Downarrow B$, with $\Gamma_L, \widehat{\alpha_1}, \widehat{\alpha_2}, \widehat{\alpha}{=}\widehat{\alpha_1}{\to}\widehat{\alpha_2}, \Gamma_R$ yields

$$\Gamma_L, \widehat{\alpha_1}, \widehat{\alpha_2}, \widehat{\alpha}{=}\widehat{\alpha_1}{\to}\widehat{\alpha_2}, \Gamma_R \vdash e_2 \Downarrow \widehat{\alpha_1} \dashv \Gamma'_2$$

Applying $\to$E$\widehat{\alpha}$ gives $\Gamma'_1 \vdash e_1 e_2 \Uparrow \widehat{\alpha_2} \dashv \Gamma'_2$.

- **Case** $\mathbf{1}$I: Since $A = \mathbf{1}$, either $A' = \mathbf{1}$ and we just apply $\mathbf{1}$I, or $A' = \widehat{\alpha}$ where $[\Omega]\widehat{\alpha} = \mathbf{1}$, and $\mathbf{1}$I, ExSubst$\Downarrow$ give the result.

- **Case** $\forall$I: $\mathcal{D} :: \dfrac{\Gamma, \alpha \vdash e \Downarrow A_0}{\Gamma \vdash e \Downarrow \forall\alpha.\, A_0}$

A' is either $\forall\alpha.\, A'_0$ or $\widehat{\beta}$. But if $A' = \widehat{\beta}$ then $[\Omega]\widehat{\beta} = \forall\alpha.\, A_0$, violating the assumption that Ω is predicative. Therefore $A' = \forall\alpha.\, A'_0$, and $[\Omega]A'_0 = A_0$.

$$\begin{array}{lrl}
& \Gamma'_1, \alpha \vdash e \Downarrow A'_0 \dashv \Gamma'_2, \alpha, \Gamma_Z & \text{By IH} \\
☞ & \Gamma'_1 \vdash e \Downarrow \forall\alpha.\, A'_0 \dashv \Gamma'_2 & \text{By } \forall\text{I}
\end{array}$$

- **Case** $\forall$E: $\mathcal{D} :: \dfrac{\Gamma \vdash e \Uparrow \forall\alpha.\, A_0 \qquad \Gamma \vdash B\ \mathsf{wf}}{\Gamma \vdash e \Uparrow [B/\alpha]A_0}$

Let $\Omega' = \mathit{Artic}(\widehat{\alpha}{=}B)$. By IH with (Ω, Ω'), we have $\Gamma'_1 \vdash e \Uparrow A' \dashv \Gamma'_2$ where $[\Omega, \Omega']A' = \forall\alpha.\, A_0$. Since Ω is predicative, A' must have the form $\forall\alpha.\, A'_0$ where $[\Omega]A'_0 = A_0$. By $\forall$E$\widehat{\alpha}$, $\Gamma'_1 \vdash e \Uparrow [\widehat{\alpha}/\alpha]A'_0 \dashv \Gamma'_2, \widehat{\alpha}$. The context Ω, Ω' includes the articulation of $\widehat{\alpha}{=}B$, so $[\Omega, \Omega']\widehat{\alpha} = B$. Then $[\Omega, \Omega'][\widehat{\alpha}/\alpha]A'_0 = [B/\alpha]A_0$. □