FPH: First-class polymorphism for Haskell
Extended Version

Dimitrios Vytiniotis

dimitriv@cis.upenn.edu

Computer and Information Science Department

University of Pennsylvania

May 2008

---

joint work with Simon Peyton Jones and Stephanie Weirich

Outline

Abstract ii

1 Introduction 1

2 Type inference for first-class polymorphism 2
- 2.1 Marking impredicative instantiation . . . 3
- 2.2 Expressive power . . . 4
- 2.3 Limitations of FPH . . . 5

3 Declarative specification of the FPH type system 5
- 3.1 Typing rules . . . 6
- 3.2 The subsumption rule . . . 7
- 3.3 Properties . . . 8
- 3.4 Higher rank types and System F . . . 9
- 3.5 Predictability and robustness . . . 11

4 An equivalent declarative specification 11

5 Syntax-directed specification 13

6 Algorithmic implementation 18
- 6.1 The basic ideas . . . 18
- 6.2 Description of the algorithm . . . 19
  - 6.2.1 Bounds, and the meaning of constraints . . . 20
  - 6.2.2 Inference implementation . . . 23
  - 6.2.3 Instance checking and unification . . . 27
  - 6.2.4 Summary of the algorithmic implementation properties . . . 27
- 6.3 Detailed algorithm metatheory . . . 28
  - 6.3.1 Unification termination . . . 29
  - 6.3.2 Unification soundness properties . . . 37
  - 6.3.3 Unification completeness lemmas . . . 46
  - 6.3.4 Main algorithm soundness . . . 48
  - 6.3.5 Main algorithm completeness . . . 51

7 Discussion 53
- 7.1 Bidirectionality . . . 54
- 7.2 η-conversion and deep instance relations . . . 55
- 7.3 Alternative design choices . . . 55

8 Related work 57

9 Future work and conclusions 60

Abstract

Languages supporting polymorphism typically have ad-hoc restrictions on where polymorphic types may occur. Supporting "first-class" polymorphism, by lifting those restrictions, is obviously desirable, but it is hard to achieve this without sacrificing type inference. We present a new type system for higher-rank and impredicative polymorphism that improves on earlier proposals: it is an extension of Damas-Milner; it relies only on System F types; it has a simple, declarative specification; it is robust to program transformations; and it enjoys a complete and decidable type inference algorithm.

1 Introduction

Consider this program fragment[1]:

```
($)   :: forall a b. (a->b) -> a -> b
runST :: forall r. (forall s. ST s r) -> r
foo   :: forall s. Int -> ST s Int

...(runST $ foo 4)...
```

Here `($)`, whose type is given, is the apply combinator, often used by Haskell programmers to avoid writing parentheses.[2] From a programmer's point of view there is nothing very complicated about this program, yet it goes well beyond the traditional Damas-Milner type system (Damas and Milner 1982), by using two distinct forms of first-class polymorphism:

- `runST` takes an argument of polymorphic type—`runST` has a *higher-rank* type.
- The quantified type variable `a` in the type of `($)` is instantiated to the polymorphic type $\forall s.\texttt{Int} \rightarrow \texttt{ST}\, s\, \texttt{Int}$. Allowing the instantiation of quantified type variables with polytypes is called *impredicative* polymorphism.

Our goal, which we share with other authors (Le Botlan and Rémy 2003; Leijen 2007a), is to make such programs "just work" by lifting the restrictions imposed by the Damas-Milner type system.

Although there are several competing designs with the same general goal, the design space is now becoming clear, so this paper is not simply "yet another impenetrable paper on impredicative polymorphism". We give a detailed comparison in Section 8, but meanwhile the distinctive feature of our system is this: rather than maximizing expressiveness or minimizing implementation complexity, we focus on programmer accessibility by *minimizing the complexity of the specification*. More specifically, we make the following contributions:

- We describe and formalize a new type system, FPH, based on System F, capable of expressing impredicative polymorphism (Section 3). We show that FPH can express all of System F (Section 3.4).
- FPH is unusually small and simple for its expressive power. It can be explained informally in a few paragraphs (Section 2), and in particular has the following delightfully simple rule for when a type annotation is required: *a type annotation may be required only for a* `let`*-binding or* λ*-abstraction that has a non-Damas-Milner type* (Section 2.2). For example, a nested function call, such as `(f (g x) (h (t y)))`, may involve lots of impredicative instantiation, but never requires a type annotation.
- We give a syntax-directed variant of the type system (Section 5), and prove it sound and complete with respect to the earlier declarative rules.
- We have a sound and complete inference algorithm for FPH, which we sketch in Section 6. Internally, this implementation uses type schemes with bounded quantification in the style of ML^F (Le Botlan and Rémy 2003), but this internal sophistication is never shown to the programmer; it is simply the mechanism used by the implementation to support the simple declarative specification.

Our system is fully compatible with the standard idea of propagating annotations via a so-called bidirectional type system. We discuss this and other design variants in Section 7.

Finally, with the scaffolding now in place, Section 8 amplifies our opening remarks by showing in detail how the various current designs relate to each other.

[1] We use Haskell syntax, and will often prefix examples with type signatures for any functions used in the fragment.
[2] The example is equivalent to `(runST (foo 4))`.

Auxiliary material accompanying this paper can be found at:

`www.cis.upenn.edu/~dimitriv/fph/`

2 Type inference for first-class polymorphism

To describe the main difficulty with first-class polymorphism, we first distinguish between Damas-Milner types (types permitting only top-level quantification) and *rich* types (types with $\forall$ quantifiers under type constructors). For example, `Int` $\rightarrow$ `Int` and $\forall a\,.\,a \rightarrow a$ are Damas-Milner types; but `Int` $\rightarrow [\forall a\,.\,a \rightarrow a]$ and $\forall a\,.(\forall b\,.\,b) \rightarrow [a]$ are rich types.

Both forms of first-class polymorphism (higher-rank and impredicative) result in a lack of principal types for expressions: a single expression may be typeable with two or more *incomparable* types, where neither is more general than the other. As a consequence, type inference cannot always choose a single type and use it throughout the scope of a `let`-bound definition.

1. With higher-rank polymorphism, functions that accept polymorphic arguments may be typed with two or more incomparable System F types. For example, consider the function `f` below:

   ```
   f get = (get 3, get True)
   ```

   It is clear that `get` must be assigned a polymorphic type in the environment, since we must be able to apply it to both `3` and `True`. But what is the exact type of `f`? For example, both $(\forall a\,.\,a \rightarrow a) \rightarrow$ (`Int`, `Bool`), and $(\forall a\,.\,a \rightarrow$ `Int`$) \rightarrow$ (`Int`, `Int`) are valid types for `f`, but there exists no *principal* type for `f` such that all others follows from it by a sequence of instantiations and generalizations. Previous work has suggested that the programmer should be required to supply a *type annotation* for any function argument that must be polymorphic, so that the type of `f` is no longer ambiguous—the above code would fail to type check, but the annotation below would fix the problem:

   ```
   f (get :: forall a. a->a) = (get 3, get True)
   ```

2. The presence of impredicative instantiation of type variables leads to a second case of incomparable types. For example:

   ```
   choose :: forall a. a -> a -> a
   id     :: forall b. b -> b

   g = choose id
   ```

   In a traditional Damas-Milner type system, `g` would get the type $\forall b\,.(b \rightarrow b) \rightarrow (b \rightarrow b)$. However, if `choose` may be instantiated with a polymorphic type, `g` is also typeable with the incomparable type $(\forall b\,.\,b \rightarrow b) \rightarrow (\forall b\,.\,b \rightarrow b)$. This problem has been identified in the ML^F work and circumvented by *extending* the type language to include instantiation constraints. This extended type language can express a principal type for `g`, namely $\forall(a \geq \forall b\,.\,b \rightarrow b)\,.\,a \rightarrow a$. However, if one wants to remain within the type language of System F, the type system must specify which of these incomparable types is assigned to `g`. In FPH, `g` is typeable with its best Damas-Milner type $\forall b\,.(b \rightarrow b) \rightarrow (b \rightarrow b)$, but the type $(\forall b\,.\,b \rightarrow b) \rightarrow (\forall b\,.\,b \rightarrow b)$ is also available by using an explicit type signature, as follows:

   ```
   g = choose id :: (forall b.b->b) -> (forall b.b->b)
   ```

The focus of this paper is on impredicativity (item (2) above), since earlier work has essentially solved the question of higher-rank types (e.g. (Peyton Jones et al. 2007)). The core type system we present in Section 3 therefore does not support λ-abstractions with higher-rank types, focusing exclusively on impredicative instantiations. A practical system must accommodate higher-rank types as well, and we describe how previous work can be adapted to our setting in Sections 3.4 and 7.1.

2.1 Marking impredicative instantiation

We present a flavor of FPH in this section, and use several examples to motivate its design principles. Consider this program fragment:

```
str :: [Char]
ids :: [forall a. a->a]
length :: forall b. [b] -> Int

l1 = length str
l2 = length ids
```

First consider type inference for `l1`. The polymorphic `length` returns the length of its argument list, where the type `[b]` means "list of `b`". In the standard Damas-Milner type system, one instantiates the type of `length` with `Char`, so that the occurrence of `length` has type `[Char]` $\rightarrow$ `Int`, which marries up correctly with `length`'s argument, `str`. In Damas-Milner, a polymorphic function can only be instantiated with monotypes, where a monotype τ is a type containing no quantification:

$$\tau ::= a \mid \tau_1 \rightarrow \tau_2 \mid \mathsf{T}\,\tau$$

This Damas-Milner restriction means that `l2` is untypeable, because here we must instantiate `length` with $\forall a.a \rightarrow a$. We cannot simply lift the Damas-Milner restriction, because that directly leads to the problem identified at the start of this section: different choices can lead to incomparable types. However, `l2` also shows that there are benign uses of impredicative instantiation. Although we need an impredicative instantiation to make `l2` type check, there is no danger here—the type of `l2` will always be `Int`. It is only when a `let`-binding can be assigned two ore more incomparable types that we run into trouble.

Our idea is to mark impredicative instantiations so that we know when an expression may be typed with different incomparable types. Technically, this means that we instantiate polymorphic functions with a form of type τ' that is more expressive than a mere monotype, but less expressive than an arbitrary polymorphic type:

$$\begin{array}{lcl} \tau' & ::= & a \mid \tau'_1 \rightarrow \tau'_2 \mid \mathsf{T}\,\tau' \mid \boxed{\sigma} \\ \sigma & ::= & \forall a.\sigma \mid a \mid \sigma \rightarrow \sigma \mid \mathsf{T}\,\sigma \end{array}$$

Unlike a monotype τ, a *boxy monotype* τ' may contain quantification, but only inside a box, thus $\boxed{\sigma}$. **Idea 1** is this: a polymorphic function is instantiated with boxy monotypes. A boxy type marks the place in the type where "guessing" is required to fill in a type that makes the rest of the typing derivation go through.

Now, when typing `l2` we may instantiate `length` with $\boxed{\forall a.a \rightarrow a}$. Then the application `length ids` has a function expecting an argument of type `[`$\boxed{\forall a.a \rightarrow a}$`]`, applied to an argument of type `[`$\forall a.a \rightarrow a$`]`. Do these types marry up? Yes, they do, because of **Idea 2**: when comparing types, discard all boxes. The sole purpose of boxes is to mark polytypes that arise from impredicative instantiations. That completes the typing of `l2`.

Boxes are ignored when typing an application, but they play a critical role in `let` polymorphism. **Idea 3** is this: to make sure that there is no ambiguity about guessed polytypes, the type environment contains no boxes. Let us return to the example `g = choose id` given above. If we instantiated `choose` with the boxy monotype $\boxed{\forall a.a \rightarrow a}$, the application `(choose id)` would marry up fine, but its result type would be $\boxed{\forall a.a \rightarrow a} \rightarrow \boxed{\forall a.a \rightarrow a}$. However, **Idea 3** prevents that type from entering the environment as the type for `g`, so this instantiation for `choose` is rejected. If we instead instantiate `choose` with $c \rightarrow c$, the application again marries up (this time by instantiating the type of `id` with c), so the application has type $(c \rightarrow c) \rightarrow c \rightarrow c$, which can be generalized and then enter the environment as the type of `g`. This type is the principal Damas-Milner type of `g`—all Damas-Milner types for `g` are also available without annotation. What we have achieved effectively is that, instead of having two or more incomparable types for `g`, we have allowed only those typing derivations for `g` that admit a principal type.

However, if the programmer actually wanted the other, rich, type for `g`, she can use a type annotation:

```
g = choose id :: (forall b.b->b) -> (forall b.b->b)
```

Such type annotations use **Idea 2**—when typing an annotated expression e`::`σ, ignore boxes on e's type when comparing with σ (which is box-free, being a programmer annotation). Now we may instantiate `choose` with $\boxed{\forall a.a \to a}$, because the type annotation is compatible with the type of `(choose id)`, $\boxed{\forall a.a \to a} \to \boxed{\forall a.a \to a}$.

2.2 Expressive power

As we have seen, a type annotation may be required on a `let`-bound expression, but annotations are never required on function applications, even when they are nested and higher order, or involve impredicativity. Here is the example from the Introduction, with some variants:

```
runST  :: forall a. (forall s. ST s a) -> a
app    :: forall a b. (a -> b) -> a -> b
revapp :: forall a b. a -> (a -> b) -> b
arg    :: forall s. ST s Int

h0 = runST arg
h1 = app runST arg
h2 = revapp arg runST
```

All definitions `h0`, `h1`, `h2` are typeable without annotation because, in each case, the return type is a (non-boxy) monotype `Int`.

Actually, we have a much more powerful guideline for programmers, which does not even require them to think about boxes:

> **Annotation Guideline.** Write your programs as you like, without type annotations at all. Then you are required to annotate only those `let`-bindings and λ-abstractions that you want to be typed with rich types.

For instance, for a term consisting of applications and variables to be `let`-bound (without any type annotations), it *does not matter* what impredicative instantiations may happen to type it, provided that the result type is an ordinary Damas-Milner type! For example, the argument `choose id` to the function `f` below involves an impredicative instantiation (in fact for both `f` *and* `choose`), but no annotation is required whatsoever:

```
f :: forall a. (a -> a) -> [a] -> a
g = f (choose id) ids
```

In particular `choose id` gets type $\boxed{\forall a.a \to a} \to \boxed{\forall a.a \to a}$. However, `f`'s arguments types can be married up using **Idea 2**, and its result type (ignoring boxes) is a Damas-Milner type ($\boxed{\forall a\,.\,a \to a}$), and hence no annotation is required for `g`.

Since the Annotation Guideline does not require the programmer to think about boxes at all, why does our specification use boxes? Because the Annotation Guideline is conservative: it guarantees to make the program typeable, but it adds more annotations than are necessary. For example:

```
f' :: forall a. [a] -> [forall b. b -> b]
g' = f' ids
```

Notice that the rich result type `[forall b. b -> b]` is non-boxy, and hence no annotation is required for `g'`. In general, even if the type of a `let`-bound expression is rich, if that type does not result from impredicative instantiation (which is the common case), then no annotations are required. Boxes precisely specify what "that type does not result from impredicative instantiation" means. Nevertheless, a box-free specification is an attractive alternative design, as we discuss in Section 7.3.

$$
\begin{array}{lllll}
\text{Types} & \sigma & ::= & \forall \overline{a}\,.\,\rho & \\
 & \rho & ::= & \tau \mid \sigma \to \sigma \mid \mathsf{T}\ \overline{\sigma} & \\
 & \tau & ::= & a \mid \tau \to \tau \mid \mathsf{T}\ \overline{\tau} & \\
 & & & & \\
\text{Boxy Types} & \sigma' & ::= & \forall \overline{a}\,.\,\rho' & \\
 & \rho' & ::= & \tau' \mid \sigma' \to \sigma' \mid \mathsf{T}\ \overline{\sigma}' & \\
 & \tau' & ::= & a \mid \boxed{\sigma} \mid \tau' \to \tau' \mid \mathsf{T}\ \overline{\tau}' & \\
 & & & & \\
\text{Environments} & \Gamma & ::= & \Gamma, (x{:}\sigma) \mid \cdot &
\end{array}
$$

Figure 1: Syntax

2.3 Limitations of FPH

Although the FPH system, as we have described it so far, is expressive, it is also somewhat conservative. It requires annotations in a few instances, even when there is only one type that can be assigned to a `let`-binding, as the following example demonstrates.

```
f :: forall a. a -> [a] -> [a]
ids :: [forall a. a -> a]

h1 = f (\x -> x) ids            -- Not typeable
h2 = f (\x -> x) ids :: [forall a. a->a]  -- OK
```

Here `f` is a function that accepts an element and a list and returns a list (for example, `f` could be `cons`). Definition `h1` is not typeable in FPH. We can attempt to instantiate `f` with $\boxed{\forall a.a \to a}$, but then the right hand side of `h1` has type `[`$\boxed{\forall a.a \to a}$`]`, and that type cannot enter the environment. The problem can of course be fixed by adding a type annotation, as `h2` shows.

You may think that it is silly to require a type annotation in `h2`; after all, `h1` manifestly has only one possible type! But suppose that `f` had type $\forall ab.a \to b \to [a]$, which is a more general Damas-Milner type than the type above. With this type for `f`, our example `h1` now has *two incomparable types*, namely $[\forall a.a \to a]$ as before, and $\forall a.[a \to a]$. Without any annotations we presumably have to choose the same type as the Damas-Milner type system would; and that might make occurrences of `h1` ill typed. In short, making the type of `f` more general has caused definitions in the scope of `h1` to become ill-typed! This is bad; and that is the reason that we reject `h1`, requiring an annotation as in `h2`.

Requiring an annotation on `h2` may seem an annoyance to programmers, but it is this conservativity of FPH that results in a simple and declarative high-level specification. FPH allows `let`-bound definitions to enter environments with many different types, as is the case in the Damas-Milner type system.

3 Declarative specification of the FPH type system

We now turn our attention to a systematic treatment of FPH, beginning with the basic syntax of types and environments in Figure 1. Types are divided into box-free types σ-, ρ-, and τ-types, and boxy types σ', ρ', and τ' types. Polymorphic types, σ and σ', may contain quantifiers at top-level, whereas ρ and ρ' types contain only nested quantifiers. The important difference between box-free and boxy types occurs at the monotype level. Following previous work by Rémy *et al.* (Garrigue and Rémy 1999; Le Botlan and Rémy 2003), τ' may include boxes containing (box-free) polytypes. As we discussed in Section 2.1, these boxes represent the places where "guessed instantiations" take place. The syntax of type environments, Γ, directly expresses **Idea 3** in Section 2.1 by allowing only box-free types σ.

$$\boxed{\Gamma \vdash \mathsf{e} : \sigma'}$$

$$\frac{(x{:}\sigma) \in \Gamma}{\Gamma \vdash x : \sigma}\ \text{VAR} \qquad \frac{\Gamma \vdash e_1 : \sigma_1' \to \sigma_2' \quad \Gamma \vdash e_2 : \sigma_3' \quad \lfloor \sigma_3' \rfloor = \lfloor \sigma_1' \rfloor}{\Gamma \vdash e_1\ e_2 : \sigma_2'}\ \text{APP} \qquad \frac{\Gamma, (x{:}\tau) \vdash e : \rho}{\Gamma \vdash \lambda x\,.\,e : \tau \to \rho}\ \text{ABS}$$

$$\frac{\Gamma \vdash u : \sigma \quad \Gamma, (x{:}\sigma) \vdash e : \rho'}{\Gamma \vdash \mathtt{let}\ x = u\ \mathtt{in}\ e : \rho'}\ \text{LET} \qquad \frac{\Gamma \vdash e : \sigma_1' \quad \lfloor \sigma_1' \rfloor = \sigma}{\Gamma \vdash (e\,\mathtt{::}\sigma) : \sigma}\ \text{ANN}$$

$$\frac{\Gamma \vdash e : \forall \overline{a}\,.\,\rho'}{\Gamma \vdash e : [\overline{a \mapsto \tau'}]\rho'}\ \text{INST} \qquad \frac{\Gamma \vdash e : \rho' \quad \overline{a} \# \Gamma}{\Gamma \vdash e : \forall \overline{a}\,.\,\rho'}\ \text{GEN} \qquad \frac{\Gamma \vdash e : \rho_1' \quad \rho_1' \preceq \sqsubseteq \rho_2'}{\Gamma \vdash e : \rho_2'}\ \text{SUBS}$$

Figure 2: The FPH system

3.1 Typing rules

The declarative (i.e. not syntax-directed) specification of FPH is given in Figure 2. As usual, the judgement form $\Gamma \vdash e : \sigma'$ assigns the type σ' to the expression e in typing environment Γ. A non-syntactic invariant of the typing relation is that, in the judgement $\Gamma \vdash e : \forall \overline{a}.\rho'$, no box may intervene between a variable quantified inside ρ' and the occurrences of that variable. Thus, for example, ρ' cannot be of form $(\forall b.\boxed{b}) \to \mathtt{Int}$, because the quantified variable b appears inside a box. The top-level quantified variables may, however, appear inside boxes.

The rules in Figure 2 are modest (albeit carefully-chosen) variants of the conventional Damas-Milner rules. Indeed rule VAR is precisely as usual, simply returning the type of a variable from the environment.

Rule APP infers a function type $\sigma_1' \to \sigma_2'$ for e_1, infers a type σ_3' for the argument e_2, and checks that the argument type matches the domain of the function type *modulo boxy structure*, implementing **Idea 2** of Section 2.1. This compatibility check is performed by *stripping* the boxes from σ_1' and σ_3', then comparing for equality. The notation $\lfloor \sigma' \rfloor$ denotes the non-boxy type obtained by discarding the boxes in σ':

Definition 3.1 (Stripping)**.** We define the strip function $\lfloor \cdot \rfloor$ on boxy types as follows:

$$\begin{array}{lcll} \lfloor a \rfloor & = & a & \\ \lfloor \boxed{\sigma} \rfloor & = & \sigma & \\ \lfloor \sigma_1' \to \sigma_2' \rfloor & = & \lfloor \sigma_1' \rfloor \to \lfloor \sigma_2' \rfloor & \\ \lfloor \forall \overline{a}\,.\,\rho' \rfloor & = & \forall \overline{a}\,\overline{b}\,.\,\rho & \text{where } \lfloor \rho' \rfloor = \forall \overline{b}\,.\,\rho \end{array}$$

Equality between types is ordinary α-equivalence. Stripping is also used in rule ANN, which handles expressions with explicit programmer-supplied type annotations. It infers a boxy type for the expression and checks that, modulo its boxy structure, it is equal to the type required by the annotation σ. In effect, this rule converts the boxy type σ_1' that was inferred for the expression to a box-free type σ. If the annotated term is the right-hand side of a `let` binding `x = e::`σ, this box-free type σ can now enter the environment as the type of `x` (whereas σ' could not, by **Idea 3**).

Rule ABS infers types for λ-abstractions. It first extends the environment with a *monomorphic, box-free* typing $x : \tau$, and infers a ρ-type for the body of the function. Notice that we insist (syntactically) that the result type ρ both (a) has no top-level quantifiers, and (b) is box-free. We exclude top-level quantifiers (a) because we wish to attribute the same types as Damas-Milner for programs that are typeable by Damas-Milner; in the vocabulary of (Peyton Jones et al. 2007), we avoid "eager generalization". Choice (b), that a λ-abstraction must return a box-free type, may require more programmer annotations, but turns out to permit a much simpler type inference algorithm. We return to this issue in Section 7.3.

Rule ABS is the main reason that the type system of Figure 2 cannot type all of System F, even with the addition of type annotations: ABS allows only abstractions of type $\tau \to \rho$, whereas System F has λ-abstractions of type $\sigma_1 \to \sigma_2$. Rule ABS is however just enough to demonstrate our approach to impredicative instantiation (the contribution of this paper), while previous work (Peyton Jones et al. 2007) has shown how to address this limitation. It is easy to combine the two, as we show in Section 3.4.

Following **Idea 3** of Section 2.1, rule LET first infers a *box-free* type σ for the right-hand side expression u, and then checks the body pushing the binder x with type σ in the environment.

Generalization (GEN) takes the conventional form, where $\overline{a}\#\Gamma$ means that $\overline{a}$ is disjoint from the free type variables of Γ. In this rule, note that the generalized variables $\overline{a}$ may appear inside boxes in ρ', so that we might, for example, infer $\Gamma \vdash e : \forall a.\boxed{a} \to a$.

Instantiation (INST) is largely conventional, but it follows **Idea 1** by allowing us to instantiate a type with a *boxy* monotype τ'. However, we need to be a little careful with substitution in INST: since ρ' may contain $\overline{a}$ inside boxes, a naive substitution might leave us with nested boxes, which are syntactically ill-formed. Hence, we define a form of substitution that preserves the boxy structure of its argument.

Definition 3.2 (Monomorphic substitutions)**.** We use letter φ for *monomorphic substitutions*, that is, φ denotes finite maps of the form $[\overline{a \mapsto \tau'}]$. We let $ftv(\varphi)$ be the set of the free variables in the range and domain of φ. We define the operation of applying φ to a type σ' as follows:

$$\begin{array}{lcll} \varphi(a) & = & \tau' & \text{where } [a \mapsto \tau'] \in \varphi \\ \varphi(\boxed{\sigma}) & = & \boxed{\lfloor \varphi(\sigma) \rfloor} & \\ \varphi(\sigma'_1 \to \sigma'_2) & = & \varphi(\sigma'_1) \to \varphi(\sigma'_2) & \\ \varphi(\forall \overline{a}\,.\,\rho') & = & \forall \overline{a}\,.\,\varphi(\rho') & \text{where } \overline{a}\#ftv(\varphi) \end{array}$$

We write $[\overline{a \mapsto \tau'}]\sigma'$ for the application of the $[\overline{a \mapsto \tau'}]$ to σ'.

3.2 The subsumption rule

The final rule, SUBS, is tricky but important. Consider the code fragment in Example 3.3.

Example 3.3 (Boxy instantiation)**.**

```
head :: forall a. [a] -> a
h = head ids 3
```

Temporarily ignoring rule SUBS in Figure 2, the application `head ids` can get type $\boxed{\forall a\,.\,a \to a}$, and only that type. Hence, the application `(head ids) 3` cannot be typed. This situation would be rather unfortunate as one would, in general, have to use type annotations to extract polymorphic expressions out of polymorphic data structures. For example, programmers would have to write:

```
h = (head ids :: forall b. b -> b) 3
```

This situation would also imply that some expressions which consist only of applications of closed terms, and are typeable in System F, could not be typed in FPH.

Rule SUBS addresses these limitations. Rule SUBS modifies the types of expression in two ways with the relation $\preceq\!\!\sqsubseteq$, which is the composition of two relations, $\preceq$, and $\sqsubseteq$. The relation $\preceq$, called *boxy instantiation*, simply instantiates a polymorphic type within a box. The relation $\sqsubseteq$, called *protected unboxing*, removes boxes around monomorphic types and pushes boxes congruently down the structure of types. The most important rules of this relation are TBOX and REFL. The first simply removes a box around a monomorphic type, while the second ensures reflexivity. If a ρ' type contains only boxes with monomorphic information, then these boxes can be completely dropped along the $\sqsubseteq$ relation to yield a box-free type.

$$\boxed{\sigma_1' \sqsubseteq \sigma_2'}$$

$$\frac{}{\boxed{\tau} \sqsubseteq \tau}\ \text{TBOX} \qquad \frac{}{\sigma' \sqsubseteq \sigma'}\ \text{REFL} \qquad \frac{\sigma_1' \sqsubseteq \sigma_1'' \quad \sigma_2' \sqsubseteq \sigma_2''}{\sigma_1' \to \sigma_2' \sqsubseteq \sigma_1'' \to \sigma_2''}\ \text{CONG}$$

$$\frac{\rho' \sqsubseteq \rho'' \quad \overline{a}\ \text{unboxed in}\ \rho', \rho''}{\forall \overline{a}\,.\,\rho' \sqsubseteq \forall \overline{a}\,.\,\rho''}\ \text{POLY} \qquad \frac{\boxed{\sigma_1} \sqsubseteq \sigma_1' \quad \boxed{\sigma_2} \sqsubseteq \sigma_2'}{\boxed{\sigma_1 \to \sigma_2} \sqsubseteq \sigma_1' \to \sigma_2'}\ \text{CONBOX}$$

$$\boxed{\sigma_1' \preceq \sigma_2}$$

$$\frac{}{\boxed{\forall \overline{a}\,.\,\rho} \preceq \boxed{[\overline{a} \mapsto \overline{\sigma}]\rho}}\ \text{BI} \qquad \frac{}{\sigma' \preceq \sigma'}\ \text{BR}$$

Figure 3: Protected unboxing and boxy instantiation relation

Because SUBS uses $\preceq\sqsubseteq$ instead of merely $\sqsubseteq$, `h` in Example 3.3 is typeable. When we infer a type for `head ids`, we may have the following derivation:

$$\frac{\dfrac{\Gamma \vdash \texttt{head ids} : \boxed{\forall a\,.\,a \to a} \quad \boxed{\forall a\,.\,a \to a} \preceq \boxed{a \to a} \sqsubseteq a \to a}{\Gamma \vdash \texttt{head ids} : a \to a}\ \text{SUBS}}{\Gamma \vdash \texttt{head ids} : \forall a\,.\,a \to a}\ \text{GEN}$$

Therefore, no annotation is required on `h`. Incidentally, because the $\sqsubseteq$ relation can remove boxes around monomorphic types, it also follows that

```
h = head ids
```

is typeable. More generally, we have the following lemma.

Lemma 3.4. *If* $\Gamma \vdash e : \boxed{\forall \overline{a}\,.\,\tau}$ *then* $\Gamma \vdash e : \forall \overline{a}\,.\,\tau$.

Proof. By rule BI we have $\boxed{\forall \overline{a}\,.\,\tau} \preceq \boxed{\tau}$ where without loss of generality $\overline{a}\#\Gamma$, and by rule TBOX we get $\boxed{\tau} \sqsubseteq \tau$, hence $\boxed{\forall \overline{a}\,.\,\tau} \preceq\sqsubseteq \tau$. Applying rule SUBS to the derivation of $\Gamma \vdash e : \boxed{\forall \overline{a}\,.\,\tau}$ gives $\Gamma \vdash e : \tau$, and by rule GEN we get $\Gamma \vdash e : \forall \overline{a}\,.\,\tau$, as required. □

3.3 Properties

The FPH system is type safe with respect to the semantics of System F. The following lemma is an easy induction after observing that whenever $\sigma_1' \preceq\sqsubseteq \sigma_2'$, it is the case that $\vdash^{\mathsf{F}} \lfloor \sigma_1' \rfloor \leq \lfloor \sigma_2' \rfloor$, where $\vdash^{\mathsf{F}}$ is the System F *type instance relation*. The relation $\vdash^{\mathsf{F}}$ specifies typeability of an expression of one type with another type through a series of instantiations and generalizations, and is given by the rule below:

$$\frac{\overline{b}\#ftv(\forall \overline{a}\,.\,\rho)}{\vdash^{\mathsf{F}} \forall \overline{a}\,.\,\rho \leq \forall \overline{b}\,.\,[\overline{a} \mapsto \overline{\sigma}]\rho}\ \text{FSUBS}$$

The (implicitly typed) System F typing relation is given in Figure We additionally define a function

$$\boxed{\Gamma \vdash^{\mathsf{F}} \mathsf{e_F} : \sigma \rightsquigarrow \mathsf{e}}$$

$$\frac{(x{:}\sigma) \in \Gamma}{\Gamma \vdash^{\mathsf{F}} x : \sigma}\ \text{FVAR} \qquad \frac{\Gamma \vdash^{\mathsf{F}} e_1 : \sigma_1 \to \sigma_2 \quad \Gamma \vdash^{\mathsf{F}} e_2 : \sigma_1}{\Gamma \vdash^{\mathsf{F}} e_1\ e_2 : \sigma_2}\ \text{FAPP}$$

$$\frac{\Gamma \vdash^{\mathsf{F}} e : \forall \overline{a}\,.\,\rho}{\Gamma \vdash^{\mathsf{F}} e : [\overline{a \mapsto \sigma}]\rho}\ \text{FINST} \qquad \frac{\Gamma \vdash^{\mathsf{F}} e : \rho \quad \overline{a} \# \Gamma}{\Gamma \vdash^{\mathsf{F}} e : \forall \overline{a}\,.\,\rho}\ \text{FGEN}$$

$$\frac{\Gamma, (x{:}\sigma_1) \vdash^{\mathsf{F}} e : \sigma_2}{\Gamma \vdash^{\mathsf{F}} \lambda x\,.\,e : \sigma_1 \to \sigma_2}\ \text{FABS} \qquad \frac{\Gamma \vdash^{\mathsf{F}} u : \sigma_1 \quad \Gamma, (x{:}\sigma_1) \vdash^{\mathsf{F}} e : \sigma}{\Gamma \vdash^{\mathsf{F}} \mathtt{let}\ x = u\ \mathtt{in}\ e : \sigma}\ \text{FLET}$$

Figure 4: Implicitly typed System F (with local definitions)

$(\cdot)^\flat$ from terms of FPH, that removes any annotations from terms. Its definition follows:

$$\begin{array}{lcl} x^\flat & = & x \\ (e :: \sigma)^\flat & = & e^\flat \\ (\lambda x\,.\,e)^\flat & = & \lambda x\,.\,e^\flat \\ (e_1\ e_2)^\flat & = & e_1^\flat\ e_2^\flat \\ (\mathtt{let}\ x = u\ \mathtt{in}\ e)^\flat & = & \mathtt{let}\ x = u^\flat\ \mathtt{in}\ e^\flat \end{array}$$

Lemma 3.5. *If* $\Gamma \vdash e : \sigma'$ *then* $\Gamma \vdash^{\mathsf{F}} e^\flat : \lfloor \sigma' \rfloor$.

Proof. Straightforward induction. □

Moreover, FPH is an extension of the Damas-Milner type system. The idea of the following lemma is that instantiation to τ' types always subsumes instantiation to τ types.

Lemma 3.6 (Extension of Damas-Milner)**.** *Assume that* Γ *contains only types with top-level quantifiers and* e *is annotation-free. Then* $\Gamma \vdash^{\mathsf{DM}} e : \sigma$ *implies that* $\Gamma \vdash e : \sigma$.

Proof. Straightforward induction. □

We conjecture that the converse direction is also true, that is, unannotated programs in contexts that use only Damas-Milner types are typeable in Damas-Milner if they are typeable in FPH, but we leave this result as future work.

3.4 Higher rank types and System F

As we remarked in the discussion of rule ABS in Section 3.1, the system described so far deliberately does not support λ-abstractions with higher-rank types, and hence cannot yet express all of System F. For example:

Example 3.7.

```
f   :: forall a. a -> [a] -> Int
foo :: [Int -> forall b.b->b]

bog = f (\x y ->y) foo
```

Here, `foo` requires the λ-abstraction `\x y -> y` to be typed with type `Int` $\to \forall b\,.\,b \to b$, but no such type can be inferred for the λ-abstraction, as it is not of the form $\tau \to \rho$. We may resolve this issue by adding a new syntactic form, the annotated λ-abstraction, thus $(\lambda x\,.\,e :: \sigma_1 \to \sigma_2)$. This

$$\boxed{\Gamma \vdash^{\mathsf{F}} \mathsf{e_F} : \sigma \rightsquigarrow \mathsf{e}}$$

$$\frac{(x{:}\sigma) \in \Gamma}{\Gamma \vdash^{\mathsf{F}} x : \sigma \rightsquigarrow x}\ \text{FVAR} \qquad \frac{\Gamma \vdash^{\mathsf{F}} e_1 : \sigma_1 \to \sigma_2 \rightsquigarrow e_3 \quad \Gamma \vdash^{\mathsf{F}} e_2 : \sigma_1 \rightsquigarrow e_4}{\Gamma \vdash^{\mathsf{F}} e_1\ e_2 : \sigma_2 \rightsquigarrow e_3\ e_4}\ \text{FAPP}$$

$$\frac{\Gamma \vdash^{\mathsf{F}} e : \forall \overline{a}\,.\,\rho \rightsquigarrow e_1}{\Gamma \vdash^{\mathsf{F}} e : [\overline{a \mapsto \sigma}]\rho \rightsquigarrow e_1}\ \text{FINST} \qquad \frac{\Gamma \vdash^{\mathsf{F}} e : \rho \rightsquigarrow e_1 \quad \overline{a}\#\Gamma}{\Gamma \vdash^{\mathsf{F}} e : \forall \overline{a}\,.\,\rho \rightsquigarrow e_1}\ \text{FGEN}$$

$$\frac{\Gamma, (x{:}\tau_1) \vdash^{\mathsf{F}} e : \tau_2 \rightsquigarrow e_1}{\Gamma \vdash^{\mathsf{F}} \lambda x\,.\,e : \tau_1 \to \tau_2 \rightsquigarrow \lambda x\,.\,e_1}\ \text{FABS0} \qquad \frac{\Gamma, (x{:}\sigma_1) \vdash^{\mathsf{F}} e : \sigma_2 \rightsquigarrow e_1}{\Gamma \vdash^{\mathsf{F}} \lambda x\,.\,e : \sigma_1 \to \sigma_2 \rightsquigarrow (\lambda x\,.\,e_1 ::: \sigma_1 \to \sigma_2)}\ \text{FABS1}$$

Figure 5: Type-directed translation of System F

construct provides an annotation for both argument (σ_1, instead of a monotype τ) and result (σ_2 instead of ρ). Its typing rule is simple:

$$\frac{\Gamma, (x{:}\sigma_1) \vdash e : \sigma_2' \quad \lfloor \sigma_2' \rfloor = \sigma_2}{\Gamma \vdash (\lambda x\,.\,e ::: \sigma_1 \to \sigma_2) : \sigma_1 \to \sigma_2}\ \text{ABS-ANN}$$

With this extra construct we can translate any implicitly-typed System F term into a well-typed term in FPH, using the translation of Figure 5. This type-directed translation of implicitly typed System F is specified as a judgement $\Gamma \vdash^{\mathsf{F}} e_{\mathsf{F}} : \sigma \rightsquigarrow e$ where e is a term that type checks in our language. Notice that the translation requires annotations *only* on λ-abstractions that involve rich types[3].

A subtle point is that the translation may generate *open* type annotations. For example, consider the implicitly typed System F below:

$$\vdash \lambda x\,.\,e : \forall a\,.\,(\forall b\,.\,b \to a) \to a$$

Translating this term using Figure 5 gives

$$\vdash (\lambda x\,.\,e ::: (\forall b\,.\,b \to a) \to a)$$

Note that the type annotation mentions a which is nowhere bound. Although we have not emphasized this point, FPH already accommodates such annotations.

The following theorem captures the essence of the translation.

Theorem 3.8. *If* $\Gamma \vdash^{\mathsf{F}} e : \sigma \rightsquigarrow e_1$ *then* $\Gamma \vdash e_1 : \sigma'$ *for some* σ' *such that* $\lfloor \sigma' \rfloor = \sigma$.

Proof. The proof is by induction on the derivation of $\Gamma \vdash^{\mathsf{F}} e : \sigma \rightsquigarrow e_1$. The case for FVAR is trivial. For FAPP we have that $\Gamma \vdash^{\mathsf{F}} e_1\ e_2 : \sigma_2 \rightsquigarrow e_3\ e_4$ given that $\Gamma \vdash^{\mathsf{F}} e_1 : \sigma_1 \to \sigma_2 \rightsquigarrow e_3$ and $\Gamma \vdash^{\mathsf{F}} e_2 : \sigma_1 \rightsquigarrow e_4$. By induction we have that $\Gamma \vdash e_1 : \sigma_0'$ such that $\lfloor \sigma_1' \rfloor = \sigma_1 \to \sigma_2$. This implies that $\sigma_0' \sqsubseteq \sigma_1' \to \sigma_2'$ such that $\lfloor \sigma_1' \rfloor = \sigma_1$ and $\lfloor \sigma_2' \rfloor = \sigma_2$. Hence by rule SUBS and reflexivity of $\preceq$ we get that $\Gamma \vdash e_1 : \sigma_1' \to \sigma_2'$. Also by induction we get that $\Gamma \vdash e_2 : \sigma_1''$ such that $\lfloor \sigma_1'' \rfloor = \sigma_1$. By applying rule APP we get that $\Gamma \vdash e_1\ e_2 : \sigma_2'$ for which we know that $\lfloor \sigma_2' \rfloor = \sigma_2$, as required. For rule FINST we have that $\Gamma \vdash^{\mathsf{F}} e : [\overline{a \mapsto \sigma}]\rho \rightsquigarrow e_1$ where $\Gamma \vdash^{\mathsf{F}} e : \forall \overline{a}\,.\,\rho \rightsquigarrow e_1$. By induction $\Gamma \vdash e : \sigma'$ such that $\lfloor \sigma' \rfloor = \forall \overline{a}\,.\,\rho$. Hence, by a sequence of INST and perhaps SUBS with BI we get the result. For rule FGEN the result follows by induction hypothesis and rule GEN. For rule FABS0 we have that $\Gamma \vdash^{\mathsf{F}} \lambda x\,.\,e : \tau_1 \to \tau_2 \rightsquigarrow \lambda x\,.\,e_1$, given that $\Gamma, (x{:}\tau_1) \vdash^{\mathsf{F}} e : \tau_2 \rightsquigarrow e_1$. By induction $\Gamma, (x{:}\tau_1) \vdash e_1 : \sigma_2'$

[3] Of course, it would be fine to annotate *every* λ-abstraction, but the translation we give generates smaller terms.

such that $\lfloor \sigma_2' \rfloor = \tau_2$. Because τ_2 is monomorphic it must be that $\sigma_2' \preceq_{\sqsubseteq} \tau_2$. Consequently, by rule SUBS, $\Gamma, (x{:}\tau_1) \vdash e_1 : \tau_2$, and by rule ABS $\Gamma \vdash \lambda x\,.\,e_1 : \tau_1 \to \tau_2$ as required. The case for rule FABS1 follows from induction hypothesis and rule ABS-ANN. □

In practice, however, we do not recommend adding annotated λ-abstractions as a clunky new syntactic construct. Instead, with a bidirectional typing system we can get the same benefits (and more besides) from ordinary type annotations $e\,\texttt{::}\sigma$, as we sketch in Section 7.1.

3.5 Predictability and robustness

A key feature of FPH is that it is simple for the programmer to figure out when a type annotation is required. We gave some intuitions in Section 2, but now we are in a position to give some specific results. The translation of System F to FPH of Section 3.4 shows that one needs only annotate `let`-bindings or λ-abstractions that must be typed with rich types. This is a result of combining Theorem 3.8 and Lemma 3.4.

For example, every applicative expression—one consisting only of variables, constants, and applications—that is typeable in System F is typeable in FPH without annotations. We began this paper with exactly such an example, involving `runST`, and it would work equally well if we had used reverse application instead of `$`.

Theorem 3.9. *If e is an applicative expression and $\Gamma \vdash^{\mathsf{F}} e : \sigma$, then $\Gamma \vdash e : \sigma'$ for some σ' with $\lfloor \sigma' \rfloor = \sigma$.*

Proof. This result follows from Theorem 3.8 by observing that whenever e is applicative and $\Gamma \vdash^{\mathsf{F}} e : \sigma \rightsquigarrow e_1$ then $e_1 = e$, that is, the translation never adds any annotations to applicative terms. □

Additionally, a `let`-binding can always be inlined at its occurrence sites. More precisely if $\Gamma \vdash$ `let` $x = u$ `in` $e : \sigma'$, then $\Gamma \vdash [x \mapsto u]e : \sigma'$. This follows from the following lemma:

Lemma 3.10. *If $\Gamma \vdash u : \sigma$ and $\Gamma, (x{:}\sigma) \vdash e : \sigma'$ then $\Gamma \vdash [x \mapsto u]e : \sigma'$.*

Proof. Straightforward induction. □

The converse direction cannot be true in general (although it is true for ML and ML^{F}) because of the limited expressive power of System F types, as we discussed briefly in Section 2. Let $\sigma_1 = (\forall b\,.\,b \to b) \to (\forall b\,.\,b \to b)$, $\sigma_2 = \forall b\,.\,(b \to b) \to b \to b$, $f_1 : \sigma_1 \to$ `Int`, and $f_2 : \sigma_2 \to$ `Int`. One can imagine a program of the form:

$$\ldots (f_1\ (\texttt{choose id})) \ldots (f_2\ (\texttt{choose id})) \ldots$$

which may be typeable, but it cannot be the case that: `let` $x =$ `choose id in` $\ldots (f_1\ x) \ldots (f_2\ x) \ldots$ is typeable, as x can be bound with only one of the two incomparable types (in fact only with $\forall b\,.\,(b \to b) \to b \to b$).

However, notice that if an expression is typed with a box-free type at each of its occurrences in a context, it may be `let`-bound out of the context. For example, since λ-abstractions are typed with box-free types, if $\mathcal{C}[\lambda x\,.\,e]$ is typeable, where $\mathcal{C}$ is a multi-hole context, then it is always the case that `let` $f = (\lambda x\,.\,e)$ `in` $\mathcal{C}[f]$ is typeable.

4 An equivalent declarative specification

It will be convenient to introduce at this point a slight variation of the basic type system of Figure 2 where we have pushed some instantiations and generalizations in the applications and type annotation nodes. The modified system is given in Figure 6, with the relation $\Gamma \vdash^{\mathsf{int}} e : \sigma'$ ($\vdash^{\mathsf{int}}$ for *intermediate*)

$$\boxed{\Gamma \vdash^{\mathsf{int}} \mathsf{e} : \sigma'}$$

$$\frac{(x{:}\sigma) \in \Gamma}{\Gamma \vdash^{\mathsf{int}} x : \sigma}\ \text{VAR} \qquad \frac{\Gamma \vdash^{\mathsf{int}} e_1 : \sigma_1' \to \sigma_2' \quad \Gamma \vdash^{\mathsf{int}} e_2 : \sigma_3' \quad \vdash^{\mathsf{F}} \lfloor \sigma_3' \rfloor \leq \lfloor \sigma_1' \rfloor}{\Gamma \vdash^{\mathsf{int}} e_1\ e_2 : \sigma_2'}\ \text{APP} \qquad \frac{\Gamma, (x{:}\tau) \vdash^{\mathsf{int}} e : \rho}{\Gamma \vdash^{\mathsf{int}} \lambda x\,.\,e : \tau \to \rho}\ \text{ABS}$$

$$\frac{\Gamma \vdash^{\mathsf{int}} u : \sigma \quad \Gamma, (x{:}\sigma) \vdash^{\mathsf{int}} e : \rho'}{\Gamma \vdash^{\mathsf{int}} \mathtt{let}\ x = u\ \mathtt{in}\ e : \rho'}\ \text{LET} \qquad \frac{\Gamma \vdash^{\mathsf{int}} e : \rho' \quad \rho'(\preceq\sqsubseteq)\rho''}{\Gamma \vdash^{\mathsf{int}} e : \rho''}\ \text{SUBS}$$

$$\frac{\Gamma \vdash^{\mathsf{int}} e : \forall \overline{a}\,.\,\rho'}{\Gamma \vdash^{\mathsf{int}} e : [\overline{a \mapsto \tau'}]\rho'}\ \text{INST} \qquad \frac{\Gamma \vdash^{\mathsf{int}} e : \rho' \quad \overline{a} \# \Gamma}{\Gamma \vdash^{\mathsf{int}} e : \forall \overline{a}\,.\,\rho'}\ \text{GEN} \qquad \frac{\Gamma \vdash^{\mathsf{int}} e : \sigma_1' \quad \vdash^{\mathsf{F}} \lfloor \sigma_1' \rfloor \leq \sigma}{\Gamma \vdash^{\mathsf{int}} (e :: \sigma) : \sigma}\ \text{ANN}$$

Figure 6: The type system with System F instance

The differences with respect to the type system of Figure 2 are in rules APP and ANN. However, the two type systems type exactly the same programs.

Theorem 4.1. *If* $\Gamma \vdash e : \sigma'$ *then* $\Gamma \vdash^{\mathsf{int}} e : \sigma'$.

Proof. Straightforward induction, observing that $\lfloor \sigma_1' \rfloor = \lfloor \sigma_2' \rfloor$ implies $\vdash^{\mathsf{F}} \lfloor \sigma_1' \rfloor \leq \lfloor \sigma_2' \rfloor$. □

Theorem 4.2. *If* $\Gamma \vdash^{\mathsf{int}} e : \sigma'$ *then* $\Gamma \vdash e : \sigma'$.

Proof. By induction on the derivation of $\Gamma \vdash^{\mathsf{int}} e : \sigma'$. The only interesting case is really the application case (the annotation case is similar), where we have that $\Gamma \vdash^{\mathsf{int}} e_1\ e_2 : \sigma_2'$ where $\Gamma \vdash^{\mathsf{int}} e_1 : \sigma_1' \to \sigma_2'$ and $\Gamma \vdash^{\mathsf{int}} e_2 : \sigma_3'$ such that $\vdash^{\mathsf{F}} \lfloor \sigma_3' \rfloor \leq \lfloor \sigma_1' \rfloor$. By induction $\Gamma \vdash e_1 : \sigma_1' \to \sigma_2'$ and $\Gamma \vdash^{\mathsf{int}} e_2 : \sigma_3'$. It must also be the case that $\lfloor \sigma_3' \rfloor = \forall \overline{a}\,.\,\rho$ and $\lfloor \sigma_1' \rfloor = \forall \overline{b}\,.\,[\overline{a \mapsto \sigma}]\rho$, where $\overline{b} \# ftv(\forall \overline{a}\,.\,\rho)$ and without loss of generality assume also that $\overline{b} \# \Gamma$. Let us consider two cases for σ_3'.

- $\sigma_3' = \forall \overline{a}_1\,.\,\boxed{\forall \overline{a}_2\,.\,\rho}$ such that $\overline{a} = \overline{a}_1 \overline{a}_2$, and assume without loss of generality that $\overline{a} \# \Gamma$. Then by rule INST we get that $\Gamma \vdash e_2 : \boxed{\forall \overline{a}_2\,.\,\rho}$ and by rule SUBS and rule BI we get $\Gamma \vdash e_2 : \boxed{\rho}$. By rule GEN we get $\Gamma \vdash e_2 : \forall \overline{a}\,.\,\boxed{\rho}$ and by INST we get $\Gamma \vdash e_2 : \boxed{[\overline{a \mapsto \sigma}]\rho}$. By rule GEN we get finally that $\Gamma \vdash e_2 : \forall \overline{b}\,.\,\boxed{[\overline{a \mapsto \sigma}]\rho}$, and rule APP is applicable to finish the case.
- $\sigma_3' = \forall \overline{a}\,.\,\rho'$ such that $\rho = \lfloor \rho' \rfloor$. Then by rule INST we get that $\Gamma \vdash e_2 : [\overline{a \mapsto \boxed{\sigma}}]\rho'$ and by rule GEN we get that $\Gamma \vdash e_2 : \forall \overline{b}\,.\,[\overline{a \mapsto \boxed{\sigma}}]\rho'$. Then, rule APP is applicable and finishes the case.

□

In fact, as the previous theorem suggests, System F instance can be simulated by subsumptions, instantiations and generalizations in our type system.

Lemma 4.3. *If* $\Gamma \vdash e : \sigma_1'$ *and* $\vdash^{\mathsf{F}} \lfloor \sigma_1' \rfloor \leq \lfloor \sigma_2' \rfloor$ *then* $\Gamma \vdash e : \sigma_3'$ *such that* $\lfloor \sigma_3' \rfloor = \lfloor \sigma_2' \rfloor$.

Proof. Straightforward. □

Hence, the two declarative systems are equivalent. The type system of Figure 6 is more convenient to work with in the rest of this document, and hence we simply write $\vdash$ instead of $\vdash^{\mathsf{int}}$ below.

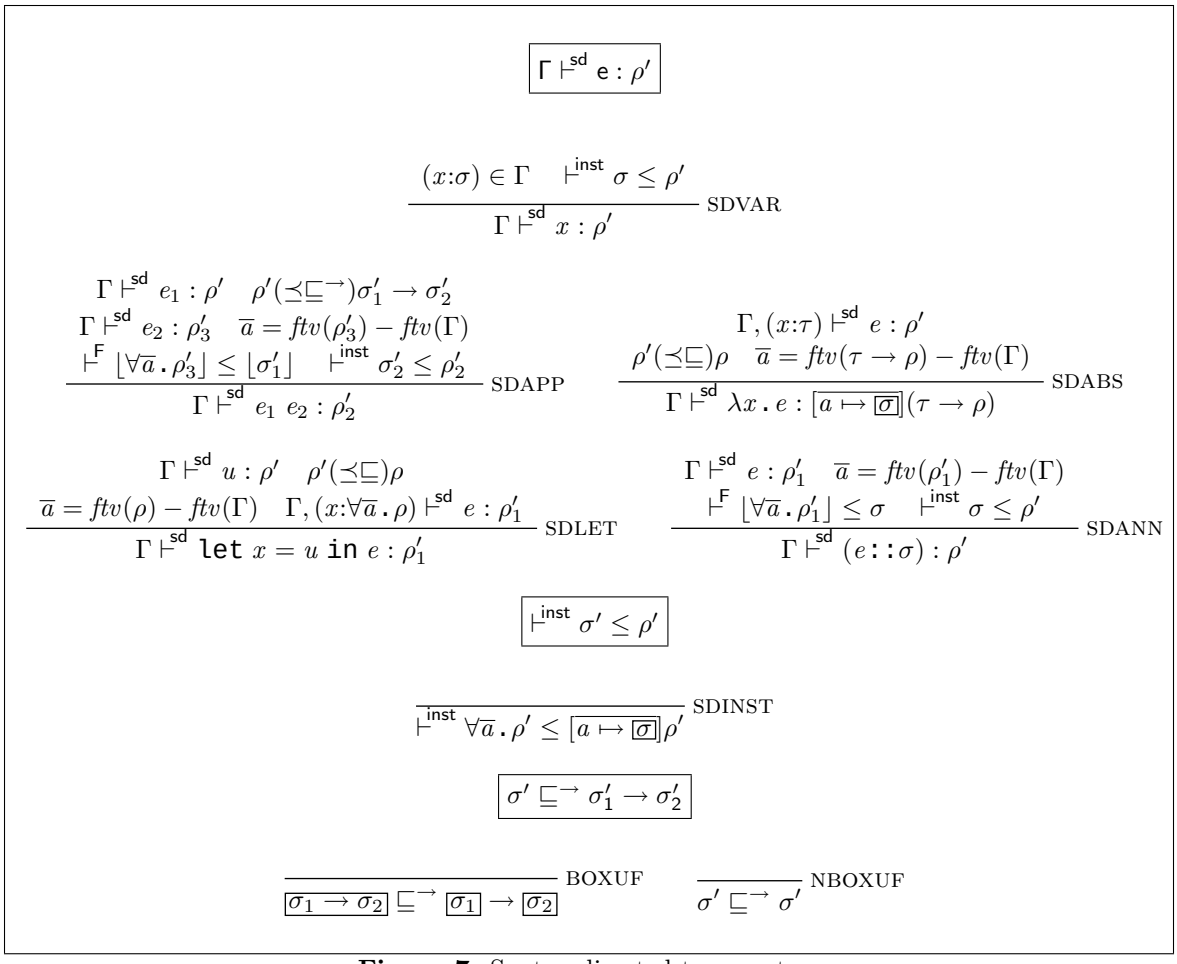

Figure 7: Syntax-directed type system

5 Syntax-directed specification

We now show how FPH may be implemented. The first step in establishing an algorithmic implementation is to specify a syntax-directed version of the type system of Figure 6, where uses of the non-syntax-directed rules (SUBS, INST, and GEN) have been pushed to appropriate nodes inside the syntax-tree. Subsequently we may proceed with a low-level implementation of the syntax-directed system (Section 6). Our syntax-directed presentation appears in Figure 7.

Rule SDVAR instantiates the type of a variable bound in the environment, using the auxiliary judgement, $\vdash^{\mathsf{inst}} \sigma' \leq \rho'$. The latter instantiates the top-level quantifiers of σ' to yield a ρ' type. However, we instantiate with boxes instead of τ' types, which is closer to the actual algorithm as boxes correspond to fresh "unification" variables.

Rule SDAPP deals with applications. It infers a type ρ' for the function, and uses $\preceq$ (Figure 3) and $\sqsubseteq^{\rightarrow}$ (a subset of $\sqsubseteq$) to expose an arrow constructor. The latter step is called *arrow unification*. Then SDAPP infers a ρ'_3 type for the argument of the application, generalizes over free variables that do not appear in the environment and checks that the result is more polymorphic (along the System F type instance) than the required type. Finally SDAPP instantiates the return type.

Rule SDABS uses a τ type for the argument of the λ-abstraction, and then forces the returned type ρ' for the body to be unboxed to a ρ-type using $\rho' \preceq\sqsubseteq \rho$. Finally, we consider all the free variables of the abstraction type that do not appear in the environment, and substitute them with arbitrary boxes. The returned type for the λ-abstraction is $\lceil a \mapsto \boxed{\sigma} \rceil (\tau \rightarrow \rho)$.

This last step, of generalization and instantiation, is perhaps puzzling. After all rule ABS (in the declarative specification of Figure 2) seems to only force λ-abstractions to have box-free types. Here is an example to show why it is needed:

Example 5.1 (Impredicative instantiations in λ-abstractions)**.** The following derivation holds: $\Gamma \vdash (\lambda x\,.\,x)\ \texttt{ids} : \boxed{[\forall a\,.\,a \to a]}$.

To construct a derivation for Example 5.1 observe that we can instantiate $\lambda x\,.\,x$ with a polymorphic argument type, as follows:

$$\dfrac{\dfrac{\dfrac{\Gamma, (x{:}a) \vdash x : a}{\Gamma \vdash \lambda x\,.\,x : a \to a}\ \text{ABS}}{\Gamma \vdash \lambda x\,.\,x : \forall a\,.\,a \to a}\ \text{GEN}}{\Gamma \vdash \lambda x\,.\,x : \boxed{[\forall a\,.\,a \to a]} \to \boxed{[\forall a\,.\,a \to a]}}\ \text{INST}$$

The use of GEN and INST are essential to make the term applicable to $\texttt{ids} : [\forall a.a \to a]$. The generalization and instantiation in SDABS ensure that GEN and INST are performed at each λ-abstraction, much as SDLET ensures that GEN is performed at each `let`-binding.

Rule SDLET is straightforward; after inferring a type for u which may contain boxes, we check that the boxes can be removed by $\preceq\!\sqsubseteq$ to get a ρ-type, which can subsequently be generalized and pushed in the environment.

Finally, rule SDANN infers a type ρ_1' for the expression e, generalizes over its free variables not in the environment, and checks that this type is more polymorphic than the annotations. As the final step, the annotation type is instantiated.

We can now establish the soundness of the syntax-directed system with respect to the declarative one (of Figure 6):

Theorem 5.2 (Soundness of $\vdash^{\mathsf{sd}}$)**.** *If* $\Gamma \vdash^{\mathsf{sd}} e : \rho'$ *then* $\Gamma \vdash e : \rho'$.

Proof. Straightforward induction. □

To prove the converse direction we need some additional machinery. First, we introduce the predicative restriction of the $\vdash^{\mathsf{F}}$ relation, given below:

$$\dfrac{\overline{b} \# ftv(\forall \overline{a}\,.\,\rho)}{\vdash^{\mathsf{DM}} \forall \overline{a}\,.\,\rho \leq \forall \overline{b}\,.\,[\overline{a \mapsto \tau}]\rho}\ \text{SHSUBS}$$

This relation is the relation used in the original Damas-Milner type system. We additionally write $\vdash^{\mathsf{DM}} \Gamma_2 \leq \Gamma_1$ if for every $(x{:}\sigma_1) \in \Gamma_1$, there exists a σ_2 such that $(x{:}\sigma_2) \in \Gamma_2$, and $\vdash^{\mathsf{DM}} \sigma_2 \leq \sigma_1$.

We have the following lemmas about the System F, and the Damas-Milner instance relations.

Lemma 5.3 (Transitivity of $\vdash^{\mathsf{DM}}$ and $\vdash^{\mathsf{F}}$)**.** *If* $\vdash^{\mathsf{DM}} \sigma_1 \leq \sigma_2$ *and* $\vdash^{\mathsf{DM}} \sigma_2 \leq \sigma_3$ *then* $\vdash^{\mathsf{DM}} \sigma_1 \leq \sigma_3$. *If* $\vdash^{\mathsf{F}} \sigma_1 \leq \sigma_2$ *and* $\vdash^{\mathsf{F}} \sigma_2 \leq \sigma_3$ *then* $\vdash^{\mathsf{F}} \sigma_1 \leq \sigma_3$.

Proof. Easy inductions. □

Lemma 5.4 ($\vdash^{\mathsf{F}}$ substitution stability)**.** *If* $\vdash^{\mathsf{F}} \sigma_1 \leq \sigma_2$ *then* $\vdash^{\mathsf{F}} \lfloor \varphi\sigma_1 \rfloor \leq \lfloor \varphi\sigma_2 \rfloor$.

Proof. Easy induction. □

Lemma 5.5. *If* $\vdash^{\mathsf{DM}} \sigma_1 \leq \sigma_2$ *then* $ftv(\sigma_1) \subseteq ftv(\sigma_2)$.

Proof. Easy check. □

Lemma 5.6. *If* $\vdash^{\mathsf{F}} \sigma_1 \leq \sigma_2$ *then* $ftv(\sigma_1) \subseteq ftv(\sigma_2)$.

Proof. Easy check. □

The next folklore lemma is particualrly useful for establishing type inference completeness, and in essence is a commutation property between substitutions and generalizations.

Lemma 5.7 (Gen-subs commutation for $\vdash^{\mathsf{DM}}$)**.** *Assume that φ is box-free. Let $\overline{a} = ftv(\rho) - ftv(\Gamma)$ and $\overline{b} = ftv(\varphi\rho) - ftv(\varphi\Gamma)$. It follows that $\vdash^{\mathsf{DM}} \varphi(\forall \overline{a}\,.\,\rho) \leq \forall \overline{b}\,.\,\varphi\rho$.*

Proof. Easy unfolding of the definitions. □

Lemma 5.8 ($\vdash^{\mathsf{inst}}$ substitution stability)**.** *If $\vdash^{\mathsf{inst}} \sigma' \leq \rho'$ then $\vdash^{\mathsf{inst}} \varphi\sigma' \leq \varphi\rho'$.*

Proof. Assume $\sigma' = \forall \overline{a}\,.\,\rho'_1$ such that $\overline{a}\#ftv(\varphi)$. Then we know that $\rho' = [\overline{a \mapsto \boxed{\sigma}}]\rho'_1$ and hence $\varphi\rho' = \varphi[\overline{a \mapsto \boxed{\sigma}}]\rho'_1 = [\overline{a \mapsto \boxed{\lfloor\varphi\sigma\rfloor}}]\varphi\rho'_1$ as required. □

Lemma 5.9 ($\sqsubseteq$ substitution stability)**.** *If $\sigma'_1 \sqsubseteq \sigma'_2$ then $\varphi\sigma'_1 \sqsubseteq \varphi\sigma'_2$.*

Proof. Easy induction. □

Corollary 5.10 ($\preceq\!\!\sqsubseteq$ substitution stability)**.** *If $\sigma'_1 \preceq\!\!\sqsubseteq \sigma'_2$ then $\varphi\sigma'_1 \preceq\!\!\sqsubseteq \varphi\sigma'_2$.*

Proof. Follows by Lemma 5.4 and Lemma 5.9. □

Lemma 5.11 ($\sqsubseteq$ transitivity)**.** *If $\sigma'_1 \sqsubseteq \sigma'_2$ and $\sigma'_2 \sqsubseteq \sigma'_3$ then $\sigma'_1 \sqsubseteq \sigma'_3$.*

Proof. Easy induction on the sums of the heights of the two derivations. □

Lemma 5.12 (Bijection substitution)**.** *If $\Gamma \vdash^{\mathsf{sd}} e : \rho'$ and φ is a variable bijection then $\varphi\Gamma \vdash^{\mathsf{sd}} e : \varphi\rho'$ and the new derivation has the same height.*

Proof. Easy induction. □

Theorem 5.13 (Substitution)**.** *If $\Gamma \vdash^{\mathsf{sd}} e : \rho'$ and $dom(\varphi)\#ftv(\Gamma)$ then $\Gamma \vdash^{\mathsf{sd}} e : \varphi\rho'$.*

Proof. By induction on the height of the derivation of $\Gamma \vdash^{\mathsf{sd}} e : \rho'$. We have the following cases to consider:

- Case SDVAR. We have that $\Gamma \vdash^{\mathsf{sd}} x : \rho'$ given that $(x{:}\sigma) \in \Gamma$ and $\vdash^{\mathsf{inst}} \sigma \leq \rho'$. By inverting $\vdash^{\mathsf{inst}}$ we know that $\sigma = \forall \overline{a}\,.\,\rho$ and $\rho' = [\overline{a \mapsto \boxed{\sigma}}]\rho$. But $ftv(\forall \overline{a}\,.\,\rho)\#dom(\varphi)$ and assuming without loss of generality that $\overline{a}\#ftv(\varphi)$ we only need show that: $\Gamma \vdash^{\mathsf{sd}} x : \varphi[\overline{a \mapsto \boxed{\sigma}}]\rho$ But then:

$$\varphi[\overline{a \mapsto \boxed{\sigma}}]\rho = [\overline{a \mapsto \boxed{\lfloor\varphi\sigma\rfloor}}]\varphi\rho = [\overline{a \mapsto \boxed{\lfloor\varphi\sigma\rfloor}}]\rho$$

  To finish the case by rule SDVAR we only need show that $\vdash^{\mathsf{inst}} \forall \overline{a}\,.\,\rho \leq [\overline{a \mapsto \boxed{\lfloor\varphi\sigma\rfloor}}]\rho$, which is true.
- Case SDABS. In this case we have that: $\Gamma \vdash^{\mathsf{sd}} \lambda x\,.\,e : [\overline{a \mapsto \boxed{\sigma}}](\tau \to \rho)$, given that $\Gamma, (x{:}\tau) \vdash^{\mathsf{sd}} e : \rho'$, $\rho' \preceq\!\!\sqsubseteq \rho$, and $\overline{a} = ftv(\tau \to \rho) - ftv(\Gamma)$. Let us assume using Lemma 5.12 that $\overline{a}\#ftv(\varphi)$. Then we need to show that $\Gamma \vdash^{\mathsf{sd}} \lambda x\,.\,e : \varphi[\overline{a \mapsto \boxed{\sigma}}](\tau \to \rho)$. But $\varphi[\overline{a \mapsto \boxed{\sigma}}](\tau \to \rho) = [\overline{a \mapsto \boxed{\lfloor\varphi\sigma\rfloor}}]\varphi(\tau \to \rho)$. But notice that $ftv(\tau \to \rho) = \overline{a} \cup (ftv(\tau \to \rho) \cap ftv(\Gamma))$. Hence $\varphi(\tau \to \rho) = \tau \to \rho$. Hence we only need to show that $\Gamma \vdash^{\mathsf{sd}} \lambda x\,.\,e : [\overline{a \mapsto \boxed{\lfloor\varphi\sigma\rfloor}}](\tau \to \rho)$, which follows by applying directly rule SDABS.

- Case SDAPP. We have in this case that: $\Gamma \vdash^{\mathsf{sd}} e_1\ e_2 : \rho_2'$, given that $\Gamma \vdash^{\mathsf{sd}} e_1 : \rho'$, $\rho' \preceq\sqsubseteq^{\rightarrow} \sigma_1' \rightarrow \sigma_2'$, $\Gamma \vdash^{\mathsf{sd}} e_2 : \rho_3'$, $\overline{a} = ftv(\rho_3') - ftv(\Gamma)$, $\vdash^{\mathsf{F}} \lfloor \forall \overline{a}\,.\,\rho_3' \rfloor \leq \lfloor \sigma_1' \rfloor$, and finally $\vdash^{\mathsf{inst}} \sigma_2' \leq \rho_2'$. By induction hypothesis we have that: $\Gamma \vdash^{\mathsf{sd}} e_1 : \varphi\rho'$, and we can observe that $\varphi\rho' \preceq\sqsubseteq^{\rightarrow} \varphi\sigma_1' \rightarrow \varphi\sigma_2'$. Moreover, let us assume that $\overline{a}\#ftv(\varphi)$ by appealing to Lemma 5.12. Hence $\varphi(\lfloor \forall \overline{a}\,.\,\rho_3' \rfloor) = \lfloor \forall \overline{a}\,.\,\rho_3' \rfloor$. Then, we know that: $\vdash^{\mathsf{F}} \lfloor \forall \overline{a}\,.\,\rho_3' \rfloor \leq \lfloor \sigma_1' \rfloor$ and hence: $\vdash^{\mathsf{F}} \lfloor \varphi(\forall \overline{a}\,.\,\rho_3') \rfloor \leq \lfloor \varphi(\sigma_1') \rfloor$ by appealing to the $\vdash^{\mathsf{F}}$ substitution property. The case is finished with rule SDAPP if we show that $\vdash^{\mathsf{inst}} \varphi\sigma_2' \leq \varphi\rho_2'$, which holds by Lemma 5.8.
- Case SDLET. In this case we have that $\Gamma \vdash^{\mathsf{sd}} \mathtt{let}\ x = u\ \mathtt{in}\ e : \rho_1'$ given that $\Gamma \vdash^{\mathsf{sd}} u : \rho'$, $\rho' \preceq\sqsubseteq \rho$, $\overline{a} = ftv(\rho) - ftv(\Gamma)$. and $\Gamma, (x{:}\forall \overline{a}\,.\,\rho) \vdash^{\mathsf{sd}} e : \rho_1'$. By know that $\Gamma \vdash^{\mathsf{sd}} u : \rho'$. We can apply the induction hypothesis then to get that $\Gamma, (x{:}\forall \overline{a}\,.\,\rho) \vdash^{\mathsf{sd}} e : \varphi\rho_1'$, since $ftv(\forall \overline{a}\,.\,\rho) \subseteq ftv(\Gamma)$ and hence also $dom(\varphi)\#ftv(\forall \overline{a}\,.\,\rho)$, and the case is finished.
- Case SDANN. In this case we have that $\Gamma \vdash^{\mathsf{sd}} (e :: \sigma) : \rho'$ given that $\Gamma \vdash^{\mathsf{sd}} e : \rho_1'$, $\overline{a} = ftv(\rho_1') - ftv(\Gamma)$, and $\vdash^{\mathsf{F}} \lfloor \forall \overline{a}\,.\,\rho_1' \rfloor \leq \sigma$. Let us assume by appealing to Lemma 5.12 that $\overline{a}\#ftv(\varphi)$. Then we know that $\varphi\lfloor \forall \overline{a}\,.\,\rho_1' \rfloor = \lfloor \forall \overline{a}\,.\,\rho_1' \rfloor$. By stability of System F instance under substitution we then get that $\vdash^{\mathsf{F}} \lfloor \forall \overline{a}\,.\,\rho_1' \rfloor \leq \varphi\sigma$, and applying Lemma 5.8 and rule SDANN finishes the case.

□

Lemma 5.14 (Arrow unification). *If $\sigma' \sqsubseteq \sigma_1' \rightarrow \sigma_2'$, then*

- $\sigma' = \boxed{\sigma_1 \rightarrow \sigma_2}$ *with* $\boxed{\sigma_1} \sqsubseteq \sigma_1'$ *and* $\boxed{\sigma_2} \sqsubseteq \sigma_2'$, *or*
- $\sigma' = \sigma_1'' \rightarrow \sigma_2''$ *with* $\sigma_1'' \sqsubseteq \sigma_1'$ *and* $\sigma_2'' \sqsubseteq \sigma_2'$.

Proof. Easy induction on $\sqsubseteq$. □

Lemma 5.15 (Transitivity of $\preceq\sqsubseteq$). *If $\sigma_1' \preceq\sqsubseteq \sigma_2'$ and $\sigma_2' \preceq\sqsubseteq \sigma_3'$ then $\sigma_1' \preceq\sqsubseteq \sigma_3'$.*

Proof. Assume that $\sigma_1' \preceq\sqsubseteq \sigma_2'$ and $\sigma_2' \preceq\sqsubseteq \sigma_3'$. We consider cases on σ_2'.

- Assume that $\sigma_2' = \boxed{\sigma}$. By inversion on $\sqsubseteq$, since $\sigma_1' \preceq\sqsubseteq \boxed{\sigma}$, it must be that $\sigma_1' \preceq \boxed{\sigma}$, and hence we have two cases by inversion on $\preceq$:
  - Case $\sigma_1' = \boxed{\sigma}$ (using rule BR). Then, we know by assumption that $\boxed{\sigma} = \sigma_2' \preceq\sqsubseteq \sigma_3'$, and therefore the case is finished.
  - Case $\sigma_1' = \boxed{\forall \overline{a}\,.\,\rho}$ and $\boxed{\sigma} = \boxed{[\overline{a \mapsto \sigma_a}]\rho}$ (using rule BI). Moreover we know by assumptions that $\boxed{[\overline{a \mapsto \sigma_a}]\rho} \preceq\sqsubseteq \sigma_3'$. We consider cases on whether $\rho = a \in \overline{a}$ or not. If $\rho = a$ then assume that $\sigma_a = \forall \overline{b}\,.\,\rho_b$ in which case we know that $\boxed{\sigma_a} \preceq \boxed{[\overline{b \mapsto \sigma_b}]\rho_b} \sqsubseteq \sigma_3'$. But in that case, also $\sigma_1' = \boxed{\forall \overline{a}\,.\,\rho} = \boxed{\forall \overline{a}\,.\,a} \preceq \boxed{[\overline{b \mapsto \sigma_b}]\rho_b} \sqsubseteq \sigma_3'$. Consequently, $\sigma_1' \preceq\sqsubseteq \sigma_3'$ as required. If on the other hand $\rho \neq a$, this means that $\boxed{[\overline{a \mapsto \sigma_a}]\rho} \sqsubseteq \sigma_3'$. Hence, $\sigma_1' = \boxed{\forall \overline{a}\,.\,\rho} \preceq \boxed{[\overline{a \mapsto \sigma_a}]\rho} \sqsubseteq \sigma_3'$, that is $\sigma_1' \preceq\sqsubseteq \sigma_3'$.
- Assume that $\sigma_2' \neq \boxed{\sigma}$ for any σ. It follows by inversion on $\preceq$ that $\sigma_2' \sqsubseteq \sigma_3'$. Therefore we have $\sigma_1' \preceq\sqsubseteq \sigma_2' \sqsubseteq \sigma_3'$, and by transitivity of $\sqsubseteq$ we get $\sigma_1' \preceq\sqsubseteq \sigma_3'$.

□

Theorem 5.16 (Completeness of syntax-directed system). *Assume that $\Gamma_1 \vdash e : \forall \overline{a}\,.\,\rho'$. Then, for all Γ_2 with $\vdash^{\mathsf{DM}} \Gamma_2 \leq \Gamma_1$ and for all $\overline{\sigma}$ there exists a ρ_0' such that $\Gamma_2 \vdash^{\mathsf{sd}} e : \rho_0'$ and $\rho_0' \preceq\sqsubseteq [\overline{a \mapsto \boxed{\sigma}}]\rho'$.*

Proof. By induction on the derivation of $\Gamma_1 \vdash e : \forall \overline{a}\,.\,\rho'$. We consider the following cases:

- Case VAR. We have in this case that $(x{:}\sigma_1) \in \Gamma_1$ and $\Gamma_1 \vdash x : \sigma_1$. Then $(x{:}\sigma_2) \in \Gamma_2$ and moreover $\vdash^{\mathsf{DM}} \sigma_2 \leq \sigma_1$. Assume that $\sigma_2 = \forall \overline{a}\,.\,\rho$ and $\sigma_1 = \forall \overline{b}\,.\,[\overline{a \mapsto \tau}]\rho$ given that $\overline{b} \# ftv(\forall \overline{a}\,.\,\rho)$. Then, let us pick $\overline{\sigma}$. We need to find $\overline{\sigma}_a$ such that $[\overline{a \mapsto \boxed{\sigma_a}}]\rho \preceq\!\sqsubseteq [\overline{b \mapsto \boxed{\sigma}}][\overline{a \mapsto \tau}]\rho$. For this we can simply pick each σ_a to be the type $\lfloor [\overline{b \mapsto \boxed{\sigma}}]\tau \rfloor$, and it is actually $[\overline{a \mapsto \boxed{\sigma_a}}]\rho \sqsubseteq [\overline{b \mapsto \boxed{\sigma}}][\overline{a \mapsto \tau}]\rho$. With reflexivity of $\preceq$, we get $[\overline{a \mapsto \boxed{\sigma_a}}]\rho \preceq\!\sqsubseteq [\overline{b \mapsto \boxed{\sigma}}][\overline{a \mapsto \tau}]\rho$ as required.
- Case ABS. We have that $\Gamma_1 \vdash \lambda x\,.\,e : \tau \to \rho$ given that $\Gamma_1, (x{:}\tau) \vdash e : \rho$. By induction hypothesis we get that $\Gamma_2, (x{:}\tau) \vdash^{\mathsf{sd}} e : \rho_0'$ such that $\rho_0' \preceq\!\sqsubseteq \rho$. Then, consider all $\overline{a} \in ftv(\tau, \rho) - ftv(\Gamma_2)$. Consider the substitution: $[a \mapsto \boxed{a}]$ and $[\overline{a \mapsto \boxed{a}}](\tau \to \rho) \preceq\!\sqsubseteq \tau \to \rho$, as required if we apply rule SDABS to get that $\Gamma_2 \vdash^{\mathsf{sd}} \lambda x\,.\,e : [\overline{a \mapsto \boxed{a}}](\tau \to \rho)$.
- Case INST. We have in this case that $\Gamma_1 \vdash e : [\overline{a \mapsto \tau'}]\rho'$ given that $\Gamma_1 \vdash e : \forall \overline{a}\,.\,\rho'$. By induction hypothesis we have that for all vectors $\overline{\sigma}$, we have $\Gamma_2 \vdash^{\mathsf{sd}} e : \rho_0'$ such that $\rho_0' \preceq\!\sqsubseteq [\overline{a \mapsto \boxed{\sigma}}]\rho'$. We need to show that $\rho_0' \preceq\!\sqsubseteq [\overline{a \mapsto \tau'}]\rho'$. By picking $\overline{\sigma} = \lfloor \overline{\tau'} \rfloor$, we know that:

$$\rho_0' \preceq\!\sqsubseteq [\overline{a \mapsto \boxed{\lfloor \tau' \rfloor}}]\rho' \sqsubseteq [\overline{a \mapsto \tau'}]\rho$$

  and by transitivity of $\sqsubseteq$, we get $\rho_0' \preceq\!\sqsubseteq [\overline{a \mapsto \tau'}]\rho$ as required.
- Case GEN. We have that $\Gamma_1 \vdash e : \forall \overline{a}\,.\,\rho'$ given that $\Gamma_1 \vdash e : \rho'$ and $\overline{a} \# \Gamma_1$. By induction hypothesis we get that $\Gamma_2 \vdash^{\mathsf{sd}} e : \rho_0'$ such that $\rho_0' \preceq\!\sqsubseteq \rho'$. We need to find a ρ_0'' such that $\rho_0'' \preceq\!\sqsubseteq [\overline{a \mapsto \boxed{\sigma}}]\rho'$. Because $\vdash^{\mathsf{DM}} \Gamma_2 \leq \Gamma_1$ it is easy to prove that also $\overline{a} \# \Gamma_2$. Then, by Theorem 5.13 we get that: $\Gamma_2 \vdash^{\mathsf{sd}} e : [\overline{a \mapsto \boxed{\sigma}}]\rho_0'$. Let $\rho_0'' = [\overline{a \mapsto \boxed{\sigma}}]\rho_0'$. Because $\rho_0' \preceq\!\sqsubseteq \rho'$, by substitution stability for $\preceq\!\sqsubseteq$ we can get $[\overline{a \mapsto \boxed{\sigma}}]\rho_0' \preceq\!\sqsubseteq [\overline{a \mapsto \boxed{\sigma}}]\rho'$ as required.
- Case SUBS. We have that $\Gamma_1 \vdash e : \rho''$ given that $\Gamma_1 \vdash e : \rho'$ and $\rho' \preceq\!\sqsubseteq \rho''$. By induction hypothesis we get that $\Gamma_2 \vdash^{\mathsf{sd}} e : \rho_0'$ such that $\rho_0' \preceq\!\sqsubseteq \rho'$ and by transitivity of $\preceq\!\sqsubseteq$ (Lemma 5.15) we get $\rho_0' \preceq\!\sqsubseteq \rho''$ as required.
- Case LET. In this case we have that: $\Gamma_1 \vdash \mathtt{let}\ x = u\ \mathtt{in}\ e : \rho'$, given that $\Gamma_1 \vdash u : \forall \overline{a}\,.\,\rho$ and $\Gamma_1, (x{:}\forall \overline{a}\,.\,\rho) \vdash e : \rho'$. By induction hypothesis we get that for all vectors σ there exists a ρ_0' such that: $\Gamma_2 \vdash^{\mathsf{sd}} u : \rho_0'$ and $\rho_0' \preceq\!\sqsubseteq [\overline{a \mapsto \boxed{\sigma}}]\rho$. Pick in particular the ρ_0' such that $\rho_0' \preceq\!\sqsubseteq [\overline{a \mapsto \boxed{a}}]\rho$. Then, it is easy to see that $[\overline{a \mapsto \boxed{a}}]\rho \sqsubseteq \rho$ and hence $\rho_0' \preceq\!\sqsubseteq \rho$ by transitivity of $\sqsubseteq$. Since $\vdash^{\mathsf{DM}} \Gamma_2 \leq \Gamma_1$ it is the case that $ftv(\Gamma_2) \subseteq ftv(\Gamma_1)$ (by Lemma 5.5), and hence for $\overline{b} = ftv(\rho) - ftv(\Gamma_2)$ we get that $\vdash^{\mathsf{DM}} \forall \overline{b}\,.\,\rho \leq \forall \overline{a}\,.\,\rho$. By induction hypothesis then we get that $\Gamma_2, (x{:}\forall \overline{b}\,.\,\rho) \vdash^{\mathsf{sd}} e : \rho_1'$ such that $\rho_1' \preceq\!\sqsubseteq \rho'$. Applying rule SDLET finishes the case.
- Case ANN. We have that $\Gamma_1 \vdash (e :: \sigma) : \sigma$, given that $\Gamma_1 \vdash e : \sigma_1'$ and $\vdash^{\mathsf{F}} \lfloor \sigma_1' \rfloor \leq \sigma$. Assume that $\sigma_1' = \forall \overline{a}\,.\,\rho_1'$. Then, by induction we get that for all $\overline{\sigma}$ there exists a ρ_0' with: $\Gamma_2 \vdash^{\mathsf{sd}} e : \rho_0'$ such that $\rho_0' \preceq\!\sqsubseteq [\overline{a \mapsto \boxed{\sigma}}]\rho_1'$. Pick in particular the ρ_0' that makes true that $\rho_0' \preceq\!\sqsubseteq [\overline{a \mapsto \boxed{a}}]\rho_1'$. Then let $\overline{b} = ftv(\rho_0') - ftv(\Gamma_2)$. If $\vdash^{\mathsf{F}} \lfloor \forall \overline{a}\,.\,\rho_1' \rfloor \leq \sigma$ then it must also be $\vdash^{\mathsf{F}} \lfloor \forall \overline{b}\,.\,\rho_0' \rfloor \leq \sigma$, by using transitivity of $\vdash^{\mathsf{F}}$ since, $\vdash^{\mathsf{F}} \lfloor \forall \overline{b}\,.\,\rho_0' \rfloor \leq \lfloor \forall \overline{a}\,.\,\rho_1' \rfloor$. The case concludes by appealing to rule SDANN in the same way as in the case of rule VAR.
- Case APP. We have that $\Gamma_1 \vdash e_1\ e_2 : \sigma_2'$ given that $\Gamma_1 \vdash e_1 : \sigma_1' \to \sigma_2'$, $\Gamma_1 \vdash e_2 : \sigma_3'$, and $\vdash^{\mathsf{F}} \lfloor \sigma_3' \rfloor \leq \lfloor \sigma_1' \rfloor$. By induction hypothesis we have that $\Gamma_2 \vdash^{\mathsf{sd}} e_1 : \rho_1'$ such that $\rho_1' \preceq\!\sqsubseteq \sigma_1' \to \sigma_2'$. Let ρ_1'' be the particular type such that $\rho_1' \preceq \rho_1'' \sqsubseteq \sigma_1' \to \sigma_2'$. Then, by Lemma 5.14 we have two cases:
  - $\rho_1'' = \boxed{\sigma_1 \to \sigma_2}$ with $\boxed{\sigma_1} \sqsubseteq \sigma_1'$ and $\boxed{\sigma_2} \sqsubseteq \sigma_2'$, or
  - $\rho_1'' = \sigma_1'' \to \sigma_2''$ with $\sigma_1'' \sqsubseteq \sigma_1'$ and $\sigma_2'' \sqsubseteq \sigma_2'$.

  Let σ_{11}' be either σ_1'' or $\boxed{\sigma_1}$. In any case it is the case that $\lfloor \sigma_{11}' \rfloor = \lfloor \sigma_1' \rfloor$. And let σ_{22}' be either σ_2'' or $\boxed{\sigma_2}$. Moreover, let us assume that $\sigma_3' = \forall \overline{b}\,.\,\rho_3'$. Then, by induction hypothesis there exists a ρ_3'' such that $\Gamma_2 \vdash^{\mathsf{sd}} e_3 : \rho_3''$ and $\rho_3'' \preceq\!\sqsubseteq [\overline{b \mapsto \boxed{b}}]\rho_3'$. Moreover let $\overline{a} = ftv(\rho_3'') - ftv(\Gamma_2)$. It must be the case that: $\vdash^{\mathsf{F}} \lfloor \forall \overline{a}\,.\,\rho_3'' \rfloor \leq \lfloor \forall \overline{b}\,.\,\rho_3' \rfloor$ and we know that $\vdash^{\mathsf{F}} \lfloor \sigma_3' \rfloor \leq \lfloor \sigma_1' \rfloor$. By transitivity of $\vdash^{\mathsf{F}}$ then we get that $\vdash^{\mathsf{F}} \lfloor \forall \overline{a}\,.\,\rho_3'' \rfloor \leq \lfloor \sigma_1' \rfloor = \lfloor \sigma_{11}' \rfloor$. To conclude the case using rule SDAPP we have the following. Assume that $\sigma_2' = \forall \overline{c}\,.\,\rho_2'$. Pick any vector $\overline{\sigma}$. We need to find an instantiation of σ_{22}', ρ_{22}' such that $\rho_{22}' \preceq\!\sqsubseteq [\overline{c \mapsto \boxed{\sigma}}]\rho_2'$. We take cases according to the form of σ_{22}':

– Case $\sigma'_{22} = \boxed{\sigma_2}$. Then it is the case that $\boxed{\sigma_2} \sqsubseteq \forall \overline{c}\,.\,\rho'_2$, which means that $\overline{c} = \emptyset$ and σ_2 is a ρ-type, by inversion on the $\sqsubseteq$ relation. In which case we have the required result because σ'_{22} cannot be further instantiated.
– Case $\sigma'_{22} = \sigma''_2 \sqsubseteq \forall \overline{c}\,.\,\rho'_2 = \sigma'_2$. By inversion again we either get that $\sigma''_2 = \forall \overline{c}\,.\,\rho'_2$, or that $\sigma''_2 = \forall \overline{c}\,.\,\rho''_2$ with $\rho''_2 \sqsubseteq \rho'_2$ and by the substitution lemma for $\sqsubseteq$ we get that $[\overline{c \mapsto \boxed{\sigma}}]\rho''_2 \sqsubseteq [\overline{c \mapsto \boxed{\sigma}}]\rho'_2$, i.e. by reflexivity of $\preceq$, $[\overline{c \mapsto \boxed{\sigma}}]\rho''_2 \preceq\sqsubseteq [\overline{c \mapsto \boxed{\sigma}}]\rho'_2 \vdash^{\mathsf{inst}} \sigma'_{22} \leq [\overline{c \mapsto \boxed{\sigma}}]\rho''_2$, as required.

□

To see an example of this theorem, consider the following context $\Gamma = \{head{:}\forall a\,.\,[a] \to a, ids{:}[\forall b\,.\,b \to b]\}$. Now, one can see that $\Gamma \vdash (head\ ids) : \boxed{\forall c\,.\,c \to c} \to \boxed{\forall c\,.\,c \to c}$ but in the syntax-directed system we only get $\Gamma \vdash^{\mathsf{sd}} (head\ ids) : \boxed{\forall a\,.\,a \to a}$, and it is the case that $\boxed{\forall a\,.\,a \to a} \preceq\sqsubseteq \boxed{\forall c\,.\,c \to c} \to \boxed{\forall c\,.\,c \to c}$.

We are now in a state to give the completeness of the syntax-directed specification with respect to the declarative system.

Corollary 5.17. *If* $\Gamma \vdash e : \rho'$ *then* $\Gamma \vdash^{\mathsf{sd}} e : \rho'_0$ *such that* $\rho'_0 \preceq\sqsubseteq \rho'$.

Proof. The result follows by Theorem 5.16, observing that $\vdash^{\mathsf{DM}}$ is reflexive. □

We also state one further corollary, which is a key ingredient to showing the implementability of the syntax-directed system by a low-level algorithm (to be described in Section 6).

Corollary 5.18 (Strengthening)**.** *If* $\Gamma_1 \vdash^{\mathsf{sd}} e : \rho'_1$ *and* $\vdash^{\mathsf{DM}} \Gamma_2 \leq \Gamma_1$ *then* $\Gamma_2 \vdash^{\mathsf{sd}} e : \rho'_2$ *such that* $\rho'_2 \preceq\sqsubseteq \rho'_1$.

Proof. The corollary follows by appealing to Theorem 5.2 and Theorem 5.16. □

Corollary 5.18 means that if we change the types of expressions in the environments to be the most general according to the predicative $\vdash^{\mathsf{DM}}$, typeability is not affected. This property is important for type inference completeness for the following reason: All types that are pushed in the environment are box-free and hence can only differ by each other in $\vdash^{\mathsf{DM}}$ relation—their polymorphic parts are determined by annotations. In fact the algorithm will choose the most general of them according to $\vdash^{\mathsf{DM}}$. Therefore, if an expression is typeable in the declarative type system with bindings in the environments that *do not have their most general types*, the above corollary shows that the expression will also be typeable if these bindings are assigned their most general types, that is, the types that the algorithm infers for them.

6 Algorithmic implementation

The syntax-directed specification of Figure 7 can be implemented by a low-level constraint-based algorithm, which resembles the algorithm of ML^{F} (Le Botlan and Rémy 2003). To ease the transition into the detailed low-level description of the algorithm, we present an approximate description of the basic ideas behind the implementation in Section 6.1, and we elaborate in Section 6.2. We present the metatheory of the implementation in Section 6.3.

6.1 The basic ideas

Like Hindley-Damas-Milner type inference (Damas and Milner 1982; Milner 1978), our algorithm creates fresh *unification variables* to instantiate polymorphic types, and to use as the argument types of abstractions. In Hindley-Damas-Milner type inference these variables are unified with other

types. Hence, a Hindley-Damas-Milner type inference engine maintains a set of *equality constraints* that map each unification variable to some type, updating the constraints as type inference proceeds.

Our algorithm uses a similar structure to Hindley-Damas-Milner type inference, but maintains both equality and *instance constraints* during type inference, so we use the term *constrained variable* instead of unification variable. A constrained variable in the algorithm corresponds to a box in the high-level specification. To distinguish between constrained variables and (rigid) quantified variables, we use greek letters α, β, for the latter. Therefore, the algorithm manipulates types with the following syntax:

$$\begin{array}{lcl} \tau & ::= & a \mid \tau \to \tau \mid \mathsf{T}\,\overline{\tau} \mid \alpha \\ \rho & ::= & \tau \mid \sigma \to \sigma \mid \mathsf{T}\,\overline{\sigma} \\ \sigma & ::= & \forall \overline{a}\,.\,\rho \end{array}$$

An algorithmic type with no constrained variables can be viewed as an ordinary System F (box-free) type, and we will ommit explicit coercions when we need to do this.

The need for instance constraints can be motivated by the typing of `choose ids` from the introduction. First, since `choose` has type $\forall a\,.\,a \to a \to a$, we may instantiate the quantified variable a with a fresh constrained variable α. However, when we meet the argument `id`, it becomes unclear whether α should be equal to $\beta \to \beta$ (that would arise from instantiating the type of `id`), or $\forall b\,.\,b \to b$ (if we do not instantiate `id`). In the high-level specification we can clairvoyantly make a (potentially boxed) choice that suits us. The algorithm does not have the luxury of clairvoyance, so rather than making a choice, it must instead simply record an *instance constraint*. In this case, the instance constraint specifies that α can be *any System F instance* of $\forall b\,.\,b \to b$. To express this, at first approximation, we need constraints of the form $\alpha \geq \sigma$.

However, we need to go slightly beyond this constraint form. Consider the program `f (choose id)` where `f` has type $\forall c\,.\,c \to c$. After we instantiate the quantified variable c with a fresh variable γ, we must constrain γ by the type of `choose id`, thus

$$\gamma \geq (\text{principal type of } \texttt{choose id})$$

But, the principal type of `choose id` must be a type that is quantified *and* constrained at the same time: $[\alpha \geq \forall b\,.\,b \to b] \Rightarrow \alpha \to \alpha$. Following ML^F (Le Botlan and Rémy 2003), this *scheme* captures the *set* of all types for `choose id`, such as $\forall d\,.\,(d \to d) \to (d \to d)$ or $(\forall b\,.\,b \to b) \to (\forall b\,.\,b \to b)$. We hence extend the bounds of constrained variables to include $\gamma \geq \varsigma$, where ς is a scheme. Schemes ς are of the form $[c_1, \ldots, c_n] \Rightarrow \rho$, where each constraint c_i is of the form $(\alpha \geq \varsigma)$, or $(\alpha = \sigma)$, or $(\alpha\ \bot)$. [4] The constraint $\alpha\ \bot$ means that α is unconstrained. Ordinary System F types can be viewed as schemes whose quantified variables are unconstrained, and hence the type $\forall b\,.\,b \to b$ can be written as $[\beta\ \bot] \Rightarrow \beta \to \beta$. The meaning of the constraint $\gamma \geq \varsigma$ is that γ belongs in the *set of System F types that* ς *represents*, which we write $[\![\varsigma]\!]$. For example, if $\varsigma = [\alpha \geq ([\beta\ \bot] \Rightarrow \beta \to \beta)] \Rightarrow (\alpha \to \alpha)$, then we have:

$$\begin{array}{rcl} (\forall b\,.\,b \to b) \to (\forall b\,.\,b \to b) & \in & [\![\varsigma]\!] \\ \forall c\,.\,(c \to c) \to c \to c & \in & [\![\varsigma]\!] \\ \forall c\,.\,([c] \to [c]) \to [c] \to [c] & \in & [\![\varsigma]\!] \end{array}$$

6.2 Description of the algorithm

We immediately proceed to the detailed description of the low-level algorithmic implementation of the syntax-directed type system in Figure 7. The formal language of types and constraints that we use in the implementation is given in Figure 8. Monomorphic types τ contain "rigid" variables a (such as quantified variables), and constrained (unification) variables $\alpha, \beta, \ldots$. As outlined previously, constrained variables represent the boxes of the high-level type system. A box that is filled in with some type should be viewed as two components: a constrained variable, and a substitution that

[4] In reality, the syntax of constraints involves also special flags on variables. As a first approximation we may ignore this detail, but we will be precise in the detailed description of the algorithm, in the next section.

Algorithm types	σ	::=	$\forall \overline{a}\,.\,\rho$
	ρ	::=	$\tau \mid \sigma \to \sigma$
	τ	::=	$a \mid \alpha \mid \tau \to \tau$
Constraints	C	::=	$\overline{\alpha_\mu\ bnd}$
Flags	μ	::=	$\mathsf{m} \mid \star$
Schemes	ς	::=	$[C] \Rightarrow \rho$
Bounds	bnd	::=	$(\geq \varsigma) \mid (= \sigma) \mid \bot$
Flagsets	Δ	::=	$\overline{\alpha_\mu}$

Figure 8: Algorithmic types, schemes, and constraints

satisfies the constraint that is attached to the variable, and which maps the variable to the contents of the box.

Constraints C are finite maps of the form $\overline{\alpha_\mu\ bnd}$. Every variable α is mapped to a flag μ and a bound bnd. The domain of a constraint C, written as $dom(C)$, is the set $\{\alpha \mid (\alpha_\mu\ bnd) \in C\}$. As described previously, bounds bnd specify either flexible constraints, rigid constraints, or the fact that a constrained variable is yet completely unconstrained.

The new bit, compared to our previous description, is the presence of the flags μ. These flags are used to ensure that the variables that enter the environments are never unified with types that contain quantifiers, and are hence monomorphic. This is important to ensure that abstraction argument types are kept monomorphic, as the high-level specification requires. Those constrained variables that must be kept monomorphic are flagged with m, whereas variables with no restrictions are flagged with $\star$.

Flagsets Δ are a notational convenience, and can be viewed as special constraints that only give information about the flags of the variables of a constraint. For a constraint C, we write Δ_C for the flagset of the domain of C. Additionally, and overloading notation, we write $fcv(\cdot)$ (for *free cosntrained variables*) for the set of constrained variables of the argument. For example, $fcv(C)$ is the set of all constrained variables appearing anywhere in C. We write $\Delta(\beta) = \mu$ whenever $\beta_\mu \in \Delta$, and $dom(\Delta)$ for the domain of Δ. We write $\Delta_1\Delta_2$ for the disjoint union of the two flagsets with respect the variables in their domains.

We elaborate in the next section on the form of the bounds bnd, as well as the meaning of constraints.

6.2.1 Bounds, and the meaning of constraints

Figure 8 presents the concrete syntax for constraints and schemes. Apart from the *flexible* bounds of the form $\geq \varsigma$ we additionally need *rigid* bounds that stand for equalities and are of the form $= \sigma$. Equalities arise because of invariant data type constructors. Unrestricted bounds $\bot$ impose no constraints. Schemes are of the form $[C] \Rightarrow \rho$. The variables that appear in the domain of C are *considered bound* in ρ. Hence the domain of C can be viewed as a set of quantified but constrained variables.

Because schemes $[D] \Rightarrow \rho$ bind the constrained variables in the domain of D inside the body ρ we can assume without loss of generality that D can be concatenated with any external constraint C, to form the constraint $C \bullet D$. This corresponds to ensuring freshness of the constrained variables in the domain of D.

Definition 6.1 (Constraint disjoint concatenation)**.** For two constraints C and D, we write $C \bullet D$ to denote their union whenever $dom(D) \# fcv(C)$.

$\boxed{\Delta \vdash \mathsf{C}}$

$$\frac{\begin{array}{l}(a)\ dom(C)\#dom(\Delta)\\ (b)\text{ for all }(\alpha_\mu\ bnd)\in C, \alpha\notin \widehat{C}(bnd)\text{ and }fcv(bnd)\subseteq dom(C)\cup dom(\Delta)\\ (c)\text{ for all }(\alpha_{\mathsf{m}}\ bnd)\in C, \text{for all }\beta\in fcv(bnd), \Delta\Delta_c(\beta)=\mathsf{m}\text{ and }(bnd=\bot\text{ or }bnd=\tau)\\ (d)\text{ for all }(\alpha_\mu\geq\varsigma)\in C, \Delta\Delta_C\vdash\varsigma\text{ and }\mu=\star\end{array}}{\Delta\vdash C}\ \textsc{cwf}$$

$\boxed{\Delta \vdash \sigma}$ $\qquad$ $\boxed{\Delta \vdash \varsigma}$

$$\frac{fcv(\sigma)\subseteq dom(\Delta)}{\Delta\vdash\sigma}\ \textsc{wft} \qquad \frac{\begin{array}{l}(a)\ \Delta\vdash D\quad fcv([D]\Rightarrow\rho)\subseteq dom(\Delta)\\ (b)\text{ for all }\gamma\in dom(D), \Delta_D(\gamma)=\star\\ (c)\text{ if }\rho=\gamma\text{ then }(\gamma_\star\ \bot)\in D\text{ or }\Delta(\gamma)=\mathsf{m}\end{array}}{\Delta\vdash[D]\Rightarrow\rho}\ \textsc{wfs}$$

Figure 9: Well formed constraints, types, and schemes

We now define the reachable variables of types and schemes *through a constraint*, with the (overloaded) function $\widehat{C}(\cdot)$, below:

$$\begin{array}{lcl}\widehat{C}(\alpha)\mid\alpha\notin dom(C) & = & \{\alpha\}\\ \widehat{C}(\alpha)\mid(\alpha_\mu\ \bot)\in C & = & \{\alpha\}\\ \widehat{C}(\alpha)\mid(\alpha_\mu=\sigma)\in C & = & \{\alpha\}\cup\widehat{C}(\sigma)\\ \widehat{C}(\alpha)\mid(\alpha_\mu\geq\varsigma)\in C & = & \{\alpha\}\cup\widehat{C}(\varsigma)\\ \widehat{C}(a) & = & \emptyset\\ \widehat{C}(\sigma_1\rightarrow\sigma_2) & = & \widehat{C}(\sigma_1)\cup\widehat{C}(\sigma_2)\\ \widehat{C}(\forall\overline{a}\,.\,\rho) & = & \widehat{C}(\rho)\\ \widehat{C}([D]\Rightarrow\rho) & = & \widehat{C\bullet D}(\rho, dom(D))-dom(D)\end{array}$$

In the case for schemes, $\widehat{C}([D]\Rightarrow\rho)$, we gather all variables reachable from ρ and the domain of D through $C\bullet D$ except those quantified by the scheme ($dom(D)$). However, an invariant of our algorithm will be that there exist no useless quantified variables in $dom(D)$ in any scheme of the form $[D]\Rightarrow\rho$ —hence in reality it will suffice to only consider $\widehat{C\bullet D}(\rho)-dom(D)$.

Finally, the function $\widehat{C}(\cdot)$ exists since any constraint involves a finite set of constrained variables, and we can compute it by an iteration process, that adds new variables from the constraint C at each step. Although formally this corresponds to a guaranteed-to-exist fixpoint, we will simply be using the above set of equalities as the actual definition of $\widehat{C}(\sigma)$. We can now define $fcv(\cdot)$ using the reachable variables through the empty constraint. In particular $fcv(\varsigma)=\widehat{\emptyset}(\varsigma)$.

We now need to define the meaning of constraints and bounds. Before presenting the actual definition, we give conditions on the constraints that will help get insights about their meaning. Figure 9 presents the necessary conditions on constraints, types, and schemes, with the notations $\Delta\vdash C$, $\Delta\vdash\sigma$, and $\Delta\vdash\varsigma$ respectively.

The judgement $\Delta\vdash C$ ensures that the constraint C is well-formed in a given flagset Δ. In rule CWF, condition (a) ensures that the domain of C consists of fresh variables with respect to Δ. Condition (b) ensures that there exist no cycles in C and that all constrained variables in the bounds of C belong either in the domain of C or in the external $dom(\Delta)$. Condition (c) ensures that if a variable is flagged as monomorphic with m then it must be either unrestricted (bound to $\bot$) or mapped rigidly to some type τ whose constrained variables must be themselved flagged as m either in Δ or Δ_C. Condition (d) ensures that all flags of flexibly bound variables are $\star$. It makes no sense for a variable that must be monomorphic to be flexibly bound to a scheme, since a monomorphic variable can only be equal to an instance of the scheme. Finally, condition (d) also asserts that in all bounds of the form $\alpha_\star\geq\varsigma$, the constraint ς must itself be well-formed in the extended flagset $\Delta\Delta_C$.

The judgement $\Delta \vdash \sigma$ simply ensures that all constrained variables appear in the flagset Δ. The judgement $\Delta \vdash \varsigma$ ensures that the scheme ς is well-formed in the flagset Δ. Let us examine in detail the conditions of rule WFS, which asserts that $\Delta \vdash [D]\Rightarrow\rho$. Condition (a) ensures that the constraint D of the scheme is well-formed in Δ, and that all the free constrained variables of $[D]\Rightarrow\rho$ belong in the domain of Δ. Condition (b) states that all constrained variables in $dom(D)$ cannot be flagged as m. The reason is the following: If such a variable were flagged with m then it must have been the case that some variable *from the environment* has been mapped to it—in order to force its flagging. It follows that such a variable could not have been quantified in the scheme anyway, because schemes are constructed as results of generalization. Condition (c) ensures that the scheme $[D]\Rightarrow\rho$ is *normal*. That is, if the body is a single variable, then it must either be bound to $\bot$ in $dom(D)$ or it must be monomorphically flagged in Δ. During inference a normalization procedure ensures that schemes are normal and our unification algorithm preserves normal schemes. We return to this condition in Section 6.2.2.

We are particularly interested in constraints that are *well-formed in their own flagsets*, i.e. constraints C where $\vdash C$. Our type inference algorithm produces and manipulates such constraints. Well-formed constraints correspond to sets of substitutions from constrained variables to constrained-variable-free types, generalizing the interpretation of schemes as sets of constrained-variable-free (System F) types. The idea exists in the ML^F line of work, and we take it a step further, to formalize type inference using this interpretation of constraints, instead of employing a syntactic instance relation that encodes set semantics. In order to present the interpretation of constraints and schemes we need to introduce the *quantifier rank of a constraint*.

Definition 6.2 (Constraint quantifier rank)**.** The quantifier rank of a constraint C, written $q(C)$, is defined as:

$$q(C) = |dom(C)| + \sum_{(\alpha_\mu \geq [D]\Rightarrow\rho) \in C} q(D)$$

Intuitively the metric $q(C)$ sums up the number of all constrained variables in C, icluding the constrained variables of all schemes appearing in C.

Definition 6.3 (Well-formed constraint interpretation)**.** A substitution θ from constrained variables to constrained-variable-free (System F) types satisfies C, written $\theta \models C$, iff:

1. for all $(\alpha_\mathsf{m}\ bnd) \in C$, it is the case that $\theta\alpha$ is quantifier-free.
2. for all $(\alpha_\mu = \sigma) \in C$, it is the case that $\theta\alpha = \theta\sigma$.
3. for all $(\alpha_\mu \geq \varsigma) \in C$, it is the case that $\theta\alpha \in [\![\theta\varsigma]\!]$.

where:

$$[\![[D]\Rightarrow\rho]\!] = \{\forall\overline{b}\,.\,\theta_D\rho \mid \overline{b}\#ftv([D]\Rightarrow\rho) \text{ and } \theta_D \models D\}$$

Of course we will be interested mainly in the interpretation of well-formed constraints, but the definition is valid for arbitrary constraints, so long that θ grounds all constrained variables to System F types.

Condition (1) ensures that all monomorphic variables are indeed mapped to monomorphic types. Condition (2) ensures that equalities are respected. Condition (3) realizes our set interpretation of schemes. We observe that the domain of θ must be a superset of the domain of C and hence in clause (3), $\theta\varsigma$ contains no free constrained variables. The interpretation $[\![[D]\Rightarrow\rho]\!]$ is nothing more than a generalization of System F instance. It requires the existence of a substitution θ_D that satisfies D, and we can generalize over type variables ($\overline{b}$) that θ_D introduced (condition $\overline{b}\#ftv([D]\Rightarrow\rho)$. Notice that, because of the well-formedness condition (c) in rule WFS, $\theta_D\rho$ may add no top-level quantifiers on ρ. Finally, the mutual definition of satisfying substitutions and scheme interpretations is well-formed by using as metric the quantifier rank $q(C)$ for $\theta \models C$, and $q(D)$ for the interpretation of $[D]\Rightarrow\rho$.

As an example, let us see how we can derive that

$$\forall b\,.\,(b \to b) \to (b \to b) \in [\![[\alpha_\star \geq ([\beta_\star \bot]\Rightarrow\beta \to \beta)]\Rightarrow(\alpha \to \alpha)]\!]$$

Consider the substitution $\theta_\alpha = [\alpha \mapsto (b \to b)]$. Since $(b \to b) \to (b \to b) = \theta_\alpha(\alpha \to \alpha)$ it suffices to show that $\theta_\alpha \models (\alpha_\star \geq [\beta_\star \bot] \Rightarrow \beta \to \beta)$. This will be the case if $\theta_\alpha \alpha \in [\![[\beta_\star \bot] \Rightarrow \beta \to \beta]\!]$, that is, if $b \to b \in [\![[\beta_\star \bot] \Rightarrow \beta \to \beta]\!]$. This follows by picking $\theta_\beta = [\beta \mapsto b]$.

6.2.2 Inference implementation

We proceed with explaining the basic ideas behind the reference implementation. The top-level inference algorithm is given with the function *infer*, that follows the syntax-directed presentation of Figure 7, and has signature:

$$\mathit{infer} : \mathit{Constraint} \times \mathit{Env} \times \mathit{Term} \to \mathit{Constraint} \times \mathit{Type}$$

The function *infer* accepts a constraint C_1, an environment Γ, and a term e. A call to $\mathit{infer}(C_1, \Gamma, e)$ either fails with *fail* or returns an updated constraint C_2 and a type ρ. The most interesting case, which demonstrates the power of schemes, is the application case:

$$\begin{array}{rcl} \mathit{infer}(C, \Gamma, e_1\ e_2) & = & E_1, \rho_1 = \mathit{infer}(C, \Gamma, e_1) \\ & & E_2, \sigma_1 \to \sigma_2 = \mathit{instFun}(E_1, \Gamma, \rho_1) \\ & & E_3, \rho_3 = \mathit{infer}(E_2, \Gamma, e_2) \\ & & E_4, \varsigma_3 = \mathit{generalize}(\Gamma, E_3, \rho_3) \\ & & E_5 = \mathit{subsCheck}(E_4, \varsigma_3, \sigma_1) \\ & & \mathit{inst}(E_5, \sigma_2) \end{array}$$

In a call to $\mathit{infer}(C, \Gamma, e_1\ e_2)$ we peform the following steps:

- We first infer a type ρ_1 for e_1 and an updated constraint E_1, by calling $\mathit{infer}(C, \Gamma, e_1)$.
- However, type ρ_1 may itself be a constrained type variable, that is, it may correspond to a single box in the syntax-directed specification. The function $\mathit{instFun}(E_1, \Gamma, \rho_1)$ implements the relation $\preceq\!\!\sqsubseteq^{\to}$. Intuitively, if ρ_1 is a variable α that is not monomorphic in E_1, that is if $(\alpha_\star = \forall \overline{a}\,.\,\rho) \in E_1$ or $(\alpha_\star \geq [D] \Rightarrow \rho) \in E_1$, it is replaced by an instantiation of its bound (that is, with $[\overline{a \mapsto \alpha}]\rho$ for fresh $\overline{\alpha}$, or ρ respectively). This is achieved with a function called *normalize* (a subcall of *generalize*), that accepts a constraint and a type and inlines/instantiates all constraint bounds if the type is a single variable. For example

  $$\begin{array}{rcl} \mathit{normalize}((\alpha_\star = \forall b\,.\,b \to b), \alpha) & = & [\beta_\star \bot] \Rightarrow \beta \to \beta \\ \mathit{normalize}((\alpha_\star \geq [\beta_\star \bot] \Rightarrow \beta), \alpha) & = & [\beta_\star \bot] \Rightarrow \beta \\ \mathit{normalize}((\alpha_\star = \sigma), \alpha \to \alpha) & = & [\alpha_\star = \sigma] \Rightarrow \alpha \to \alpha \end{array}$$

  With this step we have effectively performed an instantiation along $\preceq$. To push boxes down along $\sqsubseteq^{\to}$ we consider two cases: if the inlined and instantiated type was a function type we simply return it, otherwise we attempt to unify the inlined type with some function type $\beta \to \gamma$ consisting of fresh unification variables β and γ. We return the resulting type as $\sigma_1 \to \sigma_2$.
- Subsequently, we infer a type and an updated constraint for the argument e_2 with $E_3, \rho_3 = \mathit{infer}(E_2, \Gamma, e_2)$.
- At this point we need to compare the function argument type σ_1 to the type that we have inferred for the argument. However, we do not know the precise type of the argument at this point and hence we call $\mathit{generalize}(\Gamma, E_3, \rho_3)$ to get back a new constraint E_4 and a scheme ς_3. The scheme ς_3 expresses the set of *all possible types* of the argument e_2. We return to *generalize* later in this section.
- Now that we have a scheme ς_3 expressing all possible types of the argument e_2 we must check that the required type σ_1 belongs in the set that ς_3 expresses. This is achieved with the call to $\mathit{subsCheck}(E_4, \varsigma_3, \sigma_1)$, which simply returns an updated constraint E_5.
- Finally, type σ_2 may be equal to some type $\forall \overline{a}\,.\,\rho_2$, which we instantiate to $[\overline{a \mapsto \alpha}]\rho_2$ for fresh $\overline{\alpha}$. This is achieved with the call to $\mathit{inst}(E_5, \sigma_2)$, which implements the $\vdash^{\mathsf{inst}}$ judgement of the syntax-directed presentation.

$$
\begin{array}{lll}
 & \mathit{eqCheck} & : \quad \mathit{Constraint} \times \mathit{Type} \times \mathit{Type} \to \mathit{Constraint} \\
\textsc{e}1 & \mathit{eqCheck}(C, a, a) & = \quad C \\
\textsc{e}2 & \mathit{eqCheck}(C, \alpha, \sigma) & = \quad \mathit{updateRigid}(C, \alpha, \sigma) \\
\textsc{e}3 & \mathit{eqCheck}(C, \sigma, \alpha) & = \quad \mathit{updateRigid}(C, \alpha, \sigma) \\
\textsc{e}4 & \mathit{eqCheck}(C, \sigma_1 \to \sigma_2, \sigma_3 \to \sigma_4) & = \quad E = \mathit{eqCheck}(C, \sigma_1, \sigma_3) \\
 & & \quad \mathit{eqCheck}(E, \sigma_2, \sigma_4) \\
\textsc{e}5 & \mathit{eqCheck}(C, \forall \overline{a}\,.\,\rho_1, \forall \overline{a}\,.\,\rho_2) & = \quad E = \mathit{eqCheck}(C, [\overline{a \mapsto b}]\rho_1, [\overline{a \mapsto b}]\rho_2) \\
 & & \quad \text{if } \overline{b} \# E \text{ then return } E \text{ else } \mathit{fail} \\
\textsc{e}6 & \mathit{eqCheck}(C, \_, \_) & = \quad \mathit{fail} \\
 & & \\
 & \mathit{subsCheck} & : \quad \mathit{Constraint} \times \mathit{Scheme} \times \mathit{Types} \to \mathit{Constraint} \\
\textsc{s}1 & \mathit{subsCheck}(C, \varsigma, \beta) & = \quad \mathit{updateFlexi}(C, \beta, \varsigma) \\
\textsc{s}2 & \mathit{subsCheck}(C, [D] \Rightarrow \rho_1, \forall \overline{c}\,.\,\rho_2) & = \quad \rho_3 = [\overline{c \mapsto b}]\rho_2 \\
 & & \quad E = \mathit{eqCheck}(C \bullet D, \rho_1, \rho_3) \text{ and } \overline{\gamma} = \widehat{E}(\mathit{dom}(C)) \\
 & & \quad \text{if } \overline{b} \# E_{\overline{\gamma}} \text{ then return } E_{\overline{\gamma}} \text{ else } \mathit{fail} \\
 & & \\
\textsc{ur}1 & \mathit{updateRigid}(C, \alpha, \alpha) & = \quad \text{return } C \\
\textsc{ur}2 & \mathit{updateRigid}(C, \alpha, \beta) \mid ((\beta_\mu = \gamma) \in C) & = \quad \mathit{updateRigid}(C, \alpha, \gamma) \\
\textsc{ur}3 & \mathit{updateRigid}(C, \alpha, \beta) \mid & \\
 & \quad ((\beta_\mu \geq [D] \Rightarrow \gamma) \in C) \wedge (\gamma \notin \mathit{dom}(D)) & = \quad \mathit{updateRigid}(C, \alpha, \gamma) \\
\textsc{ur}4 & \mathit{updateRigid}(C, \alpha, \sigma) & = \quad \text{if } \alpha \in \widehat{C}(\sigma) \text{ then } \mathit{fail} \text{ else } \mathit{doUpdateR}(C, \alpha, \sigma) \\
\textsc{dr}1 & \mathit{doUpdateR}(C, \alpha, \sigma) \mid ((\alpha_{\mathsf{m}} \perp) \in C) & = \quad E = \mathit{mkMono}(\mathit{True}, C, \sigma) \\
 & & \quad \text{return } (E \leftarrow (\alpha_{\mathsf{m}} = \sigma)) \\
\textsc{dr}2 & \mathit{doUpdateR}(C, \alpha, \sigma) \mid ((\alpha_\star \perp) \in C) & = \quad \text{return } (C \leftarrow (\alpha_\star = \sigma)) \\
\textsc{dr}3 & \mathit{doUpdateR}(C, \alpha, \sigma) \mid ((\alpha_\mu \geq \varsigma_a) \in C) & = \quad E = \mathit{subsCheck}(C, \varsigma_a, \sigma) \\
 & & \quad \text{return } (E \leftarrow (\alpha_\mu = \sigma)) \\
\textsc{dr}4 & \mathit{doUpdateR}(C, \alpha, \sigma) \mid ((\alpha_\mu = \sigma_a) \in C) & = \quad \mathit{eqCheck}(C, \sigma_a, \sigma) \\
 & & \\
\textsc{uf}1 & \mathit{updateFlexi}(C, \alpha, [D] \Rightarrow \gamma) \mid \gamma \notin \mathit{dom}(D) & = \quad \mathit{updateRigid}(C, \alpha, \gamma) \\
\textsc{uf}2 & \mathit{updateFlexi}(C, \alpha, \varsigma) & = \quad \text{if } \alpha \in \widehat{C}(\varsigma) \text{ then } \mathit{fail} \text{ else } \mathit{doUpdateF}(C, \alpha, \varsigma) \\
\textsc{df}1 & \mathit{doUpdateF}(C, \alpha, [D] \Rightarrow \rho) \mid ((\alpha_{\mathsf{m}} \perp) \in C) & = \quad E = \mathit{mkMono}(\mathit{True}, C \bullet D, \rho) \\
 & & \quad \text{return } (E < -(\alpha_{\mathsf{m}} = \rho)) \\
\textsc{df}2 & \mathit{doUpdateF}(C, \alpha, \varsigma) \mid ((\alpha_\star \perp) \in C) & = \quad \text{return } (C < -(\alpha_\star \geq \varsigma)) \\
\textsc{df}3 & \mathit{doUpdateF}(C, \alpha, \varsigma) \mid ((\alpha_\mu = \sigma_a) \in C) & = \quad \mathit{subsCheck}(C, \varsigma, \sigma_a) \\
\textsc{df}4 & \mathit{doUpdateF}(C, \alpha, \varsigma) \mid ((\alpha_\mu \geq \varsigma_a) \in C) & = \quad E, \varsigma_r = \mathit{join}(C, \varsigma_a, \varsigma) \\
 & & \quad \text{return } (E \leftarrow (\alpha_\mu \geq \varsigma_r)) \\
 & & \\
 & \mathit{join} & : \quad \mathit{Constraint} \times \mathit{Scheme} \times \mathit{Scheme} \to \\
 & & \qquad \mathit{Constraint} \times \mathit{Scheme} \\
\textsc{j}1 & \mathit{join}(C, [D] \Rightarrow \alpha, \varsigma_2) \mid (\alpha_\mu \perp) \in D & = \quad (C, \varsigma_2) \\
\textsc{j}2 & \mathit{join}(C, \varsigma_1, [D] \Rightarrow \alpha) \mid (\alpha_\mu \perp) \in D & = \quad (C, \varsigma_1) \\
\textsc{j}3 & \mathit{join}(C, [D_1] \Rightarrow \rho_1, [D_2] \Rightarrow \rho_2) & = \quad E = \mathit{eqCheck}(C \bullet D_1 \bullet D_2, \rho_1, \rho_2) \\
 & & \quad \overline{\delta} = \widehat{E}(\rho_1) - \widehat{E}(\mathit{dom}(C)) \\
 & & \quad \text{return } (E_{\widehat{E}(\mathit{dom}(C))}, [E_{\overline{\delta}}] \Rightarrow \rho_1) \\
 & & \\
 & \mathit{mkMono} & : \quad \mathit{Bool} \times \mathit{Constraint} \times \mathit{Type} \to \mathit{Constraint} \\
\textsc{m}1 & \mathit{mkMono}(f, C, \forall \overline{a}\,.\,\rho) & = \quad \text{if } (f \wedge \overline{a} \neq \emptyset) \text{ then } \mathit{fail} \text{ else } \mathit{mkMono}(f, C, \rho) \\
\textsc{m}2 & \mathit{mkMono}(f, C, \sigma_1 \to \sigma_2) & = \quad E = \mathit{mkMono}(f, C, \sigma_1) \\
 & & \quad \mathit{mkMono}(f, E, \sigma_2) \\
\textsc{m}3 & \mathit{mkMono}(f, C, a) & = \quad C \\
\textsc{m}4 & \mathit{mkMono}(f, C, \alpha) \mid (\alpha_{\mathsf{m}}\ \mathit{bnd}) \in C & = \quad C \\
\textsc{m}5 & \mathit{mkMono}(f, C, \alpha) \mid (\alpha_\mu \geq [D] \Rightarrow \rho) \in C) & = \quad E = \mathit{mkMono}(\mathit{True}, C \bullet D, \rho) \\
 & & \quad \text{return } (E \leftarrow (\alpha_{\mathsf{m}} = \rho)) \\
\textsc{m}6 & \mathit{mkMono}(f, C, \alpha) \mid (\alpha_\mu = \sigma) \in C & = \quad E = \mathit{mkMono}(\mathit{True}, C, \sigma) \\
 & & \quad \text{return } (E \leftarrow (\alpha_{\mathsf{m}} = \sigma)) \\
\textsc{m}7 & \mathit{mkMono}(f, C, \alpha) \mid (\alpha_\star \perp) \in C & = \quad \text{return } (C \leftarrow (\alpha_{\mathsf{m}} \perp))
\end{array}
$$

Figure 10: Unification and instance checking

$$
\begin{array}{lll}
 & \mathit{inst} & : \quad \mathit{Constraint} \times \mathit{Type} \to \mathit{Constraint} \times \mathit{Type} \\
\textsc{inst} & \mathit{inst}(C, \forall \overline{a}\,.\,\rho) & = \quad \text{return } (C \bullet (\overline{\alpha}_\star\ \bot), [\overline{a \mapsto \alpha}]\rho) \\
 & & \\
 & \mathit{normalize} & : \quad \mathit{Constraint} \times \mathit{Type} \to \mathit{Scheme} \\
\textsc{n1} & \mathit{normalize}(C, \alpha) \mid (\alpha_\star \geq [D] \Rightarrow \rho) \in C & = \quad \text{return } ([C \bullet D - \alpha] \Rightarrow \rho) \\
\textsc{n2} & \mathit{normalize}(C, \alpha) \mid (\alpha_\star = \sigma) \in C & = \quad \mathit{normalize}(C - \alpha, \sigma) \\
\textsc{n3} & \mathit{normalize}(C, \forall \overline{a}\,.\,\rho) & = \quad \text{return } ([C \bullet (\overline{\alpha}.\ \bot)] \Rightarrow [\overline{a \mapsto \alpha}]\rho) \\
\textsc{n4} & \mathit{normalize}(C, \rho) & = \quad \text{return } ([C] \Rightarrow \rho) \\
 & & \\
\textsc{gen} & \mathit{generalize}(C, \Gamma, \rho) & = \quad \overline{\beta} = \widehat{C}(\rho) - \widehat{C}(\Gamma) \\
 & & \quad\ [E] \Rightarrow \rho_1 = \mathit{normalize}(C_{\overline{\beta}}, \rho) \\
 & & \quad\ \text{return } (C - \overline{\beta}, [E] \Rightarrow \rho_1) \\
 & & \\
 & \mathit{instFun} & : \quad \mathit{Constraint} \times \mathit{Env} \times \mathit{Type} \to \mathit{Constraint} \times \mathit{Type} \\
\textsc{if1} & \mathit{instFun}(C, \Gamma, \alpha) & = \quad E_1, ([E_g] \Rightarrow \rho_g) = \mathit{generalize}(C, \Gamma, \alpha) \\
 & & \quad\ E_2 = \mathit{eqCheck}(E_1 \bullet E_g \bullet (\beta_\star\ \bot) \bullet (\gamma_\star\ \bot), \rho_g, \beta \to \gamma) \\
 & & \quad\ \text{return } (E_2, \beta \to \gamma) \\
\textsc{if2} & \mathit{instFun}(C, \Gamma, \sigma_1 \to \sigma_2) & = \quad \text{return } (C, \sigma_1 \to \sigma_2) \\
\textsc{if3} & \mathit{instFun}(C, \Gamma, \_) & = \quad \mathit{fail} \\
 & & \\
 & \mathit{instMono} & : \quad \mathit{Constraint} \times \mathit{Env} \times \mathit{Type} \to \mathit{Constraint} \times \mathit{Type} \\
\textsc{im1} & \mathit{instMono}(C, \Gamma, \alpha) & = \quad E_1, ([E_g] \Rightarrow \rho_g) = \mathit{generalize}(C, \Gamma, \alpha) \\
 & & \quad\ E_2 = \mathit{mkMono}(E_1 \bullet E_g, \mathit{True}, \rho_g) \\
 & & \quad\ \text{return } (E_2, \rho_g) \\
\textsc{im2} & \mathit{instMono}(C, \Gamma, \rho) & = \quad E = \mathit{mkMono}(C, \mathit{False}, \rho)\ ;\ \text{return } (E, \rho) \\
 & & \\
 & \mathit{infer} & : \quad \mathit{Constraint} \times \mathit{Env} \times \mathit{Term} \to \mathit{Constraint} \times \mathit{Type} \\
\textsc{ivar} & \mathit{infer}(C, \Gamma, x) & = \quad \text{if } (x{:}\sigma) \in \Gamma \text{ then } \mathit{inst}(C, \sigma) \text{ else } \mathit{fail} \\
\textsc{iapp} & \mathit{infer}(C, \Gamma, e_1\ e_2) & = \quad E_1, \rho_1 = \mathit{infer}(C, \Gamma, e_1) \\
 & & \quad\ E_2, \sigma_1 \to \sigma_2 = \mathit{instFun}(E_1, \Gamma, \rho_1) \\
 & & \quad\ E_3, \rho_3 = \mathit{infer}(E_2, \Gamma, e_2) \\
 & & \quad\ E_4, \varsigma_3 = \mathit{generalize}(\Gamma, E_3, \rho_3) \\
 & & \quad\ E_5 = \mathit{subsCheck}(E_4, \varsigma_3, \sigma_1) \\
 & & \quad\ \mathit{inst}(E_5, \sigma_2) \\
\textsc{ilet} & \mathit{infer}(C, \Gamma, \mathtt{let}\ x = u\ \mathtt{in}\ e) & = \quad E_1, \rho_1 = \mathit{infer}(C, \Gamma, u) \\
 & & \quad\ E_2, \rho_2 = \mathit{instMono}(E_1, \Gamma, \rho_1) \\
 & & \quad\ \rho_3 = \mathit{zonkType}(E_2, \rho_2) \\
 & & \quad\ \overline{\alpha} = \widehat{E_2}(\rho_3) - \widehat{E_2}(\Gamma) \quad \overline{\beta} = \overline{\alpha} \cap \mathit{fcv}(\rho_3) \quad \overline{a} \text{ fresh} \\
 & & \quad\ \mathit{infer}(E_2 - E_{2\overline{\alpha}}, \Gamma, (x{:}\forall \overline{a}\,.\,[\overline{\beta \mapsto a}]\rho_3), e) \\
\textsc{iann} & \mathit{infer}(C, \Gamma, (e\mathtt{::}\sigma)) & = \quad E_1, \rho = \mathit{infer}(C, \Gamma, e) \\
 & & \quad\ E_2, \varsigma = \mathit{generalize}(E_1, \Gamma, \rho) \\
 & & \quad\ E_3 = \mathit{subsCheck}(E_2, \varsigma, \sigma) \\
 & & \quad\ \mathit{inst}(E_3, \sigma) \\
\textsc{iabs} & \mathit{infer}(C, \Gamma, \lambda x\,.\,e) & = \quad E_1, \rho = \mathit{infer}(C \bullet (\beta_{\mathsf{m}}\ \bot), (\Gamma, (x{:}\beta)), e) \\
 & & \quad\ E_2, \rho_1 = \mathit{instMono}(E_1, (\Gamma, (x{:}\beta)), \rho) \\
 & & \quad\ \overline{\alpha} = \widehat{E_2}(\beta \to \rho_1) - \widehat{E_2}(\Gamma) \\
 & & \quad\ E^{\uparrow}_{2\overline{\alpha}} = \{(\alpha_\star\ \mathit{bnd}) \mid (\alpha_\mu\ \mathit{bnd}) \in E_{2\overline{\alpha}}\} \\
 & & \quad\ \text{return } ((E_2 - E_{2\overline{\alpha}}) \bullet E^{\uparrow}_{2\overline{\alpha}}, (\beta \to \rho_1)) \\
 & & \\
\textsc{z1} & \mathit{zonkType}(C, \alpha) \mid (\alpha_\mu = \sigma) \in C & = \quad \mathit{zonkType}(\sigma) \\
\textsc{z2} & \mathit{zonkType}(C, \alpha) \mid (\alpha_\mu\ \bot) \in C & = \quad \alpha \\
\textsc{z3} & \mathit{zonkType}(C, a) & = \quad a \\
\textsc{z4} & \mathit{zonkType}(C, \sigma_1 \to \sigma_2) & = \quad \mathit{zonkType}(C, \sigma_1) \to \mathit{zonkType}(C, \sigma_2) \\
\textsc{z5} & \mathit{zonkType}(C, \forall \overline{a}\,.\,\rho) & = \quad \forall \overline{a}\,.\,\mathit{zonkType}(C, \rho)
\end{array}
$$

Figure 11: Main inference algorithm

We now turn to *generalize* with the following signature:

$$generalize : Env \times Constraint \times Type \rightarrow Constraint \times Scheme$$

The *generalize* function accepts an evnironment, a constraint, and a ρ type which is typically the result of a prior call to *infer*, and returns a scheme which is well-formed and expresses the set of all possible System F types that can be generalized from ρ. The definition of *generalize* is:

$$\begin{array}{rcl} generalize(\Gamma, C, \rho) & = & \overline{\beta} = \widehat{C}(\rho) - \widehat{C}(\Gamma) \\ & & \varsigma = normalize(C_{\overline{\beta}}, \rho) \\ & & \text{return } (C - \overline{\beta}, \varsigma) \end{array}$$

The first step is to locate which variables can be generalized. These are the $\overline{\beta}$ which are the reachable variables of ρ through C, except for those that are reachable from the environment Γ through C. Notice that if the reachable variables of Γ through the constraint C were monomorphic this implies that there are *no monomorphic variables* in $\overline{\beta}$. Hence we could at this point return $[C_{\overline{\beta}}]\Rightarrow\rho$ ($C_{\overline{\beta}}$ is the restriction of C to $\overline{\beta}$) and condition (b) of rule CWF would not be violated. However condition (c) could be violated, and hence we must *normalize* ρ in order to inline any polymorphism of ρ, if ρ is a plain variable. After the call to *normalize*, the returned scheme ς is a normal scheme that satisfies condition (c). Finally, in the returned constraint we can safely discard all the bounds of $\overline{\beta}$.

A different interesting part of the function *infer* is related to forcing monomorphism of the constrained variables of `let`-bound expressions, or abstraction bodies. Intuitively, after inferring a type for a `let`-bound expression we need to implement the instantiation along $\preceq\!\!\sqsubseteq$ to a box-free type. In much the same way as the call to *instFun* we normalize the inferred type if it is a single variable and subsequently traverse the inlined type flagging variables as monomorphic and ensuring that all flexible bounds are instantiated and all variables are `m`-flagged and equal to quantifier-free types. This is achieved with the *instMono* function. The generalization step proceeds then just like generalization in a traditional Hindley-Damas-Milner implementation.

Finally, function abstractions are first typed by creating a fresh `m`-flagged variable for the argument type, inferring a type for the body, forcing it to be monomorphic and lifting the `m` flags from the variables that could be generalized.

Importance of normal schemes The case for applications illustrates why it is important for schemes to be normal. Consider the following code fragment:

```
head :: forall d. [d] -> d
foo :: (forall b. b -> b) -> Int

h = foo (head id)
```

The application is certainly typeable in the specification. First we instantiate the type of `head` with $\boxed{[\forall b\,.\,b \rightarrow b]} \rightarrow \boxed{\forall b\,.\,b \rightarrow b}$ and the application `head id` can be typed with $\boxed{\forall b\,.\,b \rightarrow b}$. Subsequently it is straightforward to check that $\vdash^{\mathsf{F}} \forall b\,.\,b \rightarrow b \leq \forall b\,.\,b \rightarrow b$ as rule SDAPP requires. However let us consider the algorithm operation. The type that will be inferred for `head alls` will simply be γ where $(\gamma_\star = \forall b\,.\,b \rightarrow b)$ is bound in the constraint. The non-normalized scheme is $[\gamma_\star = \forall b\,.\,b \rightarrow b]\Rightarrow\gamma$ whereas the normalized one would be $[\beta_\star\ \bot]\Rightarrow\beta \rightarrow \beta$. But now notice that according to Definition 6.3 it *is not* the case that $(\forall b\,.\,b \rightarrow b) \in [\![[\gamma_\star = \forall b\,.\,b \rightarrow b]\Rightarrow\gamma]\!]$. However, with the normal scheme, $[\beta_\star\ \bot]\Rightarrow\beta \rightarrow \beta$, *it is* the case that $(\forall b\,.\,b \rightarrow b) \in [\![[\beta_\star\ \bot]\Rightarrow\beta \rightarrow \beta]\!]$.

In short, non-normal schemes, when interpreted with Definition 6.3, do not capture all of the desired System F types. Either the interpretation should be different, or we should be using normal schemes. We chose to use normal schemes.

6.2.3 Instance checking and unification

The main type inference algorithm relies on the *subsCheck* function, which checks whether a type belongs in the interpretation of a scheme. The *subsCheck* function is mutually defined with several other functions, that together constitute our unification algorithm. The signatures of these functions are given below:

$$\begin{array}{lcl} \mathit{subsCheck} & : & \mathit{Constraint} \times \mathit{Scheme} \times \mathit{Type} \rightarrow \mathit{Constraint} \\ \mathit{eqCheck} & : & \mathit{Constraint} \times \mathit{Type} \times \mathit{Type} \rightarrow \mathit{Constraint} \\ \mathit{updateRigid} & : & \mathit{Constraint} \times \mathit{Var} \times \mathit{Type} \rightarrow \mathit{Constraint} \\ \mathit{updateFlexi} & : & \mathit{Constraint} \times \mathit{Var} \times \mathit{Scheme} \rightarrow \mathit{Constraint} \\ \mathit{join} & : & \mathit{Constraint} \times \mathit{Scheme} \times \mathit{Scheme} \rightarrow \mathit{Constraint} \times \mathit{Scheme} \end{array}$$

With the call to $\mathit{subsCheck}(C_1, \varsigma, \sigma) = C_2$ we check that σ belongs in the interpretation of the scheme ς and produce an updated constraint C_2. With $\mathit{eqCheck}(C_1, \sigma_1, \sigma_2) = C_2$ we check that σ_1 can be equated to σ_2 and produce an updated constraint C_2. The call to $\mathit{updateRigid}(C_1, \alpha, \sigma) = C_2$ updates the bounds of α in C_1 to be rigidly equal to σ. The call to $\mathit{updateFlexi}(C_1, \alpha, \varsigma) = C_2$ updates the bounds of α to ensure that any instantiation of α will belong in the interpretation of ς. Finally the call $\mathit{join}(C_1, \varsigma_1, \varsigma_2) = C_2, \varsigma$ updates the constraint and produces a new well-formed scheme whose interpretation is the *intersection* of the interpretations of ς_1 and ς_2. The *join* function is the heart of the algorithmic implementation.

6.2.4 Summary of the algorithmic implementation properties

The algorithm satisfies several properties. In this section we outline the most important propositions, and we proceed with their detailed proofs in the next section.

The first observation is that the functions that constitute our instance checking and unification terminate; and as an easy corollary the overall algorithm terminates.

Proposition (see Theorem 6.28): Assume that $\vdash C_1$ and $\Delta_C \vdash \varsigma$ and $\Delta_C \vdash \sigma$. Then the call to $\mathit{subsCheck}(C_1, \varsigma, \sigma)$ either returns *fail* or terminates and returns a constraint C_2.

Unsurprisingly, the proof of termination is very similar to the termination proof for the ML^F unification algorithm. It involves a metric that is a lexicographic triple whose first component is the sum of the quantifier ranks of the argument C_1 and the constraint of the scheme ς_1, its second component involves the sizes of ς and σ, and the third component involves the sizes of the sets of reachable variables of ς and σ through C_1.

The following proposition states soundness of instance checking and unification.

Proposition (see Section 6.3.2): Assume in all cases that $\vdash C_1$ and all the schemes and types below are well-formed in Δ_{C_1}. Then, the following are true:

1. If $\mathit{eqCheck}(C_1, \sigma_1, \sigma_2) = C_2$ and $\theta \models C_2$ then $\theta \models C_1$ and moreover $\theta\sigma_1 = \theta\sigma_2$.
2. If $\mathit{subsCheck}(C_1, \varsigma, \sigma) = C_2$ and $\theta \models C_2$ then $\theta \models C_1$ and moreover $\theta\sigma \in [\![\theta\varsigma]\!]$.
3. If $\mathit{updateRigid}(C_1, \alpha, \sigma) = C_2$ and $\theta \models C_2$ then $\theta \models C_1$ and moreover $\theta\alpha = \theta\sigma$.
4. If $\mathit{updateRigid}(C_1, \alpha, \varsigma) = C_2$ and $\theta \models C_2$ then $\theta \models C_1$ and moreover $\theta\alpha \in [\![\theta\varsigma]\!]$.
5. If $\mathit{join}(C_1, \varsigma_1, \varsigma_2) = C_2, \varsigma$ and $\theta \models C_2$ then $\theta \models C_1$ and moreover $[\![\theta\varsigma]\!] \subseteq [\![\theta\varsigma_1]\!] \cap [\![\theta\varsigma_2]\!]$.

Completeness of unification with respect to the set semantics is true, as the following proposition states.

Proposition (see Section 6.3.3): Assume in all cases that $\vdash C_1$ and all the schemes and types are well formed in Δ_{C_1} and that $\mathit{dom}(\theta) = \mathit{dom}(C_1)$. Then, the following are true:

1. If $\theta\sigma_1 = \theta\sigma_2$ then $\mathit{eqCheck}(C_1, \sigma_1, \sigma_2) = C_2$ and there exists a θ_r such that $\theta\theta_r \models C_2$.
2. If $\theta\sigma \in [\![\theta\varsigma]\!]$ then $\mathit{subsCheck}(C_1, \varsigma, \sigma) = C_2$ and there exists a θ_r such that $\theta\theta_r \models C_2$.

3. If $\theta\alpha = \theta\sigma$ then $updateRigid(C_1, \alpha, \sigma) = C_2$ and there exists a θ_r such that $\theta\theta_r \models C_2$.
4. If $\theta\alpha \in [\![\theta\varsigma]\!]$ then $updateFlexi(C_1, \alpha, \varsigma) = C_2$ and there exists a θ_r such that $\theta\theta_r \models C_2$.
5. If $\sigma \in [\![\theta\varsigma_1]\!] \cap [\![\theta\varsigma_2]\!]$ then $join(C_1, \varsigma_1, \varsigma_2) = C_2, \varsigma$ and there exists a θ_r such that $\theta\theta_r \models C_2$ and $\sigma \in [\![\theta\theta_r\varsigma]\!]$.

It is particularly illuminating to observe the unification soundness and completeness for the *join* function. Soundness and completeness of unification show that *join* computes precisely a scheme that expresses the intersection of the interpretation of the argument schemes.

We now turn our focus to the main inference function. In order to state the soundness and completeness properties of the algorithm we have to define the notion of a boxy substitution.

Definition 6.4 (Boxy substitution)**.** Given a substitution θ from constrained variables to constrained-variable-free types, we define the boxy substitution of θ on σ, denoted with $\theta[\sigma]$, that substitutes the range of θ in a boxed and capture-avoiding fashion inside σ.

For example, if $\theta = [\alpha \mapsto \sigma]$ then $\theta[\alpha \to \alpha] = \boxed{\sigma} \to \boxed{\sigma}$. We may use boxy substitutions to recover specification types from algorithmic types, provided that all their constrained variables appear in the domains of the substitutions. Additionally we write $C \vdash \Gamma$ when for all $(x{:}\sigma) \in \Gamma$ it is the case that $\Delta_C \vdash \sigma$, and additionally it is the case that $\alpha \in \widehat{C}(\Gamma)$ iff $\Delta_C(\alpha) = \mathsf{m}$. Intuitively, the notation $C \vdash \Gamma$ means that all variables of Γ are in C and monomorphically flagged, and the reachable variables from Γ are the only m-flagged variables in C. We are now ready to state soundness:

Proposition (see Lemma 6.70): If $\vdash C_1$ and $C_1 \vdash \Gamma$ and $infer(C_1, \Gamma, e) = C_2, \rho$ then for all $\theta \models C_2$ it is the case that $\theta \models C_1$ and $\theta\Gamma \vdash^{\mathsf{sd}} e : \theta[\rho]$.

Notice that we need not apply θ in a boxy fashion to Γ, since all the variables of Γ will be monomorphic in C_1, and θ can only map them to monomorphic types. A corollary is soundness with respect to the declarative specification:

Corollary 6.5. *If* $\vdash C_1$ *and* $C_1 \vdash \Gamma$ *and* $infer(C_1, \Gamma, e) = C_2, \rho$ *then for all* $\theta \models C_2$ *it is the case that* $\theta \models C_1$ *and* $\theta\Gamma \vdash e : \theta[\rho]$.

We are now ready to state the main completeness proposition.

Proposition (see Lemma 6.77): If $\vdash C_1$ and $C_1 \vdash \Gamma$ and $\theta \models C_1$ and $dom(\theta) = dom(C_1)$ and $\theta\Gamma \vdash^{\mathsf{sd}} e : \rho'$ then $infer(C_1, \Gamma, e) = C_2, \rho$ and there exists a θ_r such that $\theta\theta_r \models C_2$ and $\theta[\rho] \preceq\sqsubseteq \rho'$.

The proof is an induction on the size of e in the derivation of $\theta\Gamma \vdash^{\mathsf{sd}} e : \rho'$. The cases for `let`-bound expressions and abstractions rely on Lemma 5.7 and a Damas-Milner strengthening property (can be found in the main paper), and are similar to the corresponding cases for ordinary Hindley-Damas-Milner inference. However the cases for applications and annotations crucially rely the two propositions below:

Proposition 6.6. *If* $\vdash \varsigma$, $\sigma_1 \in [\![\varsigma]\!]$ *and* $\vdash^{\mathsf{F}} \sigma_1 \leq \sigma_2$ *then* $\sigma_2 \in [\![\varsigma]\!]$.

Proposition (See Lemma 6.75): Assume that $\vdash C_1$ and $C_1 \vdash \Gamma$ and $C_1 \vdash \rho$ and $\theta \models C_1$ and $ftv(\rho) = \emptyset$ and $E, \varsigma = generalize(C_1, \Gamma, \rho)$ and $\overline{a} = ftv(\theta\rho) - ftv(\theta\Gamma)$. Then $\lfloor \forall \overline{a}\,.\,\theta[\rho] \rfloor \in [\![\theta\varsigma]\!]$.

Notice that the last proposition can be viewed as a generalization of Lemma 5.7 that involves schemes. The following corollary is then true, by observing that $\preceq\sqsubseteq$ is transitive:

Corollary 6.7. *If* $\vdash C_1$ *and* $C_1 \vdash \Gamma$ *and* $\theta \models C_1$ *and* $dom(\theta) = dom(C_1)$ *and* $\theta\Gamma \vdash e : \rho'$ *then* $infer(C_1, \Gamma, e) = C_2, \rho$ *and there exists a* θ_r *such that* $\theta\theta_r \models C_2$ *and* $\theta[\rho] \preceq\sqsubseteq \rho'$.

Together Corollary 6.5 and Corollary 6.7 ensure the implementability of the declarative specification of Figure 2.

6.3 Detailed algorithm metatheory

We present in this section the detailed metatheory of the algorithmic implementation. We first need to extend the quantifier rank definition to schemes and types.

Definition 6.8 (Scheme quantifier rank)**.** If $\varsigma = [D] \Rightarrow \rho$, we let $q(\varsigma) = q(D)$. Overloading the notation we let $q(\sigma) = 0$.

6.3.1 Unification termination

Definition 6.9 (Acyclicity)**.** A constraint C is acyclic, written $\texttt{acyclic}(C)$ for the smallest relation that asserts that, for all $(\alpha_m \in bnd) \in C$, $\alpha \notin \widehat{C}(bnd)$; and moreover for every $(\alpha_\mu \geq [D] \Rightarrow \rho) \in C$ it is $\texttt{acyclic}(D)$.
Definition 6.10 (Closedness)**.** A constraint C is closed, written $\texttt{closed}(C)$ if $fcv(C) \subseteq dom(C)$.
Lemma 6.11 (*mkMono* domain monotonicity)**.** *If* $mkMono(C_1, f, \sigma) = C_2$ *then* $dom(C_1) \subseteq dom(C_2)$.

Proof. Easy induction on the number of recursive calls to *mkMono*. □

Lemma 6.12 (Domain monotonicity)**.** *The following are true:*

1. *If* $eqCheck(C_1, \sigma_1, \sigma_2) = C_2$ *then* $dom(C_1) \subseteq dom(C_2)$.
2. *If* $eqCheck(C_1, \sigma_1, \sigma_2) = C_2$ *then* $dom(C_1) \subseteq dom(C_2)$.
3. *If* $subsCheck(C_1, \varsigma, \sigma) = C_2$ *then* $dom(C_1) \subseteq dom(C_2)$.
4. *If* $updateRigid(C_1, \alpha, \sigma) = C_2$ *then* $dom(C_1) \subseteq dom(C_2)$.
5. *If* $doUpdateR(C_1, \alpha, \sigma) = C_2$ *then* $dom(C_1) \subseteq dom(C_2)$.
6. *If* $updateFlexi(C_1, \alpha, \varsigma) = C_2$ *then* $dom(C_1) \subseteq dom(C_2)$.
7. *If* $doUpdateF(C_1, \alpha, \varsigma) = C_2$ *then* $dom(C_1) \subseteq dom(C_2)$.
8. *If* $join(C_1, \varsigma_1, \varsigma_2) = C_2, \varsigma_r$ *then* $dom(C_1) \subseteq dom(C_2)$.

Proof. The proof is by simultaneous induction, appealing also to Lemma 6.11. For each case we assume that the property is true for all other cases when they terminate in a smaller number of recursive calls. □

The following lemma asserts that m-flagged variables never have their flags lifted during unification.
Lemma 6.13. *If* $\Delta_{C_1}(\beta) = \mathsf{m}$ *and* $mkMono(C_1, f, \sigma) = C_2$ *then* $\Delta_{C_2}(\beta) = \mathsf{m}$.

Proof. Easy induction on the number of recursive calls to *mkMono*. □

Lemma 6.14 (Monomorphic domain monotonicity)**.** *The following are true:*

1. *If* $\Delta_{C_1}(\beta) = \mathsf{m}$ *and* $eqCheck(C_1, \sigma_1, \sigma_2) = C_2$ *then* $\Delta_{C_2}(\beta) = \mathsf{m}$.
2. *If* $\Delta_{C_1}(\beta) = \mathsf{m}$ *and* $subsCheck(C_1, \varsigma, \sigma) = C_2$ *then* $\Delta_{C_2}(\beta) = \mathsf{m}$.
3. *If* $\Delta_{C_1}(\beta) = \mathsf{m}$ *and* $updateRigid(C_1, \alpha, \sigma) = C_2$ *then* $\Delta_{C_2}(\beta) = \mathsf{m}$.
4. *If* $\Delta_{C_1}(\beta) = \mathsf{m}$ *and* $doUpdateR(C_1, \alpha, \sigma) = C_2$ *then* $\Delta_{C_2}(\beta) = \mathsf{m}$.
5. *If* $\Delta_{C_1}(\beta) = \mathsf{m}$ *and* $updateFlexi(C_1, \alpha, \varsigma) = C_2$ *then* $\Delta_{C_2}(\beta) = \mathsf{m}$.
6. *If* $\Delta_{C_1}(\beta) = \mathsf{m}$ *and* $doUpdateF(C_1, \alpha, \varsigma) = C_2$ *then* $\Delta_{C_2}(\beta) = \mathsf{m}$.
7. *If* $\Delta_{C_1}(\beta) = \mathsf{m}$ *and* $join(C_1, \varsigma_1, \varsigma_2) = C_2, \varsigma_r$ *then* $\Delta_{C_2}(\beta) = \mathsf{m}$.

Proof. Easy simultaneous induction, using Lemma 6.12 and Lemma 6.13 and observing that we never lift a m flag during unification; instead we merely convert $\star$ flags to m. □

Lemma 6.15 (*mkMono* preserves closedness)**.** *If* $\texttt{closed}(C_1)$, $C_1 \vdash \sigma$, *and* $mkMono(C_1, f, \sigma) = C_2$ *then* $\texttt{closed}(C_2)$.

Proof. Induction on the number of recursive calls to *mkMono*. Case M1 follows by induction hypothesis. Case M2 follows by two uses of the inductive hypothesis and Lemma 6.11. Cases M3 and M4 are trivial. For case M5 we have that $\texttt{closed}(C_1)$ and $fcv([D] \Rightarrow \rho) \subseteq dom(C_1)$; hence $\texttt{closed}(C_1 \bullet D)$ and $fcv(\rho) \subseteq dom(C_1 \bullet D)$. By induction hypothesis $\texttt{closed}(E)$. To finish the case we need to show that $fcv(\rho) \subseteq dom(E \leftarrow (\alpha_{\mathsf{m}} = \rho))$. By Lemma 6.11 we know that $fcv(\rho) \subseteq dom(E)$ and also that $\alpha \in dom(E)$, hence $fcv(\rho) \subseteq dom(E \leftarrow (\alpha_{\mathsf{m}} = \rho))$. The case of M6 is similar, and M7 is trivial. □

Lemma 6.16 (Restriction preserves closedness)**.** *If* $\texttt{closed}(C)$ *and* $\overline{\gamma} = \widehat{C}(\overline{\beta})$ *then* $\texttt{closed}(C_{\overline{\gamma}})$.

Proof. Assume by contradiction that there exists a $\gamma \in bnd$ and $(\alpha_\mu\ bnd) \in C_{\overline{\gamma}}$ such that $\gamma \notin dom(C_{\overline{\gamma}})$. It follows that $(\alpha_\mu\ bnd) \in C$ and it also follows that $\alpha \in \widehat{C}(\overline{\beta})$. But in this case it must also be that $\gamma \in \widehat{C}(\overline{\beta})$, and hence $\gamma \in dom(C_{\overline{\gamma}})$—a contradiction. □

Theorem 6.17 (Unification preserves closedness)**.** *The following are true*

1. $\texttt{closed}(C_1) \wedge fcv(\sigma_1, \sigma_2) \subseteq dom(C_1) \wedge eqCheck(C_1, \sigma_1, \sigma_2) = C_2 \Longrightarrow \texttt{closed}(C_2)$
2. $\texttt{closed}(C_1) \wedge fcv(\varsigma, \sigma, \sigma_0) \subseteq dom(C_1) \wedge subsCheck(C_1, \varsigma, \sigma, \sigma_0) = C_2 \Longrightarrow \texttt{closed}(C_2)$
3. $\texttt{closed}(C_1) \wedge fcv(\alpha, \sigma) \subseteq dom(C_1) \wedge updateRigid(C_1, \alpha, \sigma) = C_2 \Longrightarrow \texttt{closed}(C_2)$
4. $\texttt{closed}(C_1) \wedge fcv(\alpha, \sigma) \subseteq dom(C_1) \wedge doUpdateR(C_1, \alpha, \sigma) = C_2 \Longrightarrow \texttt{closed}(C_2)$
5. $\texttt{closed}(C_1) \wedge fcv(\alpha, \sigma) \subseteq dom(C_1) \wedge updateFlexi(C_1, \alpha, \varsigma) = C_2 \Longrightarrow \texttt{closed}(C_2)$
6. $\texttt{closed}(C_1) \wedge fcv(\alpha, \varsigma) \subseteq dom(C_1) \wedge doUpdateF(C_1, \alpha, \varsigma) = C_2 \Longrightarrow \texttt{closed}(C_2)$
7. $\texttt{closed}(C_1) \wedge fcv(\varsigma_1, \varsigma_2) \subseteq dom(C_1) \wedge join(C_1, \varsigma_1, \varsigma_2) = C_2, \varsigma \Longrightarrow \texttt{closed}(C_2) \wedge fcv(\varsigma) \subseteq dom(C_2)$

Proof. We prove the cases simultaneously by induction on the number of recursive calls. For each case we assume that all hold for calls that terminate in a smaller number of recursive calls.

1. Case E1 is trivial. Case E2 follows by induction hypothesis for *updateRigid*, and similarly case E3. For E4 we have that $\texttt{closed}(C_1)$ and $fcv(\sigma_1, \sigma_2, \sigma_3, \sigma_4) \subseteq dom(C_1)$. Moreover $eqCheck(C_1, \sigma_1 \to \sigma_2, \sigma_3 \to \sigma_4) = C_2$ where $E = eqCheck(C_1, \sigma_1, \sigma_3)$ and $C_2 = eqCheck(E, \sigma_2, \sigma_4)$. By induction hypothesis we get that $\texttt{closed}(E)$. By Lemma 6.12 we get that $fcv(\sigma_2, \sigma_4) \subseteq dom(E)$. Consequently, by induction hypothesis we get that $\texttt{closed}(C_2)$, as required. For case E5 we have that $\texttt{closed}(C_1)$ and $fcv(\forall \overline{a}.\rho_1, \forall \overline{a}.\rho_2) \subseteq dom(C_1)$. Consequently also $fcv([\overline{a \mapsto b}]\rho_1, [\overline{a \mapsto b}]\rho_2) \subseteq dom(C_1)$, and by induction hypothesis $\texttt{closed}(E)$, where $C_2 = E$. Case E6 cannot happen.
2. Case S1 follows by induction hypothesis for *updateFlexi*. For case S2 we have that $\texttt{closed}(C_1)$ and $fcv([D] \Rightarrow \rho_1, \forall \overline{c}.\rho_2) \subseteq dom(C_1)$. It follows that $fcv(\rho_1, \rho_2) \subseteq dom(C_1 \bullet D)$ and moreover we get that $\texttt{closed}(C_1 \bullet D)$. By induction hypothesis then $\texttt{closed}(E)$. Because $\overline{\gamma} = \widehat{E}(dom(C_1))$, it must also be $\texttt{closed}(E_{\overline{\gamma}})$, by Lemma 6.16.
3. Case UR1 is trivial. For UR2, we show that $fcv(\alpha, \gamma) \subseteq dom(C_1)$. But we know that $fcv(\alpha, \beta) \subseteq dom(C_1)$, and moreover $\texttt{closed}(C_1)$. Since $(\beta_\mu = \gamma) \in C_1$, it must be that $\gamma \in dom(C_1)$. We can then apply the induction hypothesis to get that $\texttt{closed}(C_2)$. The case for UR3 is similar. Case UR4 follows by induction hypothesis for *doUpdateR*.
4. For case DR1 we know that $fcv(\alpha, \sigma) \in dom(C_1)$ and $\texttt{closed}(C_1)$. It follows that by Lemma 6.15 that $\texttt{closed}(E)$. To finish the case it is enough to show that $fcv(\sigma) \subseteq dom(E - \alpha)$. We know that $\alpha \notin fcv(\sigma)$ and hence it suffices to show that $fcv(\sigma) \subseteq dom(E)$. But that follows by Lemma 6.12. Case DR2 is similar. For the case of DR3 we know by induction hypothesis that $\texttt{closed}(E)$. Moreover to finish the case we need to show that $fcv(\sigma) \subseteq dom(E \leftarrow (\alpha_m = \sigma))$. But we know that $fcv(\sigma) \subseteq dom(E)$ by Lemma 6.12 and also $\alpha \in dom(E)$, and we are done. Case DR4. We know that $\texttt{closed}(C_1)$ and $fcv(\alpha, \sigma) \subseteq dom(C_1)$, hence also $fcv(\sigma_\alpha) \subseteq dom(C_1)$. Applying the induction hypothesis for *eqCheck* finishes the case.
5. For case UF1 we know that $\texttt{closed}(C_1)$ and $fcv(\alpha, [D] \Rightarrow \gamma) \subseteq dom(C_1)$—it follows that $fcv(\alpha, \gamma) \subseteq dom(C_1)$ and induction hypothesis for *updateRigid* finishes the case. For case UF2 appealing to induction hypothesis for *doUpdateF* shows the goal.
6. Cases DF1, DF2, DF3 are similar to the corresponding cases of *doUpdateR*. For DF4 we know that $\texttt{closed}(C_1)$ and $fcv(\alpha, \varsigma) \subseteq dom(C_1)$, hence also $fcv(\varsigma_a, \varsigma) \subseteq dom(C_1)$. By induction hypothesis then for *join* we get that $\texttt{closed}(E)$ and $fcv(\varsigma_r) \subseteq dom(E)$, hence also $fcv(\varsigma_r) \subseteq dom(E \leftarrow (\alpha_\mu \geq \varsigma_r))$.
7. Case J1 follows trivially, and case J2 is similar. For J3 we have that $\texttt{closed}(C_1)$ and $fcv([D_1] \Rightarrow \rho_1, [D_2] \Rightarrow \rho_2) \subseteq dom(C_1)$. It follows that $\texttt{closed}(C_1 \bullet D_1 \bullet D_2)$ and $fcv(\rho_1, \rho_2) \subseteq dom(C_1 \bullet D_1 \bullet D_2)$. By induction E is closed, and $E_{\widehat{E}(dom(C_1))}$ also closed (by Lemma 6.16). To finish the case we need to show that $fcv([E_{\overline{\delta}}] \Rightarrow \rho_1) \subseteq dom(E_{\widehat{E}(dom(C_1))})$. Pick then a variable $\alpha \in fcv([E_{\overline{\delta}}] \Rightarrow \rho_1)$. It must be that $\alpha \notin (\widehat{E}(\rho_1) - \widehat{E}(dom(C_1)))$. Hence we have two cases.

If $\alpha \in \widehat{E}(\rho_1)$ then it must be that $\alpha \in \widehat{E}(dom(C_1))$, hence $\alpha \in dom(E_{\widehat{E}(dom(C_1))})$, because it must be that $\alpha \in dom(E)$ (by domain monotonicity). On the other hand, if $\alpha \notin \widehat{E}(\rho_1)$ then $\alpha \notin fcv([E_{\overline{\delta}}] \Rightarrow \rho_1)$, a contradiction.

□

Lemma 6.18 (*mkMono* single-variable separation)**.** *If* $\alpha \notin \widehat{C_1}(\sigma, \sigma_0)$ *and* $\alpha \in dom(C_1)$*, and* $mkMono(C_1, f, \sigma) = C_2$*, then* $mkMono(C_1 - \alpha, f, \sigma) = C_2 - \alpha$ *in the same number of steps, and* $\alpha \notin \widehat{C_2}(\sigma, \sigma_0)$*.*

Proof. By induction on the number of steps that *mkMono* performs. Cases M1, M3, and M4 are straightforward. For case M2 we have that $\alpha \notin \widehat{C_1}(\sigma_1 \to \sigma_2, \sigma_0)$ and $mkMono(C_1, f, \sigma_1 \to \sigma_2) = C_2$ where $E = mkMono(C_1, f, \sigma_1)$ and $C_2 = mkMono(E, f, \sigma_2)$. By induction hypothesis $E - \alpha = mkMono(C_1 - \alpha, f, \sigma_1)$ and moreover $\alpha \notin \widehat{E}(\sigma_1, \sigma_2, \sigma_0)$. By Lemma 6.11 we know that $\alpha \in dom(E)$. Hence, by induction hypothesis $C_2 - \alpha = mkMono(E - \alpha, f, \sigma_2)$ and $\alpha \notin \widehat{C_2}(\sigma_2, \sigma_1, \sigma_0)$. That is, $\alpha \notin \widehat{C_2}(\sigma_1 \to \sigma_2, \sigma_0)$. For case M5, we have that $\beta \notin \widehat{C_1}(\alpha, \sigma_0)$ and $mkMono(C_1, f, \alpha) = (E \leftarrow (\alpha_m = \rho))$ whenever $E = mkMono(C_1 \bullet D, \rho)$ and $(\alpha_\mu \geq [D] \Rightarrow \rho) \in C_1$. Then we know that $\beta \notin \widehat{C_1 \bullet D}(\rho) - dom(D)$ but also that $\beta \in dom(C_1)$; hence it is not in the domain of D and consequently, $\beta \notin \widehat{C_1 \bullet D}(\rho)$. By induction hypothesis then $(E - \beta) = mkMono((C_1 - \beta) \bullet D, \rho)$ and $\beta \notin \widehat{E}(\rho, \sigma_0)$. Consequently $mkMono(C_1 - \beta, f, \alpha) = (E - \beta) \leftarrow (\alpha_{\mathsf{m}} = \rho)$ as required. Case M6 is similar to case M5. Case M7 is an easy check. □

Lemma 6.19. *If* $\texttt{acyclic}(C)$ *and* $\texttt{acyclic}(D)$*, then* $\texttt{acyclic}(C \bullet D)$*.*

Proof. Easy check. □

Below, we use the symbol $\rightsquigarrow_E$ for the *reachability relation* induced by the bounds of a constraint E. Each edge in this graph corresponds to a set of edges between a bound variable α and the $fcv(bnd)$, in any constraint of the form $(\alpha_\mu\ bnd) \in E$.

Lemma 6.20 (Monomorphization preserves acyclicity)**.** *If* $\texttt{acyclic}(C_1)$ *and* $mkMono(C_1, f, \sigma) = C_2$ *then* $\texttt{acyclic}(C_2)$*.*

Proof. By induction on the number of recursive calls to *mkMono*. Case M1 is trivial, and M2 follows by two uses of the induction hypothesis. Cases M3 and M4 are trivial. For M5, we first know that $mkMono(C_1 \bullet D, f, \rho) = E$. The constraint $C_1 \bullet D$ must then be acyclic as well, and by induction $\texttt{acyclic}(E)$. We want to show now that $\texttt{acyclic}(E \leftarrow (\alpha_{\mathsf{m}} = \rho))$. However we know that $\alpha \notin \widehat{C_1 \bullet D}(\rho)$ since $\alpha \in dom C_1$ and $acyclic(C_1)$. In this case, in order for $E \leftarrow (\alpha_{\mathsf{m}} = \rho)$ to have a cycle it must be that the cycle "closes-off" by some edge induced by $\alpha_{\mathsf{m}} = \rho$. In other words, there exists a variable $\gamma \rightsquigarrow_{E-\alpha} \alpha_{\mathsf{m}} = \rho \rightsquigarrow_{E-\alpha} \gamma$. Which means that $\rho \rightsquigarrow_{E-\alpha} \alpha$. But $\alpha \notin \widehat{E}(\rho)$ by Lemma 6.18, and it cannot be that $\rho \rightsquigarrow_{E-\alpha} \alpha$ either. For M6 the proof is similar. Case M7 does not affect the reachability graph induced by the constraint, and hence the proof follows. □

The following are two "separation" lemmas. Parts of the constraints that are not touched by the algorithm appear intact in the output constraint. We write $C_1 C_2$ for the disjoint union of two constraints C_1 and C_2 with resept to their domains, without any extra conditions whatsoever (as in $C_1 \bullet C_2$).

Theorem 6.21 (*mkMono* separation)**.** *If* $\texttt{closed}(C_1)$*,* $fcv(\sigma) \subseteq dom(C_1)$*, and* $mkMono(C_1 C_r, f, \sigma) = C$ *then* $mkMono(C_1, f, \sigma) = C_2$ *in the same number of recursive calls, and such that* $C = C_2 C_r$*.*

Proof. By induction on the number of recursive calls to *mkMono*. Cases M1, M3, and M4 are tirival. For M2 we have that $\texttt{closed}(C_1)$, $fcv(\sigma_1 \to \sigma_2) \subseteq dom(C_1)$. Additionally, $mkMono(C_1 C_r, f, \sigma_1) = E$ and $C = mkMono(E, f, \sigma_2)$. We have by induction that $mkMono(C_1, f, \sigma_1) = C_2$ such that $E = C_2 C_r$. Moreover by Theorem 6.17 we get that $\texttt{closed}(C_2)$ and by Lemma 6.11 $fcv(\sigma_2) \subseteq$

$dom(C_2)$. It follows by induction hypothesis that $mkMono(C_2, f, \sigma_2) = E_2$ such that $C = E_2\, C_r$, as required. For M5 we have that $\texttt{closed}(C_1)$, $\alpha \in dom(C_1)$, and $mkMono(C_1\, C_r, f, \alpha) = (E \leftarrow (\alpha_m = \rho))$, where $E = mkMono(True, C_1\, C_r \bullet D, \rho)$ and $(\alpha_\mu \geq [D] \Rightarrow \rho) \in C_1$. We know however that $\texttt{closed}(C_1 \bullet D)$ since $\texttt{closed}(C_1)$ and $\alpha \in dom(C_1)$. It also follows that $fcv(\rho) \subseteq dom(C_1 \bullet D)$. By induction hypothesis then $E_2 = mkMono(True, C_1 \bullet D, \rho)$ and $E = E_2\, C_r$. Since $\alpha \in dom(C_1)$ it follows that $\alpha \in dom(E_2)$ and hence $E \leftarrow (\alpha_m = \rho) = (E_2 \leftarrow (\alpha_m = \rho))\, C_r$ as required. The case for M6 is similar. The case for M7 is straightforward. □

Theorem 6.22 (Unification separation)**.** *The following are true:*

1. *If* $\texttt{closed}(C_1)$, $fcv(\sigma_1, \sigma_2) \subseteq dom(C_1)$, *and* $eqCheck(C_1\, C_r, \sigma_1, \sigma_2) = C$ *then there exists a* C_2 *such that* $eqCheck(C_1, \sigma_1, \sigma_2) = C_2$, *and moreover* $C = C_2\, C_r$.
2. *If* $\texttt{closed}(C_1)$, $fcv(\varsigma, \sigma) \subseteq dom(C_1)$, *and* $subsCheck(C_1\, C_r, \varsigma, \sigma) = C$ *then there exists a* C_2 *such that* $subsCheck(C_1, \varsigma, \sigma) = C_2$, *and moreover* $C = C_2\, C_r$.
3. *If* $\texttt{closed}(C_1)$, $fcv(\alpha, \sigma) \subseteq dom(C_1)$, *and* $updateRigid(C_1\, C_r, \alpha, \sigma) = C$ *then there exists a* C_2 *such that* $updateRigid(C_1, \alpha, \sigma) = C_2$, *and moreover* $C = C_2\, C_r$.
4. *If* $\texttt{closed}(C_1)$, $fcv(\alpha, \sigma) \subseteq dom(C_1)$, *and* $doUpdateR(C_1\, C_r, \alpha, \sigma) = C$ *then there exists a* C_2 *such that* $doUpdateR(C_1, \alpha, \sigma) = C_2$, *and moreover* $C = C_2\, C_r$.
5. *If* $\texttt{closed}(C_1)$, $fcv(\alpha, \varsigma) \subseteq dom(C_1)$, *and* $updateFlexi(C_1\, C_r, \alpha, \varsigma) = C$ *then there exists a* C_2 *such that* $updateFlexi(C_1, \alpha, \varsigma) = C_2$, *and moreover* $C = C_2\, C_r$.
6. *If* $\texttt{closed}(C_1)$, $fcv(\alpha, \varsigma) \subseteq dom(C_1)$, *and* $doUpdateF(C_1\, C_r, \alpha, \varsigma) = C$ *then there exists a* C_2 *such that* $doUpdateF(C_1, \alpha, \varsigma) = C_2$, *and moreover* $C = C_2\, C_r$.
7. *If* $\texttt{closed}(C_1)$, $fcv(\varsigma_1, \varsigma_2) \subseteq dom(C_1)$, *and* $join(C_1\, C_r, \varsigma_1, \varsigma_2) = C, \varsigma_r$ *then there exists a* C_2 *such that* $join(C_1, \varsigma_1, \varsigma_2) = C_2, \varsigma_r$, *and moreover* $C = C_2\, C_r$.

Moreover, in each case the number of recursive calls is preserved.

Proof. We prove the cases simultaneously by induction on the number of recursive calls. We assume that all cases are true for a smaller number of recursive calls. We have the following:

1. Case E1 follows trivially. Cases E2 and E3 follow by the induction hypothesis for *updateRigid*. Case E4 follows by two applications of the induction hypothesis, the second one enabled by making use of Lemma 6.12 and Theorem 6.17. Case E5 follows directly by induction hypothesis, and case E6 cannot happen.
2. Case S1 follows directly by induction hypothesis for *updateFlexi*. We now turn our attention fo case S2 (which is actually the only really interesting case along with case J3). In this case we have the following:

$$\texttt{closed}(C_1) \tag{1}$$
$$fcv([D] \Rightarrow \rho_1, \forall \overline{c}\,.\,\rho_2) \subseteq dom(C_1) \tag{2}$$
$$eqCheck(C_1\, C_r, [D] \Rightarrow \rho_1, \forall \overline{c}\,.\,\rho_2) = C \tag{3}$$

where

$$\rho_3 = \overline{c \mapsto b}]\rho_2 \tag{4}$$
$$E = eqCheck(C_1\, C_r \bullet D, \rho_1, \rho_2) \tag{5}$$
$$\overline{\gamma} = \widehat{E}(dom(C_1\, C_r)) \quad \overline{b} \# E_{\overline{\gamma}} \tag{6}$$

From (1) and (2) we confirm that $fcv(\rho_1, \rho_3) \in dom(C_1 \bullet D)$, and additionally $\texttt{closed}(C_1 \bullet D)$. By induction hypothesis then for (5) we get that $eqCheck(C_1 \bullet D, \rho_1, \rho_2) = E_2$ such that $E = E_2\, C_r$. To finish the case we need to show that:

$$(E_2\, C_r)_{\widehat{E_2\, C_r}(dom(C_1\, C_r))} = E_{2\widehat{E_2}(dom(C_1))}\, C_r$$

We consider each direction separately:

- Assume $(\gamma_\mu\ bnd) \in E_{2\widehat{E_2}(dom(C_1))}C_r$. If $(\gamma_\mu\ bnd) \in C_r$ then clearly $(\gamma_\mu\ bnd) \in E_2 C_r$ and $\gamma \in \widehat{E_2 C_r}(dom(C_1 C_r))$ as required. If on the other hand $(\gamma_\mu\ bnd) \in E_2$ and $\gamma \in \widehat{E_2}(dom(C_1))$, it also follows that $(\gamma_\mu\ bnd) \in E_2 C_r$ and $\gamma \in \widehat{E_2 C_r}(dom(C_1 C_r))$ as required.
- Assume now that:

$$(\gamma_\mu\ bnd) \in E_2 C_r \tag{7}$$
$$\gamma \in \widehat{E_2 C_r}(dom(C_1 C_r)) \tag{8}$$

  First of all, if $(\gamma_\mu\ bnd) \in C_r$ then directly $(\gamma_\mu\ bnd) \in E_{2\widehat{E_2}(dom(C_1))}C_r$ as required. Assume now that $(\gamma_\mu\ bnd) \in E_2$ and $\gamma \notin dom(C_r)$. Then we may have two cases:
  - Purely (i.e. without involving C_r) it is $\gamma \in \widehat{E_2}(dom(C_1))$, in which case we are done; otherwise
  - we have the following path:

$$\alpha \in dom(C_1) \rightsquigarrow_{E_2} \beta \mapsto_{C_r} \rightsquigarrow_{E_2 C_r} \gamma$$

    Let us explain: There is a variable $\alpha \in dom(C_1)$. Now because this variable is not in the domain of C_r it can only be mapped to some other type first by some uses of E_2, hence the $\rightsquigarrow_{E_2}$ notation. But at some point we will reach a variable $\beta \notin dom(E_2)$ that will be mapped via C_r to some other type (hence the notation $\beta \mapsto_{C_r}$). But E_2 is closed, and by Lemma 6.12 $dom(C_1) \subseteq dom(E_2)$. It follows that this case is impossible, and we are done.

3. Case UR1 is trivial. Case UR2 follows by closedness of C_1, the assumptions, and induction hypothesis. Case UR3 is similar. Case UR4 follows by closedness of C_1, the assumptions, and induction hypothesis.
4. Case DR1 and DR2 are straightforward, appealing to Theorem 6.21. Case DR3 and case DR4 use the induction hypothesis.
5. Cases UF1,UF2 are similar to UR1 and UR4 respectively.
6. Cases DF1 and DF2 are straightforward, appealing to Theorem 6.21. Cases DF3 and case DF4 use the induction hypothesis.
7. Cases J1 and J2 are straightforward. The interesting case is J3 which follows in a similar way as the case for S2.

□

We overload below the notation $\texttt{acyclic}(\cdot)$ to schemes. We write $\texttt{acyclic}([D] \Rightarrow \rho)$ to mean $\texttt{acyclic}(D)$.

Theorem 6.23 (Unification preserves acyclicity)**.** *Assume that in all cases, the constrained variables of the arguments are contained within the domain of C_1. Then, the following are true:*

1. $\texttt{closed}(C_1) \wedge \texttt{acyclic}(C_1) \wedge eqCheck(C_1, \sigma_1, \sigma_2) = C_2 \Longrightarrow \texttt{acyclic}(C_2)$
2. $\texttt{closed}(C_1) \wedge \texttt{acyclic}(C_1, \varsigma) \wedge subsCheck(C_1, \varsigma, \sigma, \sigma_0) = C_2 \Longrightarrow \texttt{acyclic}(C_2)$
3. $\texttt{closed}(C_1) \wedge \texttt{acyclic}(C_1) \wedge updateRigid(C_1, \alpha, \sigma) = C_2 \Longrightarrow \texttt{acyclic}(C_2)$
4. $\texttt{closed}(C_1) \wedge \texttt{acyclic}(C_1) \wedge \alpha \notin \widehat{C_1}(\sigma) \wedge doUpdateR(C_1, \alpha, \sigma) = C_2 \Longrightarrow \texttt{acyclic}(C_2)$
5. $\texttt{closed}(C_1) \wedge \texttt{acyclic}(C_1, \varsigma) \wedge updateFlexi(C_1, \alpha, \varsigma, \sigma_0) = C_2 \Longrightarrow \texttt{acyclic}(C_2)$
6. $\texttt{closed}(C_1) \wedge \texttt{acyclic}(C_1, \varsigma) \wedge \alpha \notin \widehat{C_1}(\varsigma) \wedge doUpdateF(C_1, \alpha, \varsigma) = C_2 \Longrightarrow \texttt{acyclic}(C_2)$
7. $\texttt{closed}(C_1) \wedge \texttt{acyclic}(C_1, \varsigma_1, \varsigma_2) \wedge extJoin(C_1, \varsigma_1, \varsigma_2) = C_2, \varsigma \Longrightarrow \texttt{acyclic}(C_2, \varsigma_r)$

Proof. The proof is straightforward induction on the number of recursive calls, and the only interesting cases are those that update the constraint. Let us examine one case, the case for DR3. In this

case we know that:

$$\texttt{closed}(C_1) \wedge \texttt{acyclic}(C_1) \tag{9}$$
$$doUpdateF(C_1, \alpha, \sigma) = E \leftarrow (\alpha_\mu = \sigma) \tag{10}$$
$$(\alpha_\mu \geq \varsigma) \in C_1 \tag{11}$$
$$E = subsCheck(C_1, \varsigma_a, \sigma) \tag{12}$$

Induction hypothesis here gives us that E is acyclic, hence also $E - \alpha$ is acyclic. Assume that by adding $\alpha_\mu = \sigma$ closes-off a cycle. Hence there is a $\gamma \rightsquigarrow_{E-\alpha} \alpha_\mu = \sigma \rightsquigarrow_{E-\alpha} \gamma$, which means that $\alpha \in \widehat{E - \alpha}(\sigma)$. It hence suffices to show that $\alpha \notin \widehat{E}(\sigma)$.

Consider the set of variables $\overline{\gamma} = \widehat{C_1}(\varsigma_a, \sigma)$ and the restriction of C_1 to these variables $C_{1\overline{\gamma}}$; let C_r be such that $C_{1\overline{\gamma}} C_r = C_1$. It must be that $C_{1\overline{\gamma}}$ is acyclic and closed and hence by Theorem 6.22 $E_1 = subsCheck(C_{1\overline{\gamma}}, \varsigma_a, \varsigma)$ and $E = E_1 C_r$. Then we want to show essentially that $\alpha \notin \widehat{E_1 C_r}(\sigma)$ given that $\alpha \in dom(C_r)$, $\alpha \notin fcv(\sigma)$. But this follows as E_1 is closed and we can only remain within its domain as we move along reachability. □

Lemma 6.24 (*mkMono* preserves quantifier rank)**.** *If* $mkMono(C_1, f, \sigma) = C_2$ *then* $q(C_2) \leq q(C_1)$.

Proof. Straightforward induction on the number of recursive calls to *mkMono*. □

In what follows, we use Lemma 6.15, Theorem 6.17, Lemma 6.20, and Theorem 6.23, in order to enable the induction hypotheses. We additionally use the domain monotonicity properties, Lemma 6.11 and Lemma 6.12. In order to focus on the interesting parts of the proofs we ommit explicitly refering to their uses.

Lemma 6.25 (Unification preserves quantifier rank)**.** *The following are true, assuming that the argument variables belong each time in the input constraint* C_1.

1. $\texttt{closed}(C_1) \wedge \texttt{acyclic}(C_1) \wedge eqCheck(C_1, \sigma_1, \sigma_2) = C_2 \Longrightarrow q(C_2) \leq q(C_1)$
2. $\texttt{closed}(C_1) \wedge \texttt{acyclic}(C_1) \wedge subsCheck(C_1, \varsigma, \sigma, \sigma_0) = C_2 \Longrightarrow q(C_2) \leq q(C_1) + q(\varsigma)$
3. $\texttt{closed}(C_1) \wedge \texttt{acyclic}(C_1) \wedge updateRigid(C_1, \alpha, \sigma) = C_2 \Longrightarrow q(C_2) \leq q(C_1)$
4. $\texttt{closed}(C_1) \wedge \texttt{acyclic}(C_1) \wedge \alpha \notin \widehat{C_1}(\sigma) \wedge doUpdateR(C_1, \alpha, \sigma) = C_2 \Longrightarrow q(C_2) \leq q(C_1)$
5. $\texttt{closed}(C_1) \wedge \texttt{acyclic}(C_1) \wedge updateFlexi(C_1, \alpha, \varsigma, \sigma_0) = C_2 \Longrightarrow q(C_2) \leq q(C_1) + q(\varsigma)$
6. $\texttt{closed}(C_1) \wedge \texttt{acyclic}(C_1) \wedge \alpha \notin \widehat{C_1}(\varsigma) \wedge doUpdateF(C_1, \alpha, \varsigma) = C_2 \Longrightarrow q(C_2) \leq q(C_1) + q(\varsigma)$
7. $\texttt{closed}(C_1) \wedge \texttt{acyclic}(C_1) \wedge join(C_1, \varsigma_1, \varsigma_2) = C_2, \varsigma \Longrightarrow q(C_2) + q(\varsigma) \leq q(C_1) + q(\varsigma_1) + q(\varsigma_2)$

Proof. We prove each case by induction on the number of recursive calls. We assume all cases for a smaller number of recursive calls.

1. Case E1 is trivial. Cases E2 and E3 follow by induction hypothesis for *updateRigid*. Case E4 follows by two uses of induction hypothesis, the second enabled using Theorem 6.23 and Theorem 6.22 (and domain monotonicity). Case E5 follows by one use of the induction hypothesis.
2. Case S1 follows by induction hypothesis for *updateFlexi*. In the case for S2 we have that $q(E_{\overline{\gamma}}) \leq q(E) \leq q(C \bullet D) \leq q(C) + q([D] \Rightarrow \rho)$ as required.
3. Case UR1 is trivial. Case UR2 follows by induction hypothesis for *updateRigid*. Case UR3 follows by induction hypothesis for *updateRigid* and case UR4 follows by induction hypothesis for *doUpdateR*.
4. Case DR1 follows by Lemma 6.24. Case DR2 is straightforward. The interesting cases are DR3 and DR4. In the DR3 case we have that

$$\alpha \notin \widehat{C_1}(\sigma)$$
$$(\alpha_\mu \geq \varsigma_a) \in C_1$$
$$E = subsCheck(C_1, \varsigma_a, \sigma)$$
$$C_2 = E \leftarrow (\alpha_m = \sigma)$$

In this case consider the following set $\overline{\gamma} = \widehat{C_1}(\varsigma_a, \sigma)$. It follows that $closed(C_{1\overline{\gamma}})$ and let C_r be such that $C_{1\overline{\gamma}} C_r = C_1$. From this, and Theorem 6.22 for (13) we get that $subsCheck(C_{1\overline{\gamma}}, \varsigma_a, \varsigma) = E_1$ such that $E = E_1 C_r$. But we know that $\alpha \notin dom(E_1)$. Hence $C_2 = E_1 \cup C_r \leftarrow (\alpha_\mu = \sigma)$. We then get that:

$$\begin{array}{rcl} q(C_2) & = & q(E_1) + q(C_r \leftarrow (\alpha_\mu = \sigma)) \\ & \leq & q(C_{1\overline{\gamma}}) + q(\varsigma_a) + q(C_r - \alpha) + 1 \\ & = & q(C_{1\overline{\gamma}} + q(C_r) \\ & = & q(C_1) \end{array}$$

where in the second line we made use of the inductive hypothesis for $subsCheck(C_{1\overline{\gamma}}, \varsigma_a, \varsigma) = E_1$, as we know that it uses the same number of recursive calls as does the equation (13). The DR4 case is similar.

5. The cases for UF1 follows by appealing to the induction hypothesis for *updateRigid*. The case for UF2 follows by appealing to the induction hypothesis for *doUpdateF*.
6. The case for DF1 is straightforward appealing to Lemma 6.24. Case DF2 is straightforward. Case DF3 is similar to the case for DR3. We consider now the case for DF4. We have in this case that:

$$\begin{array}{r} \alpha \notin \widehat{C_1}(\varsigma) \\ (\alpha_\mu \geq \varsigma_a) \in C_1 \\ E, \varsigma_r = join(C_1, \varsigma_a, \varsigma) \\ C_2 = E \leftarrow (\alpha_\mu \geq \varsigma_r) \end{array}$$

As in the case of DF3 consider the set $\overline{\gamma} = \widehat{C_1}(\varsigma_a, \varsigma)$. It follows that $closed(C_{1\overline{\gamma}})$ and let C_r be such that $C_{1\overline{\gamma}} C_r = C_1$. From this, and Theorem 6.22 for (13) we get that $join(C_{1\overline{\gamma}}, \varsigma_a, \varsigma) = E_1, \varsigma_r$ such that $E = E_1 C_r$. But we know that $\alpha \notin dom(E_1)$. Hence $C_2 = E_1 \cup C_r \leftarrow (\alpha_\mu \geq \varsigma_r)$. We then get that:

$$\begin{array}{rcl} q(C_2) & = & q(E_1) + q(C_r \leftarrow (\alpha_\mu \geq \varsigma_r)) \\ & = & (q(E_1) + q(\varsigma_r)) + q(C_r - \alpha) + 1 \\ & \leq & q(C_{1\overline{\gamma}} + q(\varsigma_a) + q(\varsigma) + q(C_r - \alpha) + 1) \\ & = & q(C_{1\overline{\gamma}} + q(C_r) + q(\varsigma) \\ & = & q(C_1) + q(\varsigma) \end{array}$$

where in the third line we made use of the inductive hypothesis for $join(C_{1\overline{\gamma}}, \varsigma_a, \varsigma) = E_1, \varsigma_r$, as we know that it uses the same number of recursive calls as equation (13).

7. Cases J1 and J2 are easy checks. For J3 we have the following: First of all $C_1 \bullet D_1 \bullet D_2$ is acyclic and closed and hence we can aply the induction hypothesis. It follows that $q(E_{\widehat{E}(dom(C))}, [E_{\overline{\delta}}] \Rightarrow \rho_1) \leq q(E) \leq q(C \bullet D_1 \bullet D_2) \leq q(C) + q([D_1] \Rightarrow \rho_1) + q([D_2] \Rightarrow \rho_2)$, as required.

□

Theorem 6.26 (*mkMono* weakening)**.** *If* `closed`(C_1), $fcv(\sigma) \subseteq dom(C_1)$, *and* $mkMono(C_1, f, \sigma) = C_2$ *then* $mkMono(C_1 C_r, f, \sigma) = C_2 C_r$ *in the same number of recursive calls (and the second fails if the first fails).*

Proof. Easy induction, similar to the proof of Theorem 6.21. □

Theorem 6.27 (Unification weakening)**.** *The following are true:*

1. *If* `closed`(C_1) *and* $fcv(\sigma_1, \sigma_2) \subseteq dom(C_1)$ *and* $eqCheck(C_1, \sigma_1, \sigma_2) = C_2$ *then it is the case that* $eqCheck(C_1 C_r, \sigma_1, \sigma_2) = C_2 C_r$.

2. *If* $\texttt{closed}(C_1)$ *and* $fcv(\varsigma, \sigma) \subseteq dom(C_1)$ *and* $subsCheck(C_1, \varsigma, \sigma) = C_2$ *then it is the case that* $subsCheck(C_1\,C_r, \varsigma, \sigma) = C_2\,C_r$.
3. *If* $\texttt{closed}(C_1)$ *and* $fcv(\alpha, \sigma) \subseteq dom(C_1)$ *and* $updateRigid(C_1, \alpha, \sigma) = C_2$ *then it is the case that* $updateRigid(C_1\,C_r, \alpha, \sigma) = C_2\,C_r$.
4. *If* $\texttt{closed}(C_1)$ *and* $fcv(\alpha, \sigma) \subseteq dom(C_1)$ *and* $doUpdateR(C_1, \alpha, \sigma) = C_2$ *and it is the case that* $doUpdateR(C_1\,C_r, \alpha, \sigma) = C_2\,C_r$.
5. *If* $\texttt{closed}(C_1)$ *and* $fcv(\alpha, \varsigma) \subseteq dom(C_1)$ *and* $updateFlexi(C_1, \alpha, \varsigma) = C_2$ *then it is the case that* $updateFlexi(C_1\,C_r, \alpha, \varsigma) = C_2\,C_r$.
6. *If* $\texttt{closed}(C_1)$ *and* $fcv(\alpha, \varsigma) \subseteq dom(C_1)$ *and* $doUpdateF(C_1, \alpha, \varsigma) = C_2$ *then it is the case that* $doUpdateF(C_1\,C_r, \alpha, \varsigma) = C_2\,C_r$.
7. *If* $\texttt{closed}(C_1)$ *and* $fcv(\varsigma_1, \varsigma_2) \subseteq dom(C_1)$ *and* $join(C_1, \varsigma_1, \varsigma_2) = C_2, \varsigma_r$ *then it is the case that* $join(C_1\,C_r, \varsigma_1, \varsigma_2) = C_2\,C_r, \varsigma_r$.

Moreover the number of recursive calls is preserved, and if the first call fails instead, then the second fails as well.

Proof. Simultaneous induction on the number of recursive calls, similar to the proof of Theorem 6.22. □

Theorem 6.28 (Termination)**.** *The unification algorithm terminates (written with the notation* $\Downarrow$*) when given well-formed inputs. In particular*

1. *If* $\texttt{closed}(C_1)$ *and* $\texttt{acyclic}(C_1)$ *and* $fcv(\sigma_1, \sigma_2) \subseteq dom(C_1)$ *then* $eqCheck(C_1, \sigma_1, \sigma_2) \Downarrow$.
2. *If* $\texttt{closed}(C_1)$ *and* $\texttt{acyclic}(C_1, \varsigma)$ *and* $fcv(\varsigma, \sigma) \subseteq dom(C_1)$ *then* $subsCheck(C_1, \varsigma, \sigma) \Downarrow$.
3. *If* $\texttt{closed}(C_1)$ *and* $\texttt{acyclic}(C_1)$ *and* $fcv(\alpha, \sigma) \subseteq dom(C_1)$ *then* $updateRigid(C_1, \alpha, \sigma) \Downarrow$.
4. *If* $\texttt{closed}(C_1)$ *and* $\texttt{acyclic}(C_1)$ *and* $fcv(\alpha, \sigma) \subseteq dom(C_1)$ *and* $\alpha \notin \widehat{C_1}(\sigma)$ *then it is the case that* $doUpdateR(C_1, \alpha, \sigma) \Downarrow$.
5. *If* $\texttt{closed}(C_1)$ *and* $\texttt{acyclic}(C_1, \varsigma)$ *and* $fcv(\varsigma, \alpha) \subseteq dom(C_1)$ *then* $updateFlexi(C_1, \alpha, \varsigma) \Downarrow$.
6. *If* $\texttt{closed}(C_1)$ *and* $\texttt{acyclic}(C_1, \varsigma)$ *and* $fcv(\varsigma, \alpha) \subseteq dom(C_1)$ *and* $\alpha \notin \widehat{C_1}(\varsigma)$ *then it is the case that* $doUpdateF(C_1, \alpha, \varsigma) \Downarrow$.
7. *If* $\texttt{closed}(C_1)$ *and* $\texttt{acyclic}(C_1, \varsigma_1, \varsigma_2)$ *and* $fcv(\varsigma_1, \varsigma_2) \subseteq dom(C_1)$ *then* $join(C_1, \varsigma_1, \varsigma_2) \Downarrow$.

Proof. Assume that ϖ stands for either types or schemes. We assume that $q(\varpi) = 0$ whenever $\varpi = \sigma$, or $q(\varpi) = q(\varsigma)$ when $\varpi = \varsigma$. For a triple $(C_1, \varpi_1, \varpi_2)$ we associate the following lexigoraphic triple:

$$\varrho(C_1, \varpi_1, \varpi_2) = \langle q(C_1) + q(\varpi_1) + q(\varpi_2), |\varpi_1| + |\varpi_2|, |\widehat{C_1}(\varpi_1)| + |\widehat{C_1}(\varpi_2)| \rangle$$

where $|[D] \Rightarrow \rho| = 1 + |\rho|$ and $|\sigma|$ is an ordinary size function on types. We proceed to show that eventually in every path of recursive calls, the metric ϱ on the arguments reduces.

1. For *eqCheck*, case E1 is trivial. Case E2 and case E3 follow by inlining the proof for the *updateRigid* function. For E4 we have that $eqCheck(C_1, \sigma_1 \to \sigma_2, \sigma_3 \to \sigma_4)$ relies on a call to $eqCheck(C_1, \sigma_1, \sigma_3)$. First of all we know that $\varrho(C_1, \sigma_1, \sigma_3) < \varrho(C_1, \sigma_1 \to \sigma_2, \sigma_3 \to \sigma_4)$ since the quantifier rank is preserved but the sizes of types become smaller. It follows that either $eqCheck(C_1, \sigma_1, \sigma_3)$ either *fail*s, or returns $E = eqCheck(C_1, \sigma_1, \sigma_3)$. In the first case we are trivially done. In the second case, by Lemma 6.25 we have that $q(E) \leq q(C_1)$. It follows that $\varrho(E, \sigma_2, \sigma_4) < \varrho(C_1, \sigma_1 \to \sigma_2, \sigma_3 \to \sigma_4)$, and induction hypothesis finishes the case because it must be that $eqCheck(E, \sigma_2, \sigma_4) \Downarrow$. If it *fail*s, the whole case *fail*s, or if it succeeds we return the output of $eqCheck(E, \sigma_2, \sigma_4)$. In all other cases of *eqCheck* we *fail* (and hence terminate) using E6.
2. For *subsCheck*, case S1 follows by inlining the proof of the case of *updateFlexi*. For case S2 we have a call to $subsCheck(C_1, [D] \Rightarrow \rho_1, \forall \overline{c}\,.\,\rho_2)$. The first step $\rho_3 = [\overline{b \mapsto c}]\rho_2$ clearly terminates. Then we examine the metric: $\varrho(C_1 \bullet D, \rho_1, \rho_3) < \varrho(C_1, [D] \Rightarrow \rho_1, \rho_3)$ and clearly the constraint $C_1 \bullet D$ is acyclic and closed with the variables of ρ_1 and ρ_3 in its domain. Induction hypothesis give us then that the recursive call to *eqCheck* always terminates, and so does the original call to *subsCheck*.

3. For *updateRigid*, the case UR1 is trivial. For the case of UR2 we have that $updateRigid(C_1, \alpha, \beta) = updateRigid(C_1, \alpha, \gamma)$ where $(\beta_\mu = \gamma) \in C_1$. Hence we have that $\varrho(C_1, \alpha, \gamma) = \langle q(C_1), 2, |\widehat{C_1}(\alpha)| + |\widehat{C_1}(\beta)| \rangle$. But note that $\beta \notin \widehat{C_1}(\gamma)$ because the constraint is acyclic, and hence: $|\widehat{C_1}(\gamma)| < |\widehat{C_1}(\beta)|$. It follows that $\varrho(C_1, \alpha, \gamma) < \varrho(C_1, \alpha, \beta)$, and hence the case terminates. The case for UR3 is similar. Finally, the case for UR3 follows by inlining the proof for *doUpdateR*.
4. For *doUpdateR*, the cases DR1 and DR2 are trivial by an easy termination lemma on acyclic constraints for *mkMono*. For case DR3 we have that $(\alpha_\mu \geq \varsigma_a) \in C_1$ and we have a call to $doUpdateF(C_1, \alpha, \sigma)$. We recursively call $subsCheck(C_1, \varsigma, \sigma)$. Now, it is not obvious that this terminates. However let us consider $\overline{\gamma} = \widehat{C_1}(\varsigma_a, \sigma)$ and $C_{1\overline{\gamma}}$. It must be that $C_{1\overline{\gamma}}$ is closed and acyclic and a subset of C_1, and moreover $(\alpha_\mu \geq \varsigma_a) \notin C_{1\overline{\gamma}}$ since α is not reachable from ς_a nor σ through C_1. It follows that $subsCheck(C_{1\overline{\gamma}}, \varsigma, \sigma)$ does terminate by induction hypothesis because: $\varrho(C_{1\overline{\gamma}}, \varsigma_a, \sigma) < \varrho(C_1, \alpha, \sigma)$ for the reason that $(\alpha_\mu \geq \varsigma_a) \notin C_{1\overline{\gamma}}$. By Theorem 6.27 that the call to $subsCheck(C_1, \varsigma, \sigma)$ must also terminate.
5. For *updateFlexi* the cases are similar to *updateRigid*.
6. For *doUpdateF* the cases are similar to *doUpdateR*, appealing to Theorem 6.27.
7. For *join* the cases J1 and J2 are obvious. For J3 observe that $q(C_1) + q([D_1] \Rightarrow \rho_1) + q([D_2] \Rightarrow \rho_2) = q(C_1 \bullet D_1 \bullet D_2) = q(C_1 \bullet D_1 \bullet D_2) + q(\rho_1) + q(\rho_2)$. however $|[D_1] \Rightarrow \rho_1| + |[D_2] \Rightarrow \rho_2| = 2 + |\rho_1| + |\rho_2|$ and hence, by induction hypothesis the recursive call to *eqCheck* terminates, and hence the whole case J3 terminates.

□

6.3.2 Unification soundness properties

We start by giving a series of lemmas that ensure that unification produces well-formed outputs, when given well-formed inputs.

Lemma 6.29. *Assume that* $\vdash C$ `wf` *and* $\overline{\alpha}$ *is some set of variables. If* $\overline{\gamma} = \widehat{C}(\overline{\alpha})$ *then also* $\vdash C_{\overline{\gamma}}$ `wf`.

Proof. Easy unfolding of the definitions. □

Definition 6.30 (Mono-flags not in scheme bounds)**.** We write $\texttt{mweak}(C)$ for the smallest relation that asserts that for all $(\alpha_\mu \geq [D] \Rightarrow \rho) \in C$, it is the case that $\mu = \star$ and $\texttt{mweak}(D)$.

Definition 6.31 (Well-flagged constraints)**.** Consider the relations $\Delta \vdash^\mu C$ and $\Delta \vdash^\mu \varsigma$ defined by:

$$\frac{(a)\ dom(C) \# dom(\Delta) \quad (b)\ \text{for all } (\alpha_{\mathsf{m}}\ bnd) \in C, \text{for all } \beta \in fcv(bnd), \Delta\Delta_c(\beta) = \mathsf{m} \text{ and } (bnd = \bot \text{ or } bnd = \tau) \quad (c)\ \text{for all } (\alpha_\mu \geq \varsigma) \in C, \Delta\Delta_C \vdash^\mu \varsigma \wedge \mu = \star}{\Delta \vdash^\mu C}\ \text{SMWF}$$

$$\frac{\Delta \vdash^\mu D}{\Delta \vdash^\mu [D] \Rightarrow \rho}\ \text{SMSCH}$$

We write $\vdash^\mu C$ whenever $\emptyset \vdash^\mu C$.

Lemma 6.32 (Well-flagged *mkMono* preservation)**.** *If* $\vdash^\mu C_1$, $mkMono(C_1, f, \sigma) = C_2$, $\texttt{closed}(C_1)$, *and* $fcv(\sigma) \subseteq dom(C_1)$ *then it is the case that* $\vdash^\mu C_2$, *and moreover* $\Delta_{C_2}(fcv(\sigma)) = \mathsf{m}$. *Additionally, if* $f = True$ *then* $\sigma = \tau$.

Proof. Induction on the number of recursive calls. Case M1 is trivial. For the case of M2 we have that $mkMono(C_1, f, \sigma_1 \to \sigma_2) = C_2$ where $E = mkMono(C_1, f, \sigma_1)$ and $C_2 = mkMono(E, f, \sigma_2)$. By induction hypothesis we have that $\vdash^\mu E$ and $\Delta_E(fcv(\sigma_1)) = \mathsf{m}$. By Lemma 6.15, and Lemma 6.11 we have that $\texttt{closed}(E)$ and $fcv(\sigma_2) \subseteq dom(E)$. Hence, by induction $\vdash^\mu C_2$ and $\Delta_{C_2}(fcv(\sigma_2)) = \mathsf{m}$. But, by Lemma 6.13 we also have that $\Delta_{C_2}(fcv(\sigma_1)) = \mathsf{m}$. Consequently, $\Delta_{C_2}(fcv(\sigma_1 \to \sigma_2)) = \mathsf{m}$, as required. Cases M3 and M4 are trivial. For case M5 we have that $mkMono(C_1, f, \alpha) = E \leftarrow (\alpha_{\mathsf{m}} = \rho)$ where $E = mkMono(C_1 \bullet D, True, \rho)$ and $(\alpha_\mu \geq [D] \Rightarrow \rho) \in C_1$. First of all, since $\vdash^\mu C_1$,

$\mu = \star$. Moreover it must be that $\vdash^\mu C_1 \bullet D$ and $\texttt{closed}(C_1 \bullet D)$. By induction then $\vdash^\mu E$ and $\Delta_E(fcv(\rho)) = \mathsf{m}$ and $\rho = \tau$. It follows from these last three conditions that $\vdash^\mu E \leftarrow (\alpha_\mathsf{m} = \rho)$. The case for M6 is similar. Case M7 is an easy check. □

Lemma 6.33 (Well-flagged preservation)**.** *The following are true:*

1. *If* $\vdash^\mu C_1$, $\texttt{closed}(C_1)$, $\texttt{acyclic}(C_1)$, $fcv(\sigma_1, \sigma_2) \subseteq dom(C_1)$, *and* $eqCheck(C_1, \sigma_1, \sigma_2) = C_2$ *then* $\vdash^\mu C_2$.
2. *If* $\vdash^\mu C_1$, $\texttt{closed}(C_1)$, $\texttt{acyclic}(C_1, \varsigma)$, $\Delta_{C_1} \vdash^\mu \varsigma$, *and* $subsCheck(C_1, \varsigma, \sigma) = C_2$ *then* $\vdash^\mu C_2$.
3. *If* $\vdash^\mu C_1$, $\texttt{closed}(C_1)$, $\texttt{acyclic}(C_1)$, $fcv(\alpha, \sigma) \subseteq dom(C_1)$, $updateRigid(C_1, \alpha, \sigma) = C_2$ *then* $\vdash^\mu C_2$.
4. *If* $\vdash^\mu C_1$, $\texttt{closed}(C_1)$, $\texttt{acyclic}(C_1)$, $fcv(\alpha, \sigma) \subseteq dom(C_1)$, $\alpha \notin \widehat{C_1}(\sigma)$, *and* $doUpdateR(C_1, \alpha, \sigma) = C_2$ *then* $\vdash^\mu C_2$.
5. *If* $\vdash^\mu C_1$, $\texttt{closed}(C_1)$, $\texttt{acyclic}(C_1, \varsigma)$, $fcv(\varsigma, \alpha) \subseteq dom(C_1)$, $\Delta_{C_1} \vdash^\mu \varsigma$, *and* $updateFlexi(C_1, \alpha, \varsigma) = C_2$ *then* $\vdash^\mu C_2$.
6. *If* $\vdash^\mu C_1$, $\texttt{closed}(C_1)$, $\texttt{acyclic}(C_1, \varsigma)$, $fcv(\varsigma, \alpha) \subseteq dom(C_1)$, $\alpha \notin \widehat{C_1}(\varsigma)$, $\Delta_{C_1} \vdash^\mu \varsigma$, *and it is the case that* $doUpdateF(C_1, \alpha, \varsigma) = C_2$ *then* $\vdash^\mu C_2$.
7. *If* $\vdash^\mu C_1$, $\texttt{closed}(C_1)$, $\texttt{acyclic}(C_1, \varsigma_1, \varsigma_2)$, $fcv(\varsigma_1, \varsigma_2) \subseteq dom(C_1)$, $\Delta_{C_1} \vdash^\mu \varsigma_1, \varsigma_2$, *and it is the case that* $join(C_1, \varsigma_1, \varsigma_2) = C_2, \varsigma$ *then* $\vdash^\mu C_2$ *and* $\Delta_{C_2} \vdash^\mu \varsigma$.

Proof. We show the cases simultaneously by induction on the number of recursive calls, assuming that all cases are true for calls with smaller number of recursive subcalls.

1. Case E1 is trivial, cases E2 and E3 follow by induction hypothesis for *updateRigid*. The case of E4 follows by induction hypothesis, Theorem 6.17, Lemma 6.12, and a second use of induction hypothesis. Case E5 follows directly by induction hypothesis.
2. Case S1 follows by induction hypothesis for *updateFlexi*. For S2 since $\texttt{closed}(C_1)$, $\texttt{acyclic}(C_1)$, and $fcv(\varsigma) \in dom(C_1)$, we have that $\vdash^\mu C_1 \bullet D$, $\texttt{closed}(C_1 \bullet D)$, and $\texttt{acyclic}(C_1 \bullet D)$. It follows by induction that $\vdash^\mu E$ and consequently it is straightforward to verify that $\vdash^\mu E_{\overline{\gamma}}$ where $\overline{\gamma} = \widehat{E}(dom(C_1))$.
3. Case UR1 is trivial. Cases UR2 and UR3 follow because the constraint is closed, and appealing to the induction hypothesis for *updateRigid*. Case UR4 follows by induction hypothesis for *doUpdateR*.
4. The case for DR1 follows from Lemma 6.32. The case for DR2 follows by the assumptions directly. For case DR3 we have that $doUpdateR(C_1, \alpha, \sigma) = E \leftarrow (\alpha_\mu = \sigma)$ where $(\alpha_\mu \geq \varsigma_a) \in C_1$ and $E = extSubsCheck(C_1, \varsigma_a, \sigma)$. Because $\texttt{closed}(C_1)$ it must be that $fcv(\varsigma_a, \varsigma) \subseteq dom(C_1)$. Moreover, it must also be that $\mu = \star$ because $\vdash^\mu C_1$. Hence, by induction hypothesis $\vdash^\mu E$. By the separation theorem (taking as the set of separating variables the reachable variables through C_1 of ς_a, σ) it must be that $(\alpha_\mu \geq \varsigma_a) \in E$. and hence $\vdash^\mu E \leftarrow (\alpha_\mu = \sigma)$ (i.e. it is not the case that $\Delta_E(\alpha) = \mathsf{m}$, in which case the final update would potentially break the $\vdash^\mu$ relation). Case DR4 follows by induction hypothesis.
5. Case UF1 follows by induction hypothesis for *updateRigid*. Case UF2 follows by induction hypothesis for *doUpdateF*.
6. Case DF1 is similar to the case for DR1. Case DF2 is trivial Case DF3 follows by induction hypothesis for *extSubsCheck*. Finally let us examine case DF4. We have that $doUpdateF(C_1, \alpha, \varsigma) = E \leftarrow (\alpha_\mu \geq \varsigma)$, where $E, \varsigma_r = extJoin(C_1, \varsigma, \varsigma_r)$ and $(\alpha_\mu \geq \varsigma_r) \in C_1$. It must be that $\mu = \star$, and by induction $\vdash^\mu E$. To finish the case we need to show that $\vdash^\mu E \leftarrow (\alpha_\star \geq \varsigma_r)$. By separation it must be that $(\alpha_\mu \geq \varsigma) \in E$, so we did not un-lift some monomorphic flag from E. Hence we must show that $\Delta_{E \leftarrow (\alpha_\star \geq \varsigma_r)} \vdash^\mu \varsigma_r$. But we know that $\Delta_E \vdash^\mu \varsigma_r$ by induction, and by separation $\alpha \notin fcv(\varsigma_r)$. It follows that $\Delta_{E \leftarrow (\alpha_\star \geq \varsigma_r)} \vdash^\mu \varsigma_r$.
7. Cases J1 and J2 are trivial. For J3 it is the case that

$$join(C_1, [D_1] \Rightarrow \rho_1, [D_2] \Rightarrow \rho_2) = E_{\widehat{E}(dom(C_1))}, [E_{\overline{\delta}}] \Rightarrow \rho_1$$

where $E = eqCheck(C_1 \bullet D_1 \bullet D_2, \rho_1, \rho_2)$ and $\overline{\delta} = \widehat{E}(\rho_1) - \widehat{E}(dom(C_1))$. Clearly the constraint $C_1 \bullet D_1 \bullet D_2$ satisfies $\vdash^\mu$, and is closed and acyclic. Consequently by induction hypothesis $\vdash^\mu E$, and hence $\vdash^\mu E_{\widehat{E}(dom(C_1))}$. We need to additionally show that $\Delta_{E_{\widehat{E}(dom(C_1))}} \vdash^\mu [E_{\overline{\delta}}] \Rightarrow \rho_1$, which follows because it is easy to see that $\Delta_{E_{\widehat{E}(dom(C_1))}} \vdash^\mu E_{\overline{\delta}}$ (since $\vdash^\mu E$).

□

At this point we have shown all the structural properties of the constraints being preserved by unification. But we have not talked yet about (i) the fact that schemes are not allowed to bind monomorphic variables (the intuition being that these always originate in some environment variable, and (ii) the fact that if the body of a scheme is a single variable, it is either monomorphic, or it is quantified with $\bot$ in the current scheme.

Definition 6.34 (Weak well-formedness)**.** A constraint C_1 is weakly well-formed, written $\texttt{wfweak}(C_1)$ iff $\texttt{closed}(C_1) \wedge \texttt{acyclic}(C_1) \wedge \vdash^\mu C_1$.

Lemma 6.35. *The following are true:*

1. *If* $\texttt{wfweak}(C_1)$, $fcv(\sigma_1, \sigma_2) \subseteq dom(C_1)$, $eqCheck(C_1, \sigma_1, \sigma_2) = C_2$, *and* $\Delta_{C_1}(\gamma) = \mathsf{m}$ *then* $\widehat{C_1}(\gamma) \subseteq \widehat{C_2}(\gamma)$.
2. *If* $\texttt{wfweak}(C_1)$, $\texttt{acyclic}(\varsigma)$, $\Delta_{C_1} \vdash^\mu \varsigma$, $fcv(\varsigma, \sigma) \subseteq dom(C_1)$, $subsCheck(C_1, \varsigma, \sigma) = C_2$, *and* $\Delta_{C_1}(\gamma) = \mathsf{m}$ *then* $\widehat{C_1}(\gamma) \subseteq \widehat{C_2}(\gamma)$.
3. *If* $\texttt{wfweak}(C_1)$, $fcv(\alpha, \sigma) \subseteq dom(C_1)$, $updateRigid(C_1, \alpha, \sigma) = C_2$, *and* $\Delta_{C_1}(\gamma) = \mathsf{m}$ *then* $\widehat{C_1}(\gamma) \subseteq \widehat{C_2}(\gamma)$.
4. *If* $\texttt{wfweak}(C_1)$, $fcv(\alpha, \sigma) \subseteq dom(C_1)$, $\alpha \notin \widehat{C_1}(\sigma)$, $doUpdateR(C_1, \alpha, \sigma) = C_2$, *and* $\Delta_{C_1}(\gamma) = \mathsf{m}$ *then* $\widehat{C_1}(\gamma) \subseteq \widehat{C_2}(\gamma)$.
5. *If* $\texttt{wfweak}(C_1)$, $\texttt{acyclic}(\varsigma)$, $\Delta_{C_1} \vdash^\mu \varsigma$, $fcv(\alpha, \varsigma) \subseteq dom(C_1)$, $updateFlexi(C_1, \alpha, \varsigma) = C_2$, *and* $\Delta_{C_1}(\gamma) = \mathsf{m}$ *then* $\widehat{C_1}(\gamma) \subseteq \widehat{C_2}(\gamma)$.
6. *If* $\texttt{wfweak}(C_1)$, $\texttt{acyclic}(\varsigma)$, $\Delta_{C_1} \vdash^\mu \varsigma$, $fcv(\alpha, \varsigma) \subseteq dom(C_1)$, $\alpha \notin \widehat{C_1}(\varsigma)$, $doUpdateF(C_1, \alpha, \varsigma) = C_2$, *and* $\Delta_{C_1}(\gamma) = \mathsf{m}$ *then* $\widehat{C_1}(\gamma) \subseteq \widehat{C_2}(\gamma)$.
7. *If* $\texttt{wfweak}(C_1)$, $\texttt{acyclic}(\varsigma_1, \varsigma_2)$, $\Delta_{C_1} \vdash^\mu \varsigma_1, \varsigma_2$, $fcv(\varsigma_1, \varsigma_2) \subseteq dom(C_1)$, $join(C_1, \varsigma_1, \varsigma_2) = C_2, \varsigma$, *and* $\Delta_{C_1}(\gamma) = \mathsf{m}$ *then* $\widehat{C_1}(\gamma) \subseteq \widehat{C_2}(\gamma)$.

Proof. Simultaneous induction on the number of recursive calls; assuming that all cases are true for smaller number of recursive calls. We do not show explicitly the $\vdash^\mu$, $\texttt{closed}(\cdot)$, and $\texttt{acyclic}(\cdot)$ properties are preserved, nor that the variables of the arguments are preserved in the domains of the outputs, as these follows whenever they are needed by Lemma 6.33, Theorem 6.17, Theorem 6.23, and Lemma 6.12.

1. Case E1 is trivial. Cases E2 and E3 follows directly from induction hypothesis for *updateRigid*. For E4 wwe have that $eqCheck(C_1, \sigma_1 \to \sigma_2, \sigma_3 \to \sigma_4) = C_2$ where $eqCheck(C_1, \sigma_1, \sigma_3) = E$ and $eqCheck(E, \sigma_2, \sigma_4) = C_2$. Assume that $\Delta_{C_1}(\gamma) = \mathsf{m}$. By induction $\widehat{C_1}(\gamma) \subseteq \widehat{E}(\gamma)$. Moreover from Lemma 6.14 we get that $\Delta_E(\gamma) = \mathsf{m}$. It follows by induction that $\widehat{E}(\gamma) \subseteq \widehat{C_2}(\gamma)$. Consequently $\widehat{C_1}(\gamma) \subseteq \widehat{C_2}(\gamma)$, as required. Case E5 follows by induction hypothesis, and case E6 cannot happen.
2. Case S1 follows by induction hypothesis for *updateFlexi*. For S2 we have that $\texttt{wfweak}(C_1)$, $\texttt{acyclic}([D] \Rightarrow \rho_1)$, $\Delta_{C_1} \vdash^\mu [D] \Rightarrow \rho_1$, $subsCheck(C_1, [D] \Rightarrow \rho_1, \forall \overline{c}.\rho_2) = E_{\overline{\gamma}}$ where $\rho_3 = [\overline{c \mapsto b}]\rho_2$, $E = eqCheck(C_1 \bullet D, \rho_1, \rho_3)$ and $\gamma = \widehat{E}(dom(C_1))$. It follows by induction hypothesis, if $\Delta_{C_1 \bullet D}(\gamma) = \mathsf{m}$ then $\widehat{C_1 \bullet D}(\gamma) \subseteq \widehat{E}(\gamma)$. Assume now that $\Delta_{C_1}(\gamma) = \mathsf{m}$. It follows that $\gamma \in dom(C_1)$. But also $\Delta_{C_1 \bullet D}(\gamma) = \mathsf{m}$. Hence $\widehat{C_1 \bullet D}(\gamma) = \widehat{C_1}(\gamma)$ since $dom(D) \# fcv(C_1)$. Then $\widehat{C_1}(\gamma) \subseteq \widehat{E}(\gamma)$. But $\gamma \in dom(C_1)$ and consequently $\widehat{E}(\gamma) = \widehat{E_{\overline{\gamma}}}(\gamma)$. It follows that $\widehat{C_1}(\gamma) \subseteq \widehat{E_{\overline{\gamma}}}(\gamma)$ as required.

3. Case UR1 is trivial. For case UR2 we have that $\texttt{wfweak}(C_1)$, and $\mathit{updateRigid}(C_1, \alpha, \beta) = C_2$ where $C_2 = \mathit{updateRigid}(C_1, \alpha, \gamma)$ and $(\beta_\mu = \gamma) \in C_1$. By induction hypothesis for *updateRigid* we are done. The case of UR3 is similar, and the case of UR4 follows by induction hypothesis for *doUpdateR*.
4. For case DR1 we have that $\mathit{doUpdateR}(C_1, \alpha, \sigma) = E \leftarrow (\alpha_\mathsf{m} = \sigma)$ where $(\alpha_\mathsf{m}\ \bot) \in C_1$ and $E = \mathit{mkMono}(\mathit{True}, C_1, \sigma)$. Assume that $\Delta_{C_1}(\gamma) = \mathsf{m}$. By an easy similar property for *mkMono* it follows that $\widehat{C_1}(\gamma) \subseteq \widehat{E}(\gamma)$. But by the separation theorem Theorem 6.21 (by using as the separating set of variables the set $\widehat{C_1}(\sigma)$) we know that $(\alpha_\mathsf{m}\ \bot) \in \mathit{dom}(E)$. Consequently $\widehat{E}(\gamma) \subseteq \widehat{(E \leftarrow (\alpha_\mathsf{m} = \sigma))}(\gamma)$ as required. Case DR2 is straightforward. For case DR3 we have that $\mathit{doUpdateR}(C_1, \alpha, \sigma) = C_2$ where $(\alpha_\star \geq \varsigma_a) \in C_1$ (since $\vdash^\mu C_1$)it must be that $\mu = \star$), $E = \mathit{subsCheck}(C_1, \varsigma_a, \sigma)$, and $C_2 = E \leftarrow (\alpha_\star = \sigma)$. By the separation theorem, Theorem 6.22, it must be that $(\alpha_\star \geq \varsigma_a) \in E$, but we also have that $\Delta_E(\gamma) = \mathsf{m}$ and hence $\alpha \notin \widehat{E}(\gamma)$. By induction hypothesis we get that if $\Delta_{C_1}(\gamma) = \mathsf{m}$, $\widehat{C_1}(\gamma) \subseteq \widehat{E}(\gamma)$. But beacause $\vdash^\mu E$ by Lemma 6.33 and since $\Delta_E(\gamma) = \mathsf{m}$, it must be that $\widehat{E}(\gamma) = \widehat{E \leftarrow (\alpha_\star = \sigma)}(\gamma)$. and we are done. Case DR4 follows by induction hypothesis.
5. The cases for *updateFlexi* are similar to the cases for *updateRigid*.
6. The cases for *doUpateF* are similar to the cases for *doUpdateR*.
7. Cases J1 and J2 follow trivially. For case J3 we have that

$$\mathit{join}(C_1, [D_1] \Rightarrow \rho_1, [D_2] \Rightarrow \rho_2) = (E_{\widehat{E}(\mathit{dom}(C_1))}, [E_\delta] \Rightarrow \rho_1)$$

given that $E = \mathit{eqCheck}(C_1 \bullet D_1 \bullet D_2, \rho_1, \rho_2)$, and $\overline{\delta} = \widehat{E}(\rho_1) - \widehat{E}(\mathit{dom}(C_1))$. Assume that $\Delta_{C_1}(\gamma) = \mathsf{m}$. Then also $\Delta_{C_1 \bullet D_1 \bullet D_2}(\gamma) = \mathsf{m}$. By induction hypothesis we then get that $\widehat{C_1}(\gamma) = \widehat{C_1 \bullet D_1 \bullet D_2}(\gamma) \subseteq \widehat{E}(\gamma) = \widehat{E_{\widehat{E}(\mathit{dom}(C_1))}}(\gamma)$, as required.

□

Definition 6.36 (Quantified variable condition)**.** We define the relations $\texttt{wfqvar}(C)$ and $\texttt{wfqvar}(\varsigma)$ as:

$$\frac{\text{for all } (\alpha_\mu \geq \varsigma) \in C, \texttt{wfqvar}(\varsigma)}{\texttt{wfqvar}(C)}\ \text{CWFQ} \qquad \frac{\texttt{wfqvar}(()D) \quad \text{for all} \alpha \in \mathit{dom}(D), \Delta_D(\alpha) = \star}{\texttt{wfqvar}([D] \Rightarrow \rho)}\ \text{SWFQ}$$

Lemma 6.37. *The following are true:*

1. *If* $\texttt{wfweak}(C_1)$, $\texttt{wfqvar}(C_1)$, $\mathit{fcv}(\sigma_1, \sigma_2) \subseteq \mathit{dom}(C_1)$, $\mathit{eqCheck}(C_1, \sigma_1, \sigma_2) = C_2$, *then* $\texttt{wfqvar}(C_2)$ *and if* $\Delta_{C_2}(\gamma) = \mathsf{m}$ *then there is a* $\beta \in \mathit{dom}(C_1)$ *such that* $\Delta_{C_1}(\beta) = \mathsf{m}$ *and* $\gamma \in \widehat{C_2}(\beta)$.
2. *If* $\texttt{wfweak}(C_1)$, $\texttt{wfqvar}(C_1, \varsigma)$, $\texttt{acyclic}(\varsigma)$, $\Delta_{C_1} \vdash^\mu \varsigma$, $\mathit{fcv}(\varsigma, \sigma) \subseteq \mathit{dom}(C_1)$, $\mathit{subsCheck}(C_1, \varsigma, \sigma) = C_2$, *then* $\texttt{wfqvar}(C_2)$ *and if* $\Delta_{C_2}(\gamma) = \mathsf{m}$ *then there is a* $\beta \in \mathit{dom}(C_1)$ *such that* $\Delta_{C_1}(\beta) = \mathsf{m}$ *and* $\gamma \in \widehat{C_2}(\beta)$.
3. *If* $\texttt{wfweak}(C_1)$, $\texttt{wfqvar}(C_1)$, $\mathit{fcv}(\alpha, \sigma) \subseteq \mathit{dom}(C_1)$, $\mathit{updateRigid}(C_1, \alpha, \sigma) = C_2$, *then* $\texttt{wfqvar}(C_2)$ *and if* $\Delta_{C_2}(\gamma) = \mathsf{m}$ *then there is a* $\beta \in \mathit{dom}(C_1)$ *such that* $\Delta_{C_1}(\beta) = \mathsf{m}$ *and* $\gamma \in \widehat{C_2}(\beta)$.
4. *If* $\texttt{wfweak}(C_1)$, $\texttt{wfqvar}(C_1)$, $\mathit{fcv}(\alpha, \sigma) \subseteq \mathit{dom}(C_1)$, $\alpha \notin \widehat{C_1}(\sigma)$, $\mathit{doUpdateR}(C_1, \alpha, \sigma) = C_2$, *then* $\texttt{wfqvar}(C_2)$ *and if* $\Delta_{C_2}(\gamma) = \mathsf{m}$ *then there is a* $\beta \in \mathit{dom}(C_1)$ *such that* $\Delta_{C_1}(\beta) = \mathsf{m}$ *and* $\gamma \in \widehat{C_2}(\beta)$.
5. *If* $\texttt{wfweak}(C_1)$, $\texttt{wfqvar}(C_1, \varsigma)$, $\texttt{acyclic}(\varsigma)$, $\Delta_{C_1} \vdash^\mu \varsigma$, $\mathit{fcv}(\alpha, \varsigma) \subseteq \mathit{dom}(C_1)$, $\mathit{updateFlexi}(C_1, \alpha, \varsigma) = C_2$, *then it is the case that* $\texttt{wfqvar}(C_2)$ *and if* $\Delta_{C_2}(\gamma) = \mathsf{m}$ *then there is a* $\beta \in \mathit{dom}(C_1)$ *such that* $\Delta_{C_1}(\beta) = \mathsf{m}$ *and* $\gamma \in \widehat{C_2}(\beta)$.
6. *If* $\texttt{wfweak}(C_1)$,$\texttt{wfqvar}(C_1, \varsigma)$, $\texttt{acyclic}(\varsigma)$, $\Delta_{C_1} \vdash^\mu \varsigma$, $\mathit{fcv}(\alpha, \varsigma) \subseteq \mathit{dom}(C_1)$, $\alpha \notin \widehat{C_1}(\varsigma)$, *and moreover* $\mathit{doUpdateF}(C_1, \alpha, \varsigma) = C_2$, *then* $\texttt{wfqvar}(C_2)$ *and if* $\Delta_{C_2}(\gamma) = \mathsf{m}$ *then there is a* $\beta \in \mathit{dom}(C_1)$ *such that* $\Delta_{C_1}(\beta) = \mathsf{m}$ *and* $\gamma \in \widehat{C_2}(\beta)$.
7. *If* $\texttt{wfweak}(C_1)$, $\texttt{wfqvar}(C_1, \varsigma_1, \varsigma_2)$, $\texttt{acyclic}(\varsigma_1, \varsigma_2)$, $\Delta_{C_1} \vdash^\mu \varsigma_1, \varsigma_2$, $\mathit{fcv}(\varsigma_1, \varsigma_2) \subseteq \mathit{dom}(C_1)$, $\mathit{join}(C_1, \varsigma_1, \varsigma_2) = C_2, \varsigma$, *then* $\texttt{wfqvar}(C_2, \varsigma)$ *and if* $\Delta_{C_2}(\gamma) = \mathsf{m}$ *then there is a* $\beta \in \mathit{dom}(C_1)$ *such that* $\Delta_{C_1}(\beta) = \mathsf{m}$ *and* $\gamma \in \widehat{C_2}(\beta)$.

Proof. We prove all the cases simultaneously by induction on the number of recursive calls. For each case we assume that they all hold for sub-calls. In the recursive cases we make use of Lemma 6.33, Theorem 6.17, Theorem 6.23 to show the necessary well-formedness conditions. In the case of E4 we need to appeal to Lemma 6.35. All the rest cases are straightforward except for the cases for S2 and J3. We focus on these particular cases:

- Case S2. In this case we have that $\mathtt{wfweak}(C_1)$, $\mathtt{wfqvar}(C_1, \varsigma)$, $\mathtt{acyclic}(\varsigma)$, and $fcv(\varsigma, \sigma) \subseteq dom(C_1)$, where $\varsigma = [D] \Rightarrow \rho_1$ and $\sigma = \forall \overline{c}.\rho_2$. Moreover $subsCheck(C_1, [D] \Rightarrow \rho_1, \forall \overline{c}.\rho_2) = E_{\overline{\gamma}}$ where $\rho_3 = [\overline{c \mapsto b}]\rho_2$, $E = eqCheck(C_1 \bullet D, \rho_1, \rho_2)$ and $\overline{\gamma} = \widehat{E}(dom(C_1))$. We have that $\mathtt{wfweak}(C_1 \bullet D)$, $\mathtt{wfqvar}(C_1 \bullet D)$, $fcv(\rho_1, \rho_2) \subseteq dom(C_1 \bullet D)$. Hence by induction hypothesis we have that $\mathtt{wfqvar}(E)$ and consequently $\mathtt{wfqvar}(E_{\overline{\gamma}})$ as well. Now, pick a variable γ such that $\Delta_{E_{\overline{\gamma}}}(\gamma) = \mathsf{m}$. By induction hypothesis there exists a $\beta \in dom(C_1 \bullet D)$ such that $\Delta_{C_1 \bullet D}(\beta) = \mathsf{m}$ and $\gamma \in \widehat{E_{\overline{\gamma}}}(\beta)$. But $\mathtt{wfqvar}([D] \Rightarrow \rho_1)$ and hence $\Delta_{C_1}(\beta) = \mathsf{m}$, and the case is finished.
- Case J3. In this case we have that $\mathtt{wfweak}(C_1)$, $\mathtt{acyclic}([D_1] \Rightarrow \rho_1, [D_2] \Rightarrow \rho_2)$, $\mathtt{wfqvar}([D_1] \Rightarrow \rho_1, [D_2] \Rightarrow \rho_2)$ and $\mathtt{wfqvar}(C_1)$. Moreover $join(C_1, [D_1] \Rightarrow \rho_1, [D_2] \Rightarrow \rho_2) = E_{\widehat{E}(dom(C_1))}, [E_{\overline{\delta}}] \Rightarrow \rho_1$ where $E = eqCheck(C_1 \bullet D_1 \bullet D_2, \rho_1, \rho_2)$, $\overline{\delta} = \widehat{E}(\rho_1) - \widehat{E}(dom(C_1))$. First, we can confirm that $\mathtt{wfweak}(C_1 \bullet D)$, $\mathtt{wfqvar}(C_1 \bullet D_1 \bullet D_2)$, $fcv(\rho_1, \rho_2) \subseteq dom(C_1 \bullet D_1 \bullet D_2)$, Hence by induction hypothesis $\mathtt{wfqvar}(E)$ and consequently also $\mathtt{wfqvar}(E_{\widehat{E}(dom(C_1))})$. Next, we need to show that $\mathtt{wfqvar}([E_{\overline{\delta}}] \Rightarrow \rho_1)$. Hence, we have to show two things:
  - For all $\gamma \in dom(E_{\overline{\delta}})$, $\Delta_{E_{\overline{\delta}}} = \star$. Assume by contradiction that there exists a γ such that $\Delta_{E_{\overline{\delta}}}(\gamma) = \mathsf{m}$. It follows that $\Delta_E(\gamma) = \mathsf{m}$. By induction then there exists a β such that $\Delta_{C_1 \bullet D_1 \bullet D_2}(\beta) = \mathsf{m}$ such that $\gamma \in \widehat{E}(\beta)$. But it must be the case that $\Delta_{C_1}(\beta) = \mathsf{m}$ because $\mathtt{wfqvar}([D_1] \Rightarrow \rho_1, [D_2] \Rightarrow \rho_2)$, and hence $\gamma \notin \overline{\delta}$, a contradiction!
  - That $E_{\overline{\delta}}$ itself satisfies $\mathtt{wfqvar}(E_{\overline{\delta}})$. But we know that $\mathtt{wfqvar}(E)$ and $E_{\overline{\delta}}$ is only a restriction of E.

  To finish the case assume a variable ζ such that $\Delta_{E_{\widehat{E}(dom(C_1))}}(\zeta) = \mathsf{m}$. It follows that $\Delta_E(\zeta) = \mathsf{m}$ and by induction there exists a variable δ such that $\Delta_{C_1 \bullet D_1 \bullet D_2}(\delta) = \mathsf{m}$ such that $\zeta \in \widehat{E}(\delta)$. But, since $\mathtt{wfqvar}([D_1] \Rightarrow \rho_1, [D_2] \Rightarrow \rho_2)$ it must be the case that $\Delta_{C_1}(\delta) = \mathsf{m}$ and $\zeta \in \widehat{E_{\widehat{E}(dom(C_1))}}(\delta)$ as required.

□

Definition 6.38 (Well-formed scheme bodies)**.** Consider the relations $\Delta \vdash^b C$ and $\Delta \vdash^b \varsigma$ whenever:

$$\frac{(a)\ dom(C) \# dom(\Delta) \quad (b)\ \text{for all } (\alpha_\mu \geq \varsigma) \in C, \Delta\Delta_C \vdash^b \varsigma}{\Delta \vdash^b C}\ \text{CBWF} \qquad \frac{\Delta \vdash^b D \quad \text{if } \rho = \gamma \text{ then } (\gamma_\star\ \bot) \in D \text{ or } \Delta\Delta_D(\gamma) = \mathsf{m}}{\Delta \vdash^b [D] \Rightarrow \rho}\ \text{SBWF}$$

We write $\vdash^b C$ whenever $\emptyset \vdash^b C$.
Definition 6.39 (Well-formedness)**.** We write $\mathtt{wf}(C_1)$ iff $\mathtt{wfweak}(C_1) \wedge \mathtt{wfqvar}(C_1) \wedge \vdash^b C_1$.
Lemma 6.40. *If* $\mathtt{wf}(C_1)$, $fcv(\sigma_1) \subseteq dom(C_1)$, $mkMono(C_1, f, \sigma) = C_2$ *then* $\vdash^b C_2$.

Proof. Easy induction appealing to previous lemmas for *mkMono*: 6.11, 6.32, 6.15, 6.20. □

Lemma 6.41 (Well-formed scheme bodies preservation)**.** *The following are true:*

1. *If* $\mathtt{wf}(C_1)$, $fcv(\sigma_1, \sigma_2) \subseteq dom(C_1)$, $eqCheck(C_1, \sigma_1, \sigma_2) = C_2$ *then* $\vdash^b C_2$.
2. *If* $\mathtt{wf}(C_1)$, $\mathtt{acyclic}(\varsigma)$, $\mathtt{wfqvar}(\varsigma)$, $\Delta_{C_1} \vdash^{b/\mu} \varsigma$, $fcv(\varsigma, \sigma) \subseteq dom(C_1)$, $subsCheck(C_1, \varsigma, \sigma) = C_2$ *then* $\vdash^b C_2$.
3. *If* $\mathtt{wf}(C_1)$, $fcv(\alpha, \sigma) \subseteq dom(C_1)$, $updateRigid(C_1, \alpha, \sigma) = C_2$ *then* $\vdash^b C_2$.
4. *If* $\mathtt{wf}(C_1)$, $fcv(\alpha, \sigma) \subseteq dom(C_1)$, $\alpha \notin \widehat{C_1}(\sigma)$, $doUpdateR(C_1, \alpha, \sigma) = C_2$ *then* $\vdash^b C_2$.

5. *If* $\mathtt{wf}(C_1)$, $\mathtt{wfqvar}(\varsigma)$, $\mathtt{acyclic}(\varsigma)$, $\Delta_{C_1} \vdash^{b/\mu} \varsigma$, $fcv(\alpha, \varsigma) \subseteq dom(C_1)$, $updateFlexi(C_1, \alpha, \varsigma) = C_2$ *then* $\vdash^b C_2$.
6. *If* $\mathtt{wf}(C_1)$, $\mathtt{wfqvar}(\varsigma)$, $\mathtt{acyclic}(\varsigma)$, $\Delta_{C_1} \vdash^{b/\mu} \varsigma$, $fcv(\alpha, \varsigma) \subseteq dom(C_1)$, $\alpha \notin \widehat{C_1}(\varsigma)$, $doUpdateF(C_1, \alpha, \varsigma) = C_2$ *then* $\vdash^b C_2$.
7. *If* $\mathtt{wf}(C_1)$, $\mathtt{acyclic}(\varsigma_1, \varsigma_2)$, $\mathtt{wfqvar}(\varsigma_1, \varsigma_2)$, $\Delta_{C_1} \vdash^{b/\mu} \varsigma_1, \varsigma_2$, $fcv(\varsigma_1, \varsigma_2) \subseteq dom(C_1)$, $join(C_1, \varsigma_1, \varsigma_2) = C_2, \varsigma$ *then* $\vdash^b C_2$ *and* $\Delta_{C_2} \vdash^b \varsigma$.

Proof. The proof is by simulateneous induction on the number of recursive calls. For each case we assume that the corresponding case is true for the recursive subcalls. Most cases are straightforward, except for the cases of *extJoin*. The cases for J1 and J2 are straightforward. For J3 we have that $\mathtt{wf}(C_1)$, $\mathtt{acyclic}(\varsigma_1, \varsigma_2)$, $\mathtt{wfqvar}(\varsigma_1, \varsigma_2)$, $\Delta_{C_1} \vdash^{b/\mu} \varsigma_1, \varsigma_2$, $fcv(\varsigma_1, \varsigma_2) \subseteq dom(C_1)$ and $\varsigma_1 = [D_1] \Rightarrow \rho_1$, $\varsigma_2 = [D_2] \Rightarrow \rho_2$. We also have that $join(C_1, [D_1] \Rightarrow \rho_1, [D_2] \Rightarrow \rho_2) = E_{\widehat{E}(dom(C_1))}, [E_{\overline{\delta}}] \Rightarrow \rho_1$ where $E = eqCheck(C_1 \bullet D_1 \bullet D_2, \rho_1, \rho_2)$, $\overline{\delta} = \widehat{E}(\rho_1) - \widehat{E}(dom(C_1))$. By induction it is easy to show that $\vdash^b E$, and consequently $\vdash^b E_{\widehat{E}(dom(D_1))}$ (since it is a closed constraint). To finish the case we need to show that $\Delta_{E_{\widehat{E}(dom(D_1))}} \vdash^b [E_{\overline{\delta}}] \Rightarrow \rho_1$. First of all clearly $\Delta_{E_{\widehat{E}(dom(C_1))}} \vdash^b E_{\overline{\delta}}$. Assume now that $\rho_1 = \gamma$. Then, it must have been that either $(\gamma_\star \perp) \in D_1$, in which case we are done (because this branch of the algorithm cannot have been taken). Or it must be that $\Delta_{C_1}(\gamma) = \mathtt{m}$. In which case it must be by preservation of the monomorphic domain (Lemma 6.14) that also $\Delta_{E_{\widehat{E}(dom(C_1))}}(\gamma) = \mathtt{m}$, as is required to finish the case. □

Corollary 6.42. *If* $\vdash C_1$ $\mathtt{wf}$, $\Delta_{C_1} \vdash \sigma$, *and* $mkMono(C_1, f, \sigma) = C_2$ *then* $\vdash C_2$ $\mathtt{wf}$.

Proof. The proof is by putting together the following Lemmas or Theorems: 6.11, 6.15, 6.20, 6.32, and 6.40. □

Corollary 6.43 (Well-formedness preservation)**.** *The following are true:*

1. *If* $\vdash C_1$ $\mathtt{wf}$, $\Delta_{C_1} \vdash \sigma_1, \sigma_2$, *and* $eqCheck(C_1, \sigma_1, \sigma_2) = C_2$ *then* $\vdash C_2$ $\mathtt{wf}$.
2. *If* $\vdash C_1$ $\mathtt{wf}$, $\Delta_{C_1} \vdash \varsigma, \sigma$, $subsCheck(C_1, \varsigma, \sigma) = C_2$ *then* $\vdash C_2$ $\mathtt{wf}$.
3. *If* $\vdash C_1$ $\mathtt{wf}$, $\Delta_{C_1} \vdash \alpha, \sigma$, $updateRigid(C_1, \alpha, \sigma) = C_2$ *then* $\vdash C_2$ $\mathtt{wf}$.
4. *If* $\vdash C_1$ $\mathtt{wf}$, $\alpha \notin \widehat{C_1}(\sigma)$, $\Delta_{C_1} \vdash \alpha, \sigma$, *and* $doUpdateR(C_1, \alpha, \sigma) = C_2$ *then* $\vdash C_2$ $\mathtt{wf}$.
5. *If* $\vdash C_1$ $\mathtt{wf}$, $\Delta_{C_1} \vdash \alpha, \varsigma$, $updateFlexi(C_1, \alpha, \varsigma) = C_2$ *then* $\vdash C_2$ $\mathtt{wf}$.
6. *If* $\vdash C_1$ $\mathtt{wf}$, $\alpha \notin \widehat{C_1}(\varsigma)$, $\Delta_{C_1} \vdash \alpha, \varsigma$, *and* $doUpdateF(C_1, \alpha, \varsigma) = C_2$ *then* $\vdash C_2$ $\mathtt{wf}$.
7. *If* $\vdash C_1$ $\mathtt{wf}$, $\Delta_{C_1} \vdash \varsigma_1, \varsigma_2$, $join(C_1, \varsigma_1, \varsigma_2) = C_2, \varsigma_r$ *then* $\vdash C_2$ $\mathtt{wf}$ *and* $\Delta_{C_2} \vdash \varsigma_r$.

Proof. The proof is by putting together the following Lemmas or Theorems: 6.17, 6.23, 6.33, 6.37, and 6.41.

□

Up to this point we have shown that unification preserves our (rather complex) notion of well-formed constraints. For the rest of this section, we turn to the actual soundness of unification proofs, which relies—for the inductive cases to go through—on Corollary 6.42 and Corollary 6.43. We additionally make two extra assumptions:

1. In every constraint of the form $[D] \Rightarrow \rho$ encountered in the algorithm, all quantified variables of D ($dom(D)$) appear in the reachable variables $\widehat{D}(\rho)$ (i.e. there exist no useless quantified variables in schemes). Rule J3 which is the only rule that creates quantification clearly preserves this invariant.
2. An additional requirement is that no System F types with unused quantified variables (such as $\forall ab.\gamma$ or $\forall ab.a \to a$) are encountered during type inference, and that our substitutions never substitute for types with unused quantified variables.

Definition 6.44 (Logical constraint entailment)**.** We write $C_1 \vdash C_2$ iff for every $\theta \models C_1$ it is $\theta \models C_2$.

Lemma 6.45. *If* $\vdash C_1$ $\mathtt{wf}$, $\Delta_{C_1} \vdash \sigma$ *and* $mkMono(C_1, f, \sigma) = C_2$ *then* $C_2 \vdash C_1$. *Moreover* $\Delta_{C_2}(fcv(\sigma)) = \mathsf{m}$ *and if* $f = True$ *then* $\sigma = \tau$.

Proof. The second part follows from Lemma 6.32. We show the first goal by induction on the number of recursive calls to *mkMono*. Case M1 is straightforward. Case M2 follows by the induction hypothesis, Corollary 6.42 and Lemma 6.11. Case M3 and case M4 are straightforward. For M5 we have that $mkMono(C_1, f, \alpha) = E \leftarrow (\alpha_{\mathsf{m}} = \rho)$ given that $(\alpha_\mu \geq [D] \Rightarrow \rho) \in C_1$ and $E = mkMono(True, C_1 \bullet D, \rho)$. Assume that $\theta \models E \leftarrow (\alpha_{\mathsf{m}} = \rho)$. By separation consider the constaint $C_0 = (C_1 \bullet D)_{\widehat{C_1 \bullet D}(\rho)}$ and C_r the rest such that $C_0\, C_r = C_1 \bullet D$ and then $E = E_0\, C_r$. However it must be that $\alpha \in dom(C_r)$. Then $\theta \models E_0$ and by induction (the call with C_0 as the starting constraint takes the same number of recursive calls and clearly $\vdash C_0$ $\mathtt{wf}$) $\theta \models C_0$. Additionally $\theta \models C_r - \alpha$. Consequently $\theta \models C_0(C_r - \alpha)$, that is $\theta \models C_1 \bullet D - \alpha$. It follows (since $\alpha \notin dom(D)$) that $\theta \models D$. Then $\theta\alpha = \theta\rho \in [\![\theta([D^\circ] \Rightarrow \rho^\circ)]\!]$, where the $\circ$ symbol denotes a freshening of the domain of D and ρ to fresh constrained variables. That is because $\theta\alpha$ must be monomorphic, and hence it suffices to find a substitution $\psi \models \theta(D^\circ)$ and such that $\theta\rho = \psi\theta(\rho^\circ)$. But simply take $\psi = \theta^\circ_{dom(D)}$ (i.e. the restriction of θ to the $dom(D)$ variables, where we rename its domain accordingly. Hence, $\theta \models C_1 \bullet D$ and $\theta \models C_1$ as required. Case M6 is similar, and case M7 is straightforward. □

Lemma 6.46 (Satisfiability of well-formed constraints)**.** *If* $\vdash D$ $\mathtt{wf}$ *then there always exists a* θ *such that* $\theta \models D$.

Proof. Simply instantiate all flexible bounds to convert them to equalities. The result is an acyclic equality constraint which is always solvable by ordinary first-order unification (by first creating a unifier for the m-flagged variables). □

We prove the rest of the lemmas in this section simultaneously by induction on the number of the steps that the algorithm performs. For each lemma we assume that all other lemmas are true for any sub-calls of the algorithm.

Lemma 6.47. *If* $\vdash C_1$ $\mathtt{wf}$, $\Delta_{C_1} \vdash \sigma_1, \sigma_2$, $eqCheck(C_1, \sigma_1, \sigma_2) = C_2$, *for all* θ *such that* $\theta \models C_2$ *it is the case that* $\theta \models C_1$ *and* $\theta\sigma_1 = \theta\sigma_2$.

Proof. The case for E1 is straightforward. Case E2 and E3 follow from induction hypothesis for Lemma 6.48. For case E4 the result follows from induction hypothesis, Corollary 6.43, Lemma 6.12, and a second use of induction hypothesis. Case E5 gives us that $eqCheck(C_1, \forall\overline{a}\,.\,\rho_1, \forall\overline{a}\,.\,\rho_2) = C_2$ where we have that $C_2 = eqCheck(C_1, \overline{[a \mapsto b]}\rho_1, \overline{[a \mapsto b]}\rho_2)$ for some fresh $\overline{b}$ and additionally $\overline{b} \# C_2$. Assume that $\theta \models C_2$. Induction hypothesis gives $\theta \models C_1$ and $\theta\overline{[a \mapsto b]}\rho_1 = \theta\overline{[a \mapsto b]}\rho_2$. Let us assume without loss of generality that $\overline{a} \# ftv(\theta)$. Now we know that $\overline{b} \# C_2$ and $\theta \models C_2$. Let $\theta_0 = \overline{[b \mapsto d]}\theta$, where $\overline{d}$ completely fresh. By an easy renaming property we see that $\theta_0 \models C_2$. Then by induction hypothesis also $\theta_0\overline{[a \mapsto b]}\rho_1 = \theta_0\overline{[a \mapsto b]}\rho_2$, therefore $\forall\overline{b}\,.\,\theta_0\overline{[a \mapsto b]}\rho_1 = \forall\overline{b}\,.\,\theta_0\overline{[a \mapsto b]}\rho_2$. Now $\overline{b} \# ftv(\theta_0)$ and hence we have that $\theta_0(\forall\overline{b}\,.\,\overline{[a \mapsto b]}\rho_1) = \theta_0(\forall\overline{b}\,.\,\overline{[a \mapsto b]}\rho_2)$. Then, $\theta_0(\forall\overline{a}\,.\,\rho_1) = \theta_0(\forall\overline{a}\,.\,\rho_2)$. Finally since neither $\forall\overline{a}\,.\,\rho_1$ or $\forall\overline{a}\,.\,\rho_2$ contain $\overline{b}$ or $\overline{d}$, by another renaming we get: $\theta(\forall\overline{a}\,.\,\rho_1) = \theta(\forall\overline{a}\,.\,\rho_2)$, as required to finish the case. □

Lemma 6.48. *If* $\vdash C_1$ $\mathtt{wf}$, $\Delta_{C_1} \vdash \alpha, \sigma$, $updateRigid(C_1, \alpha, \sigma) = C_2$, *for all* θ *such that* $\theta \models C_2$ *it is the case that* $\theta \models C_1$ *and* $\theta\alpha = \theta\sigma$.

Proof. Case UR1 is trivial. For UR2 we have that $updateRigid(C_1, \alpha, \beta) = C_2$ where $(\beta_\mu = \gamma) \in C_1$ and $C_2 = updateRigid(C_1, \alpha, \gamma)$. Assume $\theta \models C_2$. By induction $\theta\alpha = \theta\gamma$ and $\theta \models C_1$. Since $(\beta_\mu = \gamma) \in C_1$ it must be that $\theta\beta = \theta\gamma$. It follows as well that $\theta\alpha = \theta\beta$. For case UR3 we have that $updateRigid(C_1, \alpha, \beta) = C_2$ where $(\beta_\mu \geq [D] \Rightarrow \gamma) \in C_1$, $\gamma \notin dom(D)$, and $C_2 = updateRigid(C_1, \alpha, \gamma)$. Assume that $\theta \models C_1$ and $\theta\alpha = \theta\gamma$. Moreover, since $\gamma \notin dom(D)$ we know that $\Delta_{C_1}(\gamma) = \mathsf{m}$. It follows that $\theta\gamma$ is monomorphic. Hence we have that $\theta\beta \in [\![[\theta D] \Rightarrow \theta\gamma]\!]$ (because θD must be empty). It follows that (assuming that our substitutions add no useless quantifiers) that $\theta\beta = \theta\gamma$. Consequently $\theta\alpha = \theta\beta$ as required. Case UR4 follows from the inductive hypothesis for Lemma 6.49. □

Lemma 6.49. *If* $\vdash C_1$ `wf`, $\Delta_{C_1} \vdash \alpha, \sigma$, $\alpha \notin \widehat{C_1}(\sigma)$, $doUpdateR(C_1, \alpha, \sigma) = C_2$, *for all* θ *such that* $\theta \models C_2$ *it is the case that* $\theta \models C_1$ *and* $\theta\alpha = \theta\sigma$.

Proof. We have the following cases to consider:

- Case DR1. In this case we have $doUpdateR(C_1, \alpha, \sigma) = C_2$ where $(\alpha_{\mathsf{m}}\ \bot) \in C_1$. Moreover $E = mkMono(True, C_1, \sigma)$ and $C_2 = E \leftarrow (\alpha_{\mathsf{m}} = \sigma)$. Assume that $\theta \models E \leftarrow (\alpha_{\mathsf{m}} = \sigma)$. It must be by the separation theorem however that $(\alpha_{\mathsf{m}}\ \bot) \in E$. Consequently, $\theta\alpha = \theta\sigma$ and $\theta \models E$. By Lemma 6.45, $\theta \models C_1$ and we are done.
- Case DR2. We have that $doUpdateR(C_1, \alpha, \sigma) = C_1 \leftarrow (\alpha_\star = \sigma)$ where $(\alpha_\star\ \bot) \in C_1$. Assume that $\theta \models C_1 \leftarrow (\alpha_\star = \sigma)$. It follows that $\theta \models C_1$ since $\theta\alpha$ can be any type in order to satisfy the $(\alpha_\star\ \bot)$ bound in C_1. Moreover $\theta\alpha = \theta\sigma$ as required.
- Case DR3. We have that $doUpdateR(C_1, \alpha, \sigma)$ where $(\alpha_\mu \geq \varsigma_a) \in C_1$, $E = extSubsCheck(C_1, \varsigma_a, \sigma)$, and $C_2 = E \leftarrow (\alpha_\mu = \sigma)$. By well-formedness of the constraint C_1 we know that $\mu = \star$. Assume that $\theta \models E \leftarrow (\alpha_\mu = \sigma)$. consider the restriction of C_1 to the reachable variables of ς_a, σ through C_1, call it C_0. The $C_1 = C_0\, C_r$ and $E = E_0\, C_r$ such that $E_0 = subsCheck(C_0, \varsigma_a, \varsigma)$ in the same number of steps (by Theorem 6.22). Since $\alpha \notin dom(E_0)$ it follows that $\theta \models E_0$ and by induction hypothesis $\theta \models C_0$ and $\theta\sigma \in [\![\theta\varsigma]\!]$. But also $\theta \models E - \alpha$, that is $\theta \models E_0(C_r - \alpha)$ and hence $\theta \models C_0(C_r - \alpha) = C_1 - \alpha$. But since $\theta\sigma \in [\![\theta\varsigma]\!]$ it follows that $\theta \models C_1$. Finally $\theta\alpha = \theta\sigma$, because $\theta \models E \leftarrow (\alpha_\mu = \sigma)$.
- Case DR4. We have that $doUpdateR(C_1, \alpha, \sigma) = C_2$ where $(\alpha_\mu = \sigma_a) \in C_1$ and $C_2 = eqCheck(C_1, \sigma_a, \sigma)$. Assume $\theta \models C_2$. By induction hypothesis for Lemma 6.47 it follows that $\theta \models C_1$ and $\theta\sigma_a = \theta\sigma$. Since $(\alpha_\mu = \sigma_a) \in C_1$ it must be that $\theta\alpha = \theta\sigma_a$. It follows that $\theta\alpha = \theta\sigma$ as required.

□

Lemma 6.50. *If* $\vdash C_1$ `wf` *and* $\Delta_{C_1} \vdash \varsigma, \sigma$, *and* $subsCheck(C_1, \varsigma, \sigma) = C_2$ *then for all* θ *such that* $\theta \models C_2$ *it is the case that* $\theta \models C_1$ *and* $\theta\sigma \in [\![\theta\varsigma]\!]$.

Proof. The case for S1 follows by induction hypothesis for Lemma 6.51. The case for S2 is the interesting one. Assume that $subsCheck(C_1, [D] \Rightarrow \rho_1, \forall \overline{c}\,.\,\rho_2) = C_2$ where $\rho_3 = [\overline{c \mapsto b}]\rho_2$, $E = eqCheck(C_1 \bullet D, \rho_1, \rho_3)$, $\overline{\gamma} = \widehat{E}(dom(C_1))$, $\overline{b} \# E_{\overline{\gamma}}$, and $C_2 = E_{\overline{\gamma}}$. By the well-formedness assumptions it must be that $\vdash C_1 \bullet D$ `wf` and moreover $C_1 \bullet D \vdash \rho_1, \rho_3$, assuming that $\overline{b}$ are fresh enough.

Assume now that $\theta \models E_{\overline{\gamma}}$. Consider the restriction of θ to $\overline{\gamma}$, call it $\theta_{\overline{\gamma}}$. Consider the permutation of $\theta_{\overline{\gamma}}$ such that maps $\overline{b}$ to completely fresh variables $\overline{d}$, call it $\theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}}$. It follows since $\overline{b} \# E_{\overline{\gamma}}$ that $\theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}} \models E_{\overline{\gamma}}$. Since every well-formed constraint is satisfiable (Lemma 6.46), there is an extension θ_1, such that $\theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}}\theta_1 \models E$. Then by induction hypothesis we get that $\theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}}\theta_1(\rho_1) = \theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}}\theta_1(\rho_3)$. Since $fcv(\rho_3) \in \overline{\gamma}$ this means that $\theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}}\theta_1(\rho_1) = \theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}}(\rho_3)$. However induction also gives that $\theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}}\theta_1 \models C_1 \bullet D$. Let $\theta_{\overline{\delta}} = \theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}}\theta_1\,|_{dom(D)}$. Then $\theta_{\overline{\delta}} \models \theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}}\theta_1(D)$. To finish the case we need to show that $\theta(\forall \overline{c}\,.\,\rho_2) \in [\![[\theta D] \Rightarrow \theta(\rho_1)]\!]$. We know that:

$$\theta_{\overline{\delta}}\theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}}\rho_1 = \theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}}\rho_3 \tag{13}$$

$$\overline{b} \# ([\theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}} D^\star] \Rightarrow \theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}}\rho_1^\star) \tag{14}$$

where $D^\star$ and $\rho_1^\star$ have their $\overline{\delta} = dom(D)$ renamed to fresh $\overline{\delta}^\star$. The second holds because of the choice of choosing fresh $\overline{b}$. Moreover, by simply renaming the $\overline{\delta}$ to $\overline{\delta}^\star$ in the first equation we get

$$\theta_{\overline{\delta}^\star}\theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}}\rho_1^\star = \theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}}\rho_3$$

Hence, it follows that

$$\forall \overline{b}\,.\,\theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}}[\overline{b \mapsto c}]\rho_2 \in [\![[\theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}} D^\star] \Rightarrow \theta_{\overline{\gamma}}^{\overline{b} \mapsto \overline{d}}\rho_1^\star]\!]$$

But $\forall \overline{b}\,.\,\theta^{\overline{b}\mapsto\overline{d}}_{\overline{\gamma}}[\overline{b \mapsto c}]\rho_2 = \theta^{\overline{b}\mapsto\overline{d}}(\forall \overline{c}\,.\,\rho_2)$ and $[\![[\theta^{\overline{b}\mapsto\overline{d}}_{\overline{\gamma}} D^\star] \Rightarrow \theta^{\overline{b}\mapsto\overline{d}}_{\overline{\gamma}}\rho_1^\star]\!] = [\![\theta^{\overline{b}\mapsto\overline{d}}([D] \Rightarrow \rho_1)]\!]$ and consequently: $\theta^{\overline{b}\mapsto\overline{d}}(\forall \overline{c}\,.\,\rho_2) \in [\![\theta^{\overline{b}\mapsto\overline{d}}([D] \Rightarrow \rho_1)]\!]$. By a renaming, because of the fresh choice of $\overline{b}$ of the algorithm, we get $\theta(\forall \overline{c}\,.\,\rho_2) \in [\![\theta([D] \Rightarrow \rho_1)]\!]$, as required. □

Lemma 6.51. *If* $\vdash C_1$ `wf`, $\Delta_{C_1} \vdash \varsigma, \alpha$, *and* $updateFlexi(C_1, \alpha, \varsigma) = C_2$ *then for all* θ *such that* $\theta \models C_2$ *it is the case that* $\theta \models C_1$ *and* $\theta\alpha \in [\![\theta\varsigma]\!]$.

Proof. Similar to the proof of Lemma 6.48, appealing to the inductive hypotheses for Lemma 6.48 and Lemma 6.52. □

Lemma 6.52. *If* $\vdash C_1$ `wf`, $\Delta_{C_1} \vdash \varsigma, \alpha$, $\alpha \notin \widehat{C_1}(\varsigma)$, *and* $doUpdateF(C_1, \alpha, \varsigma) = C_2$ *then for all* θ *such that* $\theta \models C_2$ *it is the case that* $\theta \models C_1$ *and* $\theta\alpha \in [\![\theta\varsigma]\!]$.

Proof. Similar to the proof of Lemma 6.49. The only interesting case is the call to the *join* function, which we describe next. We have that $doUpdateF(C_1, \alpha, \varsigma) = C_2$ where $(\alpha_\mu \geq \varsigma_a) \in C_1$, $E, \varsigma_r = join(C_1, \varsigma_a, \varsigma)$, and $C_2 = E \leftarrow (\alpha_\mu \geq \varsigma_r)$. Consider the split of C_1 to $C_0 C_r$ such that C_0 contains all the reachable variables through C_1 of ς_a and ς. By separation it must be that $join(C_0, \varsigma_a, \varsigma) = E_0, \varsigma_r$, where $E = E_0 C_r$. It must be that $\alpha \in dom(C_r)$, and also $\theta \models E_0$, and by induction $\theta \models C_0$. Consequently $\theta \models C_0 \bullet C_r - \alpha$ and hence $\theta \models C_1 - \alpha$. However by induction for Lemma 6.53 also $[\![\theta\varsigma_r]\!] \subseteq [\![\theta\varsigma]\!] \cap [\![\theta\varsigma_a]\!]$. Because $\theta\alpha \in [\![\theta\varsigma_r]\!] \subseteq [\![\theta\varsigma_a]\!]$ it must be that $\theta \models C_1$. Additionally $\theta\alpha \in [\![\theta\varsigma_r]\!] \subseteq [\![\theta\varsigma]\!]$, or $\theta\alpha \in [\![\theta\varsigma]\!]$ as required to finish the case. □

Lemma 6.53. *If* $\vdash C_1$ `wf`, $\Delta_{C_1} \vdash \varsigma_1, \varsigma_2$, *and* $join(C_1, \varsigma_1, \varsigma_2) = C_2, \varsigma$ *then for all* θ *such that* $\theta \models C_2$ *it is the case that* $\theta \models C_1$ *and* $[\![\theta\varsigma]\!] \subseteq [\![\theta\varsigma_1]\!] \cap [\![\theta\varsigma_2]\!]$.

Proof. We have the following cases to consider:

- $join(C_1, [D] \Rightarrow \alpha, \varsigma_2) = (C_1, \varsigma_2)$ whenever $(\alpha_\mu\ \bot) \in D$. We know moreover by well-formedness of $[D] \Rightarrow \alpha$ that $\mu = \star$, and $D = \emptyset$ (no useless quantified variables). Hence it is adequate to finish the case to show that $[\![\theta\varsigma]\!] \subseteq [\![[\theta D] \Rightarrow \alpha]\!]$. This will be the case if $[\![[\theta D] \Rightarrow \alpha]\!]$ is the set of all types, which is the case, since the θD can always be satisfied by any substitution, and in fact one that can assign any type σ to α (because $(\alpha_\star\ \bot) \in \theta D$).
- $join(C_1, \varsigma_1, [D] \Rightarrow \alpha) = (C_1, \varsigma_1)$ whenever $(\alpha_\mu\ \bot) \in D$. The case is similar to the previous one.
- $join(C_1, [D_1] \Rightarrow \rho_1, [D_2] \Rightarrow \rho_2) = C_2, \varsigma$, where $E = eqCheck(C_1 \bullet D_1 \bullet D_2, \rho_1, \rho_2)$, $\overline{\gamma} = \widehat{E}(dom(C_1))$, $\overline{\delta} = \widehat{E}(\rho_1) - \overline{\gamma}$, and $C_2 = E_{\overline{\gamma}}$, $\varsigma = [E_{\overline{\delta}}] \Rightarrow \rho_1$. First of all $C_1 \bullet D_1 \bullet D_2$ is well formed. Assume $\theta \models E_{\overline{\gamma}}$. Then there is an extension θ_1 such that $\theta\theta_1 \models E$. By induction $\theta\theta_1 \models C_1 \bullet D_1 \bullet D_2$ but since $dom(\theta) \supseteq dom(E_{\overline{\gamma}}) \supseteq dom(C_1)$ and C_1 is closed, it must be that $\theta \models C_1$ as required. Now consider θ_{D_1} and θ_{D_2} to be the restrictions of $\theta\theta_1$ to $dom(D_1)$ and $dom(D_2)$ respectively. Assume $\forall \overline{c}\,.\,\rho \in [\![\theta([E_{\overline{\delta}}] \Rightarrow \rho_1)]\!]$ hence $\forall \overline{c}\,.\,\rho \in [\![[\theta E^\star_{\overline{\delta}}] \Rightarrow \theta\rho_1^\star]\!]$ where $\overline{\delta}^\star$ and $\rho_1^\star$ are a suitable renaming of $\overline{\delta}$ to some fresh variables. Then it must be that:

$$\overline{c} \# [\theta E^\star_{\overline{\delta}}] \Rightarrow \theta\rho_1^\star \tag{15}$$
$$\rho = \theta^\star_{\overline{\delta}}\theta\rho_1^\star \tag{16}$$
$$\theta^\star_{\overline{\delta}} \models \theta E^\star_{\overline{\delta}} \tag{17}$$

  for some suitable $\theta^\star_{\overline{\delta}}$. We need to show that: $\forall \overline{c}\,.\,\rho \in [\![[\theta D_1^\circ] \Rightarrow \theta\rho_1^\circ]\!]$ where D_1° and ρ_1° are completely fresh renamings of the domain of D_1. We may pick from the beginning $\overline{c}$ to be fresh from $[\![[\theta D_1^\circ] \Rightarrow \theta\rho_1^\circ]\!]$. Consider now the substitution $\theta^\circ_{D_1}$. It must be that $\theta^\circ_{D_1} \models \theta D_1^\circ$. Moreover $\theta^\circ_{D_1}\theta(\rho_1^\circ) = \theta^\star_{\overline{\delta}}\theta\rho_1^\star = \rho$. Hence $[\![\theta\varsigma]\!] \subseteq [\![\theta([D_1] \Rightarrow \rho_1)]\!]$. For showing $[\![\theta\varsigma]\!] \subseteq [\![\theta([D_2] \Rightarrow \rho_2)]\!]$, it suffices to show that $\forall \overline{c}\,.\,\rho \in [\![[\theta D_2^\circ] \Rightarrow \theta\rho_2^\circ]\!]$ where D_2° and ρ_2° are completely fresh renamings of the domain of D_2. We may pick from the beginning $\overline{c}$ to be fresh from $[\![[\theta D_2^\circ] \Rightarrow \theta\rho_2^\circ]\!]$. Consider now the substitution $\theta^\circ_{D_2}$. It must be that $\theta^\circ_{D_2} \models \theta D_2^\circ$. To conclude we must show that $\theta^\circ_{D_2}\theta(\rho_2^\circ) = \rho$. It suffices to show that $\theta^\circ_{D_2}\theta(\rho_2^\circ) = \theta^\star_{\overline{\delta}}\theta\rho_1^\star$ which holds since $\theta\theta_1\rho_1 = \theta\theta_1\rho_2$ by induction.

□

6.3.3 Unification completeness lemmas

We proceed with showing the completeness of unification with respect to the set semantics that we have defined earlier.

Lemma 6.54. *If* $\vdash C_1$ `wf`, $\Delta_{C_1} \vdash \sigma_1, \sigma_2$, $\theta \models C_1$, *with* $dom(\theta) = dom(C_1)$, $\theta\sigma_1 = \theta\sigma_2$, *and eqCheck terminates, then* $eqCheck(C_1, \sigma_1, \sigma_2) = C_2$ *such that there exists a* θ_r *with* $\theta\theta_r \models C_2$ *and* $dom(\theta\theta_r) = dom(C_2)$.

Proof. The proof is by induction on termination relation for *eqCheck*. Let us consider cases on σ_1 and σ_2. If both types are (rigid) type variables, then it must be that $\sigma_1 = a$ and $\sigma_2 = a$, and consequently case E1 is applicable. If σ_1 or σ_2 is α then, case E2 or case E3 is applicable , and the result follows by Lemma 6.56. If none of σ_1 or σ_2 are variables, assume that they are function types: $\sigma_1 = \sigma_{11} \to \sigma_{21}$ and $\sigma_2 = \sigma_{12} \to \sigma_{22}$. Then we have that $\theta\sigma_{11} = \theta\sigma_{12}$ and $\theta\sigma_{21} = \theta\sigma_{22}$. Hence by induction hypothesis we have that $eqCheck(C_1, \sigma_{11}, \sigma_{21}) = E$ such that there exists a θ_r^1 with $\theta\theta_r^1 \models E$. Moreover by the well-formedness lemmas we know that E is well formed, and moreover $\Delta_E \vdash \sigma_{21}, \sigma_{22}$. By induction hypothesis again $eqCheck(E, \sigma_{21}, \sigma_{22}) = C_2$, and there exists a θ_r^2 such that $\theta\theta_r^1\theta_r^2 \models C_2$ as required. The final case we have to consider is when $\sigma_1 = \forall \overline{a}\,.\,\rho_1$ and $\sigma_2 = \forall \overline{a}\,.\,\rho_2$, and without loss of generality assume that $\rho_1 \neq \gamma_1$ and $\sigma_2 \neq \gamma_2$ because we assume that there exist no useless quantified variables in types. Also without loss of generality assume that $\overline{a}\overline{b}\#ftv(\theta)$ (i.e. we picked the fresh variables, fresh from θ as well). Then $\theta\sigma_1 = \forall \overline{a}\,.\,\theta\rho_1$ and $\theta\sigma_2 = \forall \overline{a}\,.\,\theta\rho_2$. It follows that $\theta\rho_1 = \theta\rho_2$. Consequently by induction $eqCheck(C_1, [\overline{a \mapsto b}]\rho_1, [\overline{a \mapsto b}]\rho_2) = C_2$ such that there exists a θ_r so that $\theta\theta_r \models C_2$. To finish the case we need to show that $\overline{b}\#E$. But observe that it suffices to show that $\overline{b}\#E_{\widehat{E}(dom(C_1))}$ (by the separation lemma, the rest of the constraint E is also present in C_1 and not touched, and hence cannot have contained $\overline{b}$). If there were some $b \in \overline{b}$ in $E_{\widehat{E}(dom(C_1))}$ it must have been that $b \in range(\theta)$, a contradiction. Case E6 is not applicable, because the types cannot be equal. □

Lemma 6.55. *If* $\vdash C_1$ `wf`, $\Delta_{C_1} \vdash \varsigma, \sigma$, *and* $\theta \models C_1$ *with* $dom(\theta) = dom(C_1)$, $\theta\sigma \in [\![\theta\varsigma]\!]$ *and subsCheck terminates, then* $subsCheck(C_1, \varsigma, \sigma) = C_2$ *then there exists a* θ_r *such that* $\theta\theta_r \models C_2$ *and* $dom(\theta\theta_r) = dom(C_2)$.

Proof. Let us do a case analysis on σ.

- $\sigma = \beta$. In this case the result follows by Lemma 6.58.
- $\sigma \neq \beta$. In this case $\sigma = \forall \overline{c}\,.\,\rho_2$ where we assume that ρ_2 is not a variable. Since *subsCheck* terminates, then it must also be that $eqCheck(C_1 \bullet D, \rho_1, \rho_3)$ terminates where $\rho_3 = [\overline{c \mapsto b}]\rho_2$. Since $\theta(\forall \overline{c}\,.\,\rho_2) = [\![[\theta D] \Rightarrow \theta\rho]\!]$. We assume without loss of generality that $\overline{b}\#ftv(\theta)$ (i.e. we picked the variables $\overline{b}$ fresh from θ as well). Then there must be a substitution θ_D such that $\theta_D \models \theta D$ such that $\theta_D\theta\rho = \theta\rho_2$ (because $\theta\rho_2$ must be a ρ-type when ρ_2 is not a variable), and moreover $\overline{c}\#ftv([\theta D] \Rightarrow \theta\rho)$. It follows that $\theta\theta_D \models C_1 \bullet D$. By Lemma 6.54, since $eqCheck(C_1 \bullet D, \rho_1, \rho_3)$ terminates, there exists a θ_r such that $\theta\theta_D\theta_r \models E$. Finally consider the restriction of $\theta_D\theta_r$, call it $\theta_r^\star$, such that $\theta\theta_r^\star \models E_{\overline{\gamma}}$. To finish the case we need to show that $\overline{b}\#E_{\overline{\gamma}}$. This follows, because if $\overline{b} \in E_{\overline{\gamma}}$ it must be that $\overline{b} \in range(\theta)$, which is impossible.

□

Lemma 6.56. *If* $\vdash C_1$ `wf`, $\Delta_{C_1} \vdash \alpha, \sigma$, $\theta \models C_1$, *with* $dom(\theta) = dom(C_1)$, $\theta\alpha = \theta\sigma$, *and updateRigid terminates, then* $updateRigid(C_1, \alpha, \sigma) = C_2$ *such that there exists a* θ_r *with* $\theta\theta_r \models C_2$ *and* $dom(\theta\theta_r) = dom(C_2)$.

Proof. We consider cases on σ. If $\sigma = \beta$ we have the following cases:

- We have $\beta = \alpha$. The case finishes by taking θ_r to be the identity substitution.
- We have $\beta \neq \alpha$ and $(\beta_\mu = \gamma) \in C$. It follows that case UR2 is reachable. However since $\theta\alpha = \theta\beta$ and $\theta \models C$, it must be that $\theta\alpha = \theta\gamma$. Induction hypothesis finishes the case.

- We have $\beta \neq \alpha$ and $(\beta_\mu \geq [D] \Rightarrow \rho_1)$. Moreover we know that $\theta\alpha = \theta\beta$ and $\theta\beta \in [\![[\theta D] \Rightarrow \theta\rho_1]\!]$. We have the following cases for ρ_1. if $\rho_1 = \gamma$ then one case is if $(\gamma_\star \bot) \in D$. In that case, case UR4 is reachable and the result follows by Lemma 6.57. The other case is that $\Delta_{C_1}(\gamma) = \mathsf{m}$. In this case, case UR3 is reachable (and also $D = \emptyset$). Then, since $\theta\beta \in [\![[\emptyset] \Rightarrow \theta\gamma]\!]$ and there are no useless quantifiers, we have that $\theta\beta = \theta\gamma$ (and is monomorphic). But then it must be that $\mathit{updateRigid}(C_1, \alpha, \gamma)$ terminates, and hence by induction hypothesis we are done. Now, let us consider the cases where $\rho_1 \neq \gamma$. It must be in that case that $\alpha \notin \widehat{C_1}(\beta)$ otherwise it can't happen that $\theta\alpha = \theta\beta \in [\![[\theta D] \Rightarrow \theta\rho_1]\!]$. Hence, case UR4 is reachable and we get the result by Lemma 6.57.

Now, assume that $\sigma \neq \beta$. It must be the case that $\alpha \notin \widehat{C_1}(\sigma)$, because otherwise it cannot be that $\theta\alpha = \theta\sigma$. Then the case UR4 is reachable and Lemma 6.57 shows the goal. □

Lemma 6.57. *If* $\vdash C_1$ $\mathtt{wf}$, $\Delta_{C_1} \vdash \alpha, \sigma$, $\theta \models C_1$, *with* $dom(\theta) = dom(C_1)$, $\alpha \notin \widehat{C_1}(\sigma)$, $\theta\alpha = \theta\sigma$, *and doUpdateR terminates, then* $doUpdateR(C_1, \alpha, \sigma) = C_2$ *such that there exists a* θ_r *with* $\theta\theta_r \models C_2$ *and* $dom(\theta\theta_r) = dom(C_2)$.

Proof. The proof is by case analysis on the bound of α in C_1. If $(\alpha_m \bot) \in C_1$ and $\theta\alpha = \theta\sigma$, and moreover $\theta\alpha$ is monomorphic. By completeness for *mkMono*, we have that there exists a θ_r^1 such that $\theta\theta_r^1 \models E$. It follows that $\theta\theta_r^1 \models E \leftarrow (\alpha_\mathsf{m} = \sigma)$. If on the other hand $(\alpha_\star \bot) \in C_1$ we have as before that $\theta\alpha = \theta\sigma$ and $\theta \models C_1$, so it must also be that $\theta \models C_1 \leftarrow (\alpha_\star = \sigma)$. Now, consider the case where $(\alpha_\star \geq \varsigma_a) \in C_1$. We know that $\theta\alpha = \theta\sigma$ and moreover $\theta\alpha \in [\![\theta\varsigma_a]\!]$. It follows that case DR3 is reachable and by Lemma 6.55 we have that $\mathit{subsCheck}(C_1, \varsigma_a, \sigma) = E$ and there exists a θ_r such that $\theta\theta_r \models E$. Since $\theta\alpha = \theta\sigma$ this implies also that $\theta\theta_r \models E \leftarrow (\alpha_\star = \sigma)$. Finally assume that $(\alpha_\mu = \sigma_a) \in C_1$. Moreover we have that $\theta\alpha = \theta\sigma$ and since $\theta \models C_1$ we have $\theta\alpha = \theta\sigma_a$. Then case DR4 is reachable and then by Lemma 6.54 the case is finished. □

Lemma 6.58. *If* $\vdash C_1$ $\mathtt{wf}$, $\Delta_{C_1} \vdash \alpha, \varsigma$, $\theta \models C_1$, *with* $dom(\theta) = dom(C_1)$, $\theta\alpha \in [\![\theta\varsigma]\!]$, *and updateFlexi terminates, then* $updateFlexi(C_1, \alpha, \varsigma) = C_2$ *such that there exists a* θ_r *with* $\theta\theta_r \models C_2$ *and* $dom(\theta\theta_r) = dom(C_2)$.

Proof. Similar to the proof of Lemma 6.56, appealing to Lemma 6.56, and Lemma 6.59. (Observe that if $\varsigma = [D] \Rightarrow \gamma$ then by well-formedness γ is either a monomorphic variable, or bound to $\bot$ in D, hence the call to *updateRigid* in UF1). □

Lemma 6.59. *If* $\vdash C_1$ $\mathtt{wf}$, $\Delta_{C_1} \vdash \alpha, \varsigma$, $\theta \models C_1$, $\alpha \notin \widehat{C_1}(\alpha)$, *with* $dom(\theta) = dom(C_1)$, $\theta\alpha \in [\![\theta\varsigma]\!]$, *and doUpdateF terminates, then* $doUpdateF(C_1, \alpha, \varsigma) = C_2$ *such that there exists a* θ_r *with* $\theta\theta_r \models C_2$ *and* $dom(\theta\theta_r) = dom(C_2)$.

Proof. Similar to the proof of Lemma 6.57, by analysis to the bound of α in C_1, and appealing to Lemma 6.55. The interesting case is when $(\alpha_\star \geq \varsigma_a) \in C_1$. In this case we know that $\theta\alpha \in [\![\theta\varsigma]\!]$ and $\theta\alpha \in [\![\theta\varsigma_a]\!]$. Then we are in case DF4 and it follows by Lemma 6.60 that $\mathit{join}(C_1, \varsigma, \varsigma_a) = E$, and there exists a θ_r such that $\theta\theta_r \models E$. From the same lemma it is the case that $\theta\alpha \in [\![\theta\theta_r\varsigma_r]\!]$. It follows that $\theta\theta_r \models E \leftarrow (\alpha_\star \geq \varsigma_r)$. □

Lemma 6.60. *If* $\vdash C_1$ $\mathtt{wf}$, $\Delta_{C_1} \vdash \varsigma_1, \varsigma_2$, $\theta \models C_1$ *with* $dom(\theta) = dom(C_1)$, *and* $\mathit{join}(C_1, \varsigma_1, \varsigma_2)$ *terminates and* $\sigma \in [\![\theta\varsigma_1]\!] \cap [\![\theta\varsigma_2]\!]$, *then* $\mathit{join}(C_1, \varsigma_1, \varsigma_2) = E, \varsigma$ *such that there exists a* θ_r *with* $\theta\theta_r \models E$ *and* $dom(\theta\theta_r) = \theta(E)$, *and* $\sigma \in [\![\theta\theta_r\varsigma]\!]$.

Proof. Let us consider cases on ς_1 and ς_2. If one of them is $[\alpha_\star \bot] \Rightarrow \alpha$ then either the case of J1 or J2 is reachable and, since the set $[\alpha_\star \bot] \Rightarrow \alpha$ is the set of all types we are done. Now, let us assume that this is not the case. The reachable clause is then J3. In this case we have that $\theta \models C_1$ and the call is made to $\mathit{join}(C_1, [D_1] \Rightarrow \rho_1, [D_2] \Rightarrow \rho_2)$. We have that $\sigma \in [\![[\theta D_1] \Rightarrow \theta\rho_1]\!]$ and $\sigma \in [\![\theta D_2] \Rightarrow \theta\rho_2]\!]$. Assume that $\sigma = \forall \overline{c}\,.\,\rho$. It must be hence that $\overline{c} \# ftv([\theta D_1] \Rightarrow \theta\rho_1, [\theta D_2] \Rightarrow \theta\rho_2)$ and there exist θ_{D_1} and θ_{D_2} such that $\theta_{D_1} \models \theta D_1$ and $\theta_{D_2} \models \theta D_2$. Without loss of generality assume

that D_1 and D_2 have fresh domains, and it also must be that $\rho = \theta_{D_1}\theta\rho_1 = \theta_{D_2}\theta\rho_2$. Hence it must be that $\theta\theta_{D_1}\theta_{D_2} \models C_1 \bullet D_1 \bullet D_2$, and

$$\theta\theta_{D_1}\theta_{D_2}\rho_1 = \theta\theta_{D_1}\theta_{D_2}\rho_2 = \rho \tag{18}$$

Hence, by Lemma 6.54 $eqCheck(C_1 \bullet D_1 \bullet D_2, \rho_1, \rho_2) = E$ such that there exists a θ_r with $\theta\theta_{D_1}\theta_{D_2}\theta_r \models E$. Consider the restriction of $\theta_{D_1}\theta_{D_2}\theta_r$ to $E_{\widehat{E}(dom(C_1))}$ to be $\theta^\star_r$ and then $\theta\theta^\star_r \models E_{\widehat{E}(dom(C_1))}$. If $\overline{\delta} = \widehat{E}(\rho_1) - \widehat{E}(dom(C_1))$ then it is also the case that $\theta\theta_{D_1}\theta_{D_2}\theta_r \models E_{\overline{\delta}}$ and equation (18) finishes the case. □

6.3.4 Main algorithm soundness

We start by showing some well-formedness preservation lemmas about the algorithm of Figure 11.

Lemma 6.61 (Instantiation well-formedness)**.** *The following properties are true:*

1. *If* $inst(C_1, \sigma) = C_2, \rho$ *then* $dom(C_1) \subseteq dom(C_2)$ *and if* $\Delta_{C_1}(\gamma) = \mathsf{m}$ *then* $\Delta_{C_2}(\gamma) = \mathsf{m}$.
2. *If* $\mathtt{wfweak}(C_1)$ *and* $inst(C_1, \sigma) = C_2, \rho$ *and* $\Delta_C(\gamma) = \mathsf{m}$ *then* $\widehat{C_1}(\gamma) \subseteq \widehat{C_2}(\gamma)$.
3. *If* $\mathtt{wfweak}(C_1)$, $\mathtt{wfqvar}(C_1)$, $fcv(\sigma) \subseteq dom(C_1)$, $inst(C_1, \sigma) = C_2, \rho$ *then* $\mathtt{wfqvar}(C_2)$ *and if* $\Delta_{C_2}(\gamma) = \mathsf{m}$ *then there is a* $\beta \in dom(C_1)$ *such that* $\Delta_{C_1}(\beta) = \mathsf{m}$ *and* $\gamma \in \Delta_{C_2}(\beta)$.

Proof. Easy check. □

Lemma 6.62 (Instantiation well-formedness)**.** *If* $\vdash C_1\ \mathtt{wf}$ *and* $\Delta_{C_1} \vdash \sigma$ *and* $inst(C_1, \sigma) = C_2, \rho$ *then* $\vdash C_2\ \mathtt{wf}$ *and* $C_2 \vdash \rho$.

Proof. Easy check. □

Lemma 6.63 (Instantiation soundness)**.** *If* $\vdash C_1\ \mathtt{wf}$ *and* $\Delta_{C_1} \vdash \sigma$, *and* $inst(C_1, \sigma) = C_2, \rho$, *then for all* $\theta \models C_2$, $\theta \models C_1$ *and* $\vdash^{\mathsf{inst}} \theta[\sigma] \leq \theta[\rho]$.

Proof. It must be that $\sigma = \forall \overline{a}\,.\,\rho_1$ in which case $C_2 = C_1 \bullet (\overline{\alpha}_\star\ \bot)$ and $\rho = [\overline{a \mapsto \alpha}]\rho_1$. Assume that $\theta \models C_2$. It follows that $\theta \models C_1$. Moreover let θ_o, $\theta_{\overline{\alpha}}$ be the split of θ such that $dom(\theta_{\overline{\alpha}}) = \overline{\alpha}$. Then $\theta[\sigma] = \forall \overline{a}\,.\,\theta_0[\rho_1]$ where without loss of generality we assume that $\overline{a} \# ftv(\theta)$. But then

$$\vdash^{\mathsf{inst}} \forall \overline{a}\,.\,\theta_0[\rho_1] \leq [\overline{a \mapsto \theta_{\overline{\alpha}}[\alpha]}]\theta_0[\rho_1] = \theta[([\overline{a \mapsto \alpha}]\rho_1)] = \theta\rho$$

as required. □

The following important lemma ensures that our normalization procedure produces well-formed schemes.

Lemma 6.64 (Normalization well-formedness)**.** *Assume that:*

1. $\Delta \vdash C\ \mathtt{wf}$,
2. $\Delta\Delta_C \vdash \sigma$,
3. *for all* $\alpha \in dom(C)$, $\Delta_C(\alpha) = \star$, *and there exists a* $\beta \in fcv(\sigma)$ *such that* $\alpha \in \widehat{C}(\sigma)$, *and*
4. $normalize(C, \sigma) = [E] \Rightarrow \rho$

Then, $\Delta \vdash [E] \Rightarrow \rho$.

Proof. We prove the lemma by induction on the number of recursive calls. We have the following cases to consider:

- Case N1. We have that $\mathit{normalize}(C, \alpha) = [C\bullet D - \alpha] \Rightarrow \rho$. Hence it must also be that $\Delta \vdash C\bullet D$ $\mathtt{wf}$, and because no variable from the domain of C points to α, and C and D are acyclic it follows that $\Delta \vdash C\bullet D - \alpha$ $\mathtt{wf}$. The interesting part is to verify two conditions, (i) the domain of $E = C\bullet D - \alpha$ does not contain any m-bound variables, and (ii) if $\rho = \gamma$ it is either bound to $\bot$ in E or monomorphic in the environment Δ. For (i), observe first that, because $\Delta \vdash [D] \Rightarrow \rho$, D does not contain any m-bound variables in its domain, and so does C by assumptions. It follows that $C\bullet D$ does not contain any m-bound variables, and so does $C\bullet D - \alpha$. For (ii) let us examine the case where $\rho = \gamma$, If $\gamma \in \mathit{dom}(D)$, since $\Delta \vdash [D] \Rightarrow \rho$ it must be that $(\gamma_\star\ \bot) \in \mathit{dom}(D)$ which implies that $(\gamma_\star\ \bot) \in \mathit{dom}(C\bullet D)$ and clearly $\gamma \neq \alpha$. On the other hand if $\Delta(\gamma) = \mathsf{m}$, $\gamma \notin \mathit{dom}(C)$ and $\gamma \notin \mathit{dom}(D)$ and hence $\gamma \notin \mathit{dom}(C\bullet D)$ and hence the case is done.
- Case N2 is similar to N1 appealing additionally to the induction hypothesis.
- Case N3. We have in this case that $\mathit{normalize}(C, \forall \overline{a}\,\mathtt{.}\,\rho) = \mathit{normalize}(C\bullet(\overline{\alpha}_\star\ \bot), [\overline{a \mapsto \alpha}]\rho)$. We can assume without loss of generality that $\overline{\alpha}$ were picked fresh from Δ as well, and consequently $\Delta \vdash C\bullet(\overline{\alpha}_\star\ \bot)$. Moreover by assumptions $C\bullet(\overline{\alpha}_\star\ \bot)$ does not contain any m-bound variables. by induction then the case is finished.
- Case N4. In this case we have that either ρ is not a variable in which case the well-formedness follows easily. If on the other hand $\rho = \gamma$, since we know that, since we are in this case it must be that
  - either $(\gamma_\mu\ \bot) \in C$, and since C does not contain any m-bound variables, it is $(\gamma_\star\ \bot) \in C$,
  - or $\gamma \in \mathit{fcv}(\sigma)$ and $\gamma \notin \mathit{dom}(C)$. But in this case $\Delta(\gamma) = \mathsf{m}$ by assumptions.

  It follows in every case that $\Delta \vdash [E] \Rightarrow \rho$, as required.

□

Lemma 6.65 (Normalization soundness)**.** *Assume that:*

1. $\vdash C$ $\mathtt{wf}$,
2. $\Delta_C \vdash C_1$ $\mathtt{wf}$,
3. $\Delta\Delta_{C_1} \vdash \sigma$,
4. *for all* $\alpha \in \mathit{dom}(C_1)$, $\Delta_{C_1} = \star$, *and there exists a* $\beta \in \mathit{fcv}(\sigma)$ *such that* $\alpha \in \widehat{C_1}(\sigma)$, *and*
5. $\mathit{normalize}(C_1, \sigma) = [E] \Rightarrow \rho$

Then, for all $\theta \models C\bullet E$ *there exists a* θ_r *such that* $\theta\theta_r \models C\bullet C_1$ *and* $\vdash^{\mathsf{F}} \theta\theta_r\sigma \leq \theta\rho$.

Proof. We show the case by induction on the number of recursive calls of *normalize*. In case N1 we have that $\mathit{normalize}(C_1, \alpha) = [C_1\bullet D - \alpha] \Rightarrow \rho$ when $(\alpha_\star \geq [D] \Rightarrow \rho) \in C_1$. Assume that $\theta \models C_1\bullet D - \alpha$. Then it must be that $\theta_D \models \theta^{-D} D$, ($\theta^{-D}$ being the restriction of θ to its domain that is disjoint from D), it suffices then to choose $\theta_r(\alpha) = \theta\rho$ and we are done. In case N2 we have that $\mathit{normalize}(C_1, \alpha) = \mathit{normalize}(C_1 - \alpha, \sigma)$ whenever $(\alpha_\star = \sigma) \in C_1$. Assume that $\mathit{normalize}(C_1 - \alpha, \sigma) = [E] \Rightarrow \rho$ and let $\theta \models C\bullet E$, then there exists a θ^1_r such that $\theta\theta^1_r \models C\bullet(C_1 - \alpha)$. Let $\theta^2_r(\alpha) = \theta\theta^1_r\sigma$. It follows that $\theta\theta^1_r\theta^2_r \models C\bullet C_1$ as required. Moreover we know that $\vdash^{\mathsf{F}} \theta\theta^1_r\sigma \leq \theta\rho$, which implies $\vdash^{\mathsf{F}} \theta\theta^1_r\theta^2_r(\alpha) \leq \theta\rho$. In case N3 we have that $\mathit{normalize}(C_1, \forall\overline{a}\,\mathtt{.}\,\rho) = \mathit{normalize}(C_1\bullet(\overline{\alpha}_\star\ \bot), [\overline{a \mapsto \alpha}]\rho) = [E] \Rightarrow \rho$. Assume that $\theta \models C\bullet E$, by induction there exists a θ_r such that $\theta\theta_r \models C\bullet C_1\bullet(\overline{\alpha}_\star\ \bot)$, and $\vdash^{\mathsf{F}} \theta\theta_r[\overline{a \mapsto \alpha}]\rho \leq \theta\rho$. It follows that $\vdash^{\mathsf{F}} \theta\theta_r(\forall\overline{a}\,\mathtt{.}\,\rho) \leq \theta\rho$. Case N4 is straightforward. □

Lemma 6.66 (Generalization well-formedness)**.** *If* $\vdash C$ $\mathtt{wf}$, $C \vdash \Gamma, \rho$, *and* $\mathit{generalize}(C, \Gamma, \rho) = [E_g] \Rightarrow \rho_g, E$ *then* $\vdash E$ $\mathtt{wf}$, $E \vdash \Gamma$, *and* $E \vdash [E_g] \Rightarrow \rho_g$.

Proof. Unfolding of the definitions, appealing to Lemma 6.64 □

Lemma 6.67 (Generalization soundness)**.** *Assume that* $\vdash C$ $\mathtt{wf}$, $C \vdash \Gamma$, $C \vdash \rho$ *and* $\mathit{generalize}(C, \Gamma, \rho) = [E_g] \Rightarrow \rho_g, E$, *and* $\theta \models E$, *then, for all* θ_g *such that* $\theta\theta_g \models E\bullet E_g$ *there exists a* θ_r *such that* $\theta\theta_g\theta_r \models C$ *and* $\vdash^{\mathsf{F}} (\theta\theta_g\theta_r\rho) \leq \theta\theta_g\rho_g$.

Proof. Let $\overline{\beta} = \widehat{C}(\rho) - \widehat{C}(\Gamma)$. Then $[E_g] \Rightarrow \rho_g = \mathit{normalize}(C_{\overline{\beta}}, \rho)$, and $E = C - C_{\overline{\beta}}$. Assume that $\theta \models C - C_{\overline{\beta}}$, and asume that θ_g is such that $\theta\theta_g \models (C - C_{\overline{\beta}}) \bullet E_g$. By Lemma 6.65 there exists a θ_r such that $\theta\theta_g\theta_r \models (C - C_{\overline{\beta}}) \bullet C_{\overline{\beta}}$, that is, $\theta\theta_g\theta_r \models C$. Moreover it is the case that $\vdash^{\mathsf{F}} \theta\theta_g\theta_r\rho \leq \theta\theta_g\rho_g$ as required. □

Given the fact that unification, normalization, and generalization preserve well-formedness of constraints and preserve the sets of monomorphic variables, it is an easy check that functions *instFun*, *instMono*, and *infer* have the same properties. In the rest of this section we assume domain monotonicity, preservation of well-formedness and origination of all monomorphic variables in some monomorphic variable appearing in the environment. Establishing this property is a tedious but straightforward task.

Below we give soundness of the functions *instFun*, *instMono*, and *infer*.

Lemma 6.68. *If* $\vdash C_1$ `wf`, $\Delta_{C_1} \vdash \rho$, $C_1 \vdash \Gamma$, *and* $\mathit{instFun}(C_1, \Gamma, \rho) = C_2, \sigma_1 \to \sigma_2$ *then for every* $\theta \models C_2$, *there exists a* θ_r *such that* $\theta\theta_r \models C_1$ *and* $\theta\theta_r[\sigma] \preceq\sqsubseteq^{\rightarrow} \theta[\sigma_1] \to \theta[\sigma_2]$.

Proof. Let us consider two cases on σ. If $\rho = \sigma_1 \to \sigma_2$ we are trivially done. If on the other hand $\rho = \alpha$ we have that $\mathit{instFun}(C_1, \alpha) = (E_2, \beta \to \gamma)$ where $E, [E_g] \Rightarrow \rho_g = \mathit{generalize}(C_1, \Gamma, \alpha)$, and $E_2 = \mathit{eqCheck}(E \bullet E_g \bullet (\beta_\star\ \bot) \bullet (\gamma_\star\ \bot), \rho_g, \beta \to \gamma)$. Assume that $\theta \models E_2$. By Lemma 6.47 we have $\theta \models E \bullet E_g \bullet (\beta_\star\ \bot) \bullet (\gamma_\star\ \bot)$. By Lemma 6.67 we have that there exists a θ_r such that $\vdash^{\mathsf{F}} \theta\theta_r(\alpha) \leq \theta\rho_g$, and $\theta\theta_r \models C_1$ (the use of θ_g in the theorem is inlined inside θ), and $\theta(\beta) \to \theta(\gamma) = \theta\rho_g$. It follows that $\boxed{\theta\theta_r(\alpha)} \preceq\sqsubseteq^{\rightarrow} \theta[\beta] \to \theta[\gamma]$ (since $\theta\rho_g$ must be a ρ-type). □

Lemma 6.69. *If* $\vdash C_1$ `wf`, $\Delta_{C_1} \vdash \rho_1$, $C_1 \vdash \Gamma$, *and* $\mathit{instMono}(C_1, \Gamma, \rho_1) = C_2, \rho$ *then for every* $\theta \models C_2$, *it is the case that* $\theta \models C_1$ *and* $\theta[\rho_1] \preceq\sqsubseteq \theta\rho$

Proof. Similar to the proof of Lemma 6.68 appealing to soundness for *mkMono*. □

Lemma 6.70 (Soundness)**.** *If* $\vdash C_1$ `wf`, $C_1 \vdash \Gamma$, *and* $\mathit{infer}(C_1, \Gamma, e) = C_2, \rho$ *then for all* $\theta \models C_2$, *it is the case that* $\theta \models C_1$ *and* $\theta\Gamma \vdash e : \theta[\rho]$.

Proof. We sketch the proof of soundness by induction on the number of steps of the algorithm. We ommit showing the well-formedness conditions in the inductive cases. We and consider the following cases.

- $\mathit{infer}(C_1, \Gamma, x) = C_2$ where $(x{:}\sigma) \in \Gamma$ and $\mathit{inst}(C_1, \sigma) = C_2$. In this case we may directly appeal to Lemma 6.63.
- $\mathit{infer}(C_1, \Gamma, e_1\ e_2) = C_2, \rho_2$ where

$$(E_1, \rho_1) = \mathit{infer}(C_1, \Gamma, e_1) \tag{19}$$
$$(E_2, \sigma_1 \to \sigma_2) = \mathit{instFun}(E_1, \Gamma, \rho_1) \tag{20}$$
$$(E_3, \rho_3) = \mathit{infer}(E_2, \Gamma, e_2) \tag{21}$$
$$E_4, [E_g] \Rightarrow \rho_g = \mathit{generalize}(\Gamma, E_3, \rho_3) \tag{22}$$
$$E_5 = \mathit{subsCheck}(E_4, [E_g] \Rightarrow \rho_g, \sigma_1) \tag{23}$$
$$C_2, \rho_2 = \mathit{inst}(E_5, \sigma_2) \tag{24}$$
$$\tag{25}$$

  Assume $\theta \models C_2$, it follows that $\vdash^{\mathsf{inst}} \theta[\sigma_2] \leq \theta[\rho_2]$ by Lemma 6.63. Moreover $\theta \models E_5$. By Lemma 6.50 we get that $\theta\sigma_1 \in [\![\theta([E_g] \Rightarrow \rho_g)]\!]$ and $\theta \models E_4$. It follows that there exists a θ_g such that $\theta\theta_g \models E_4 E_g$ and let $\theta\sigma_1 = \forall\overline{a}\,.\,\rho_1$. Assume without loss of generality that $\overline{a} \# \mathit{ftv}(\theta\Gamma)$. Moreover it must be that $\overline{a} \# \mathit{ftv}([\theta E_g] \Rightarrow \theta\rho_g)$ and $\rho_1 = \theta_g\theta\rho_g$ ($\overline{a}$ may appear in the range of θ_g). By Lemma 6.67 it must then be that there exists a θ_r such that $\vdash^{\mathsf{F}} \theta\theta_g\theta_r(\rho_3) \leq \theta_g\theta\rho_g = \rho_1$. But $\overline{a} \# \theta\Gamma$ and hence it must be that $\overline{\theta\Gamma}(\theta\theta_g\theta_r(\rho_3) \leq \forall\overline{a}\,.\,\rho_1$. Moreover $\theta\theta_g\theta_r \models E_3$ and hence by induction hypothesis $\theta\Gamma \vdash e_2 : \theta\theta_g\theta_r[\rho_3]$, and $\theta\theta_g\theta_r \models E_2$. Observe that the residual

$\theta_g\theta_r$ was only refering to fresh variables, and hence at this point we know that $\theta \models E_2$. By Lemma 6.68 we get that there exists a θ_r such that $\theta\theta_r \models E_1$ and $\theta\theta_r[\rho_1] \preceq\sqsubseteq^{\rightarrow} \theta[\sigma_1] \rightarrow \theta[\sigma_2]$. Moreover, by induction we get that $\theta\theta_r \models C_1$ and because θ_r is refering to fresh variables, $\theta \models C_1$. By induction we get $\theta\Gamma \vdash e_1 : \theta\theta_r[\rho_1]$. Applying rule SDAPP is enough to finish the case.

- $infer(C_1, \Gamma, (e :: \sigma)) = C_2, \rho_2$. This case involves the same reasoning about generalization as the application case and we ommit it.
- $infer(C, \Gamma, \texttt{let}\ x = u\ \texttt{in}\ e) = C_2, \rho$ where

$$E_1, \rho_1 = infer(C, \Gamma, u) \tag{26}$$
$$E_2, \rho_2 = instMono(E_1, \rho_1) \tag{27}$$
$$\rho_3 = zonkType(E_2, \rho_2) \tag{28}$$
$$\overline{\alpha} = \widehat{E_2}(\rho_3) - \widehat{E_2}(\Gamma) \quad \overline{\beta} = \overline{\alpha} \cap fcv(\rho_3) \quad \overline{a} \text{ fresh} \tag{29}$$
$$C_2, \rho = infer(E_2 - E_{2\overline{\alpha}}, \Gamma, (x{:}\forall \overline{a}\,.\,[\overline{\beta \mapsto a}]\rho_3), e) \tag{30}$$

Assume first of all that $\theta \models C_2$, and without loss of generality assume that $\overline{\alpha}\#dom(\theta)$ (otherwise we can work with a restriction of θ). Then, by induction hypothesis for (30) we get that $\theta \models E_2 - E_{2\overline{\alpha}}$. But consider the variable $\overline{\alpha}$. $E_{2\overline{\alpha}}$ contains only monomorphic equality bounds, and we can consider the most general unifier of $E_{2\overline{\alpha}}$, call it $E^{\dagger}_{2\overline{\alpha}}$. With a substitution of $\overline{\beta}$ to $\overline{a}$, we know that $[\overline{\beta \mapsto a}]E^{\dagger}_{2\overline{\alpha}}$, call it $\theta_{\overline{\alpha}}$, satisfies $E_{2\overline{\alpha}}$. That is, $\theta_{\overline{\alpha}} \models E_{2\overline{\alpha}}$. Then $\theta\theta_{\overline{\alpha}} \models E_2$ and by Lemma 6.69 there exists a θ_r such that $\theta\theta_{\overline{\alpha}}\theta_r[\rho_1] \preceq\sqsubseteq \theta\theta_{\overline{\alpha}}\rho_2$. and $\theta\theta_{\overline{\alpha}}\theta_r \models E_1$. Hence, by induction $\theta\Gamma \vdash u : \theta\theta_{\overline{\alpha}}\theta_r[\rho_1]$ and $\theta\theta_{\overline{\alpha}}\theta_r \models C_1$. But observe that the $\overline{\alpha}$ and the domain of θ_r were freshly created and hence $\theta \models C_1$. Hence induction hypothesis will finish the case if we can show that $\overline{a} = ftv([\overline{\beta \mapsto a}]\theta\theta_{\overline{\alpha}}\rho_2) - ftv(\theta\Gamma)$:

  - $\overline{a} \subseteq ftv([\overline{\beta \mapsto a}]\theta\theta_{\overline{\alpha}}\rho_2) - ftv(\theta\Gamma)$. This direction is trivial, for fresh enough $\overline{a}$.
  - $ftv([\overline{\beta \mapsto a}]\theta\theta_{\overline{\alpha}}\rho_2) - ftv(\theta\Gamma) \subseteq \overline{a}$. Assume an $a \in ftv([\overline{\beta \mapsto a}]\theta\theta_{\overline{\alpha}}\rho_2) - ftv(\theta\Gamma)$. The hard case is when either $a \in \rho_2$ or $\exists\gamma \in fcv(\rho_2)$ such that $a \in \theta\theta_{\overline{\alpha}}\gamma$. We will show that this case cannot happen. In the first case it must be that a belongs in $\theta\Gamma$. In the second it must be the case that $\gamma \notin \overline{\beta}$, and hence $\gamma \notin \overline{\alpha}$ or $\gamma \notin fcv(\rho_2)$. If $\gamma \notin \overline{\alpha}$ it means that it must be in $\widehat{E_2}(\Gamma)$ and hence $a \in \theta\Gamma$, a contradiction. The latter case, $\gamma \notin fcv(\rho_2)$ cannot happen as $\gamma \in fcv(\rho_2)$.

  Putting the above together, induction hypothesis for (30) and application of SDLET finishes the case.
- $infer(C_1, \Gamma, \lambda x\,.\,e) = C_2, \rho$. The case is similar to the case for generalizing `let`-bindings.

□

6.3.5 Main algorithm completeness

Lemma 6.71 (Scheme substitutivity)**.** *Assume that $\Delta \vdash [D] \Rightarrow \rho$ and $\overline{a}\#ftv([D] \Rightarrow \rho)$ and Δ contains only $\star$-flagged variables. If $\theta \models D$ then $[\overline{a \mapsto \sigma}]\theta \models D$ (where $\overline{\sigma}$ are constrained-variable-free).*

Proof. By induction on the metric of the constraint D. First of all, if $(\alpha_\star = \sigma_1) \in D$ we know that equality is preserved by substitutions, hence $\theta\alpha = \theta\sigma_1$ implies $[\overline{a \mapsto \sigma}]\theta\alpha = [\overline{a \mapsto \sigma}]\theta\sigma_1$, and moreoever if $(\alpha_\star \geq \bot)$ then the result follows easily. $(\alpha_\star \geq [E] \Rightarrow \rho)$ we have that $\theta\alpha \in [\![\theta E] \Rightarrow \theta\rho]\!]$ where without loss of generality we assume that $dom(E)\#dom(\theta)$. Assume that $\theta\alpha = \forall\overline{b}\,.\,\rho_a$. Then $\overline{b}\#ftv([\theta E] \Rightarrow \theta\rho)$, and without loss of generality assume also that $\overline{b}\#\overline{a}, ftv(\overline{\sigma})$. Then, there exists also a substitution θ_E such that $\theta_E \models \theta E$ and $\rho_a = \theta_E\theta\rho$. It follows that $\theta_E\theta \models E$ and consequently by induction $[\overline{a \mapsto \sigma}] \cdot (\theta_E\theta) \models E$ and $[\overline{a \mapsto \sigma}]\rho_a = [\overline{a \mapsto \sigma}](\theta_E\theta\rho)$. This implies that $([\overline{a \mapsto \sigma}] \cdot \theta_E)([\overline{a \mapsto \sigma}]\theta) \models E$, and moreover $([\overline{a \mapsto \sigma}] \cdot \theta_E)([\overline{a \mapsto \sigma}]\theta)(\rho) = [\overline{a \mapsto \sigma}]\rho_a$. Because $\overline{b}\#\overline{a}, ftv(\overline{\sigma})$ we have that $[\overline{a \mapsto \sigma}]\theta\alpha \in [\![[\overline{a \mapsto \sigma}]\theta E] \Rightarrow [\overline{a \mapsto \sigma}]\theta\rho]\!]$, as required. □

Lemma 6.72 (Scheme F-instance transitivity)**.** *If* $\vdash \varsigma$, $\sigma_1 \in [\![\varsigma]\!]$, *and* $\vdash^{\mathsf{F}} \sigma_1 \leq \sigma_2$ *then* $\sigma_2 \in [\![\varsigma]\!]$.

Proof. Let $\varsigma = [D] \Rightarrow \rho$. Since $\vdash \varsigma$ it must be that, if $\rho = \gamma$, then $\varsigma = [(\gamma_\star \perp)] \Rightarrow \gamma$. Additionally we know that for all $\gamma \in dom(D)$, $\Delta_D(\gamma) = \star$. Let $\sigma_1 = \forall \overline{a} \,.\, \rho_1$ and $\sigma_2 = \forall \overline{b} \,.\, [\overline{a \mapsto \sigma}]\rho_1$ such that $\overline{b} \# ftv(\forall \overline{a} \,.\, \rho_1)$. Then we know that there exists a θ_D such that $\theta_D \models D$ and $\rho_1 = \theta_D \rho$. Moreover $\overline{a} \# ftv([D] \Rightarrow \rho)$. We want to show that $\forall \overline{b} \,.\, [\overline{a \mapsto \sigma}]\rho_1 \in [\![[D] \Rightarrow \rho]\!]$. First of all, assume that ρ_1 is not a single quantified variable a because in this case it must be that $\varsigma = [(\gamma_\star \perp)] \Rightarrow \gamma$, and then trivially also $\sigma_2 \in [\![\varsigma]\!]$. So, assume that all quantified variables of σ_2 are the $\overline{b}$. We have to show several things:

- $\overline{b} \# ftv([D] \Rightarrow \rho)$. Assume on the contrary that there exists a $b \in ftv([D] \Rightarrow \rho)$. It follows that $b \in \theta_D \rho = \rho_1$. But since $\overline{a} \# ftv([D] \Rightarrow \rho)$ it follows that $b \in ftv(\forall \overline{a} \,.\, \rho_1)$, a contradiction.
- We want to find a substitution θ°_D such that $[\overline{a \mapsto \sigma}]\rho_1 = \theta^\circ_D \rho$. We know that $\rho_1 = \theta_D \rho$, so consider as a candidate θ°_D the substitution $[\overline{a \mapsto \sigma}]\theta_D$ (where the notation means that we substitute the $\overline{a}$ for $\overline{\sigma}$ in the range of θ_D. We need to show that since $\theta_D \models D$ then $[\overline{a \mapsto \sigma}]\theta_D \models D$ as well. But this follows by Lemma 6.71.

□

Lemma 6.73. *If* $(\forall \overline{b} \,.\, \rho) \in [\![\varsigma]\!]$ *and* $\overline{a} \# ftv(\varsigma), \overline{b}$ *then* $\forall \overline{ab} \,.\, \rho) \in [\![\varsigma]\!]$.

Proof. Straightforward unfolding of the definitions. □

Lemma 6.74 (Normalization completeness)**.** *Assume that* $C \vdash E$ `wf`, $\Delta_C \Delta_E \vdash \rho$, $\theta \models C \bullet E$, *the domain of* E *is* $\star$*-bound variables only, and the constraints* E *and type* ρ *contain no rigid type variables, and* $normalize(E, \rho) = [E_g] \Rightarrow \rho_g$, *then* $\theta\rho \in [\![[E_g] \Rightarrow \rho_g]\!]$.

Proof. By induction on the number of recursive calls to $normalize(E, \rho)$. The only interesting case is the case for N1:

- We have that $normalize(E, \alpha) = [E \bullet D - \alpha] \Rightarrow \rho$, when $(\alpha_\star \geq [D] \Rightarrow \rho) \in E$. since $\theta \models C \bullet E$ it is the case that $\theta\alpha \in [\![[\theta D] \Rightarrow \theta\rho]\!]$ (where we assume that the domain of D is disjoint from the domain of θ). We need to show that $\theta\alpha \in [\![\theta[E^\circ \bullet (D - \alpha)^\circ] \Rightarrow \theta\rho^\circ]\!]$ where $\circ$ is a renaming of the domain of E and D such that they are disjoint from the domain of θ. Let $\theta\alpha = \forall \overline{a} \,.\, \rho_a$ such that $\overline{a} \# ftv([\theta D] \Rightarrow \theta\rho)$. then there exists a θ_D such that $\theta_D \theta\rho = \rho_a$ and $\theta_D \models \theta D$. However, it must also be that $\theta^\circ_D \theta^\circ \models \theta[E^\circ \bullet (D - \alpha)^\circ]$ and then $\theta_D \theta^\circ \theta\rho^\circ = \theta_D \theta\rho = \rho_a$. To finish the case we need to show that $\overline{a} \# ftv(\theta[E^\circ \bullet (D - \alpha)^\circ] \Rightarrow \theta\rho^\circ)$. But assume on the contrary that there exists such an a. But since $\overline{a} \in ftv(\rho_a)$ and they cannot be in $\theta\rho$ it must be that $\overline{a} \in range(\theta_D)$. If there exists such an a it must be that $a \in range(\theta_{ftv(\rho)})$ which is impossible because $\overline{a} \# ftv([\theta D] \Rightarrow \theta\rho)$.

□

Lemma 6.75 (Generalization completeness)**.** *Assume that* $\vdash C_1$ `wf`, $C_1 \vdash \Gamma, \rho$, *for all* $\theta \models C_1$ *and* $E, \varsigma = generalize(C_1, \Gamma, \rho)$, C_1 *and* ρ *do not contain any free rigid variables, and* $\overline{a} = ftv(\theta\rho) - ftv(\theta\Gamma)$. *It follows that* $(\forall \overline{a} \,.\, \theta\rho) \in [\![\theta\varsigma]\!]$ *and* $\theta \models E$.

Proof. We have that $\overline{\beta} = \widehat{C_1}(\rho) - \widehat{C_1}(\Gamma)$, $normalize(C_{\overline{\beta}}, \rho) = [E_g] \Rightarrow \rho_g$ and $E = C - C_{\overline{\beta}}$, $\varsigma = [E_g] \Rightarrow \rho_g$. We know that $\theta \models C_1$ and assume a renaming of the variables $\overline{\beta}$ to $\overline{\beta}^\circ$. Then we will first show that $\overline{a} \# ftv([\theta E^\circ_g] \Rightarrow \theta\rho^\circ_g)$. Assume, by contradiction, that there exists an $a \in \overline{a}$ such that $a \in ftv([\theta E^\circ_g] \Rightarrow \theta\rho^\circ_g)$. It follows that there exists a $\beta \in \widehat{C_1}(\Gamma)$ such that $\theta(\beta) = a$. But then it must be that $a \notin \overline{a}$. Or, it must be that $a \in ftv([E_g] \Rightarrow \rho_g)$. But this cannot happenm as E_g and ρ_g can't contain any rigid variables. By Lemma 6.73, it is hence enough to show that $\theta\rho \in [\![\theta\varsigma]\!]$. But this follows from Lemma 6.74. □

Lemma 6.76 (Inst-fun completeness)**.** *If* $\vdash C_1$ `wf` *and* $C_1 \vdash \Gamma$ *and* $\theta_1 \models C_1$*,* $dom(\theta_1) = dom(C_1)$*, and* $\theta_1[\rho] \preceq\sqsubseteq^{\rightarrow} \sigma_1' \rightarrow \sigma_2'$ *then* $instFun(C_1, \Gamma, \rho) = C_2, \sigma_1 \rightarrow \sigma_2$ *such that there exists a* θ_2 *with* $\theta_1\theta_2 \models C_2$ *and* $\sigma_1' = \theta_1\theta_2[\sigma_1]$*,* $\sigma_2' = \theta_1\theta_2[\sigma_2]$*.*

Proof. Unfolding definitions and appealing to completeness of *eqCheck* and $normalize(\cdot)$. □

We use below the fact that whenever we infer a type, that type contains no rigid type variable, but only unification ones. In essence, rigid type variables can only start appearing by open type annotations, (and the proofs need be slightly modified), but for simplicity we only treat closed type annotations.

Lemma 6.77 (Main completeness)**.** *If* $\vdash C_1$ `wf`*,* $C_1 \vdash \Gamma$*,* $\theta_1 \models C_1$*, and* $\theta_1\Gamma \vdash e : \rho'$ *then* $infer(C_1, \Gamma, e) = C_2, \rho$ *and there exists a* θ_2 *such that* $\theta_1\theta_2 \models C_2$*, and* $\theta[\rho] \preceq\sqsubseteq \rho'$*.*

Proof. By induction on the size of term e, and considering cases on the derivation $\theta_1\Gamma \vdash e : \rho'$. The cases for `let` nodes rely on Lemma 5.7 and Corollary 6.7, and is similar to the ordinary case for higher-rank types (since the types are box-free), and the substitutions monomorphic. The case for abstraction relies on Lemma 5.7 and is similar. The cases for annotations and applications are more interesting as they use the power of constrained types. We show below the case for applications, and the case for annotations is quite similar.

- Case SDAPP. In this case we have that $\vdash C_1$ `wf`, $C_1 \vdash \Gamma$, $\theta \models C_1$ and $\theta_1\Gamma \vdash e_1\ e_2 : \rho_2'$ given that

$$\theta_1\Gamma \vdash^{\mathsf{sd}} e_1 : \rho' \tag{31}$$
$$\rho'(\preceq\sqsubseteq^{\rightarrow})\sigma_1' \rightarrow \sigma_2' \tag{32}$$
$$\Gamma \vdash^{\mathsf{sd}} e_2 : \rho_3' \quad \overline{a} = ftv(\rho_3') - ftv(\Gamma) \tag{33}$$
$$\vdash^{\mathsf{F}} \lfloor \forall \overline{a}\,.\,\rho_3' \rfloor \leq \lfloor \sigma_1' \rfloor \quad \vdash^{\mathsf{inst}} \sigma_2' \leq \rho_2' \tag{34}$$

  By induction hypothesis for (31) we get that $infer(C_1, \Gamma, e_1) = E_1, \rho_1$ such that there exists a θ_2 with $\theta_1\theta_2 \models C_2$ and $\theta_1\theta_2[\rho_1] \preceq\sqsubseteq \rho'$. By Lemma 6.76 and transitivity of $\preceq\sqsubseteq$ we get that $instFun(E_1, \rho_1) = E_2, \sigma_1 \rightarrow \sigma_2$ such that there exists a θ_3 with $\theta_1\theta_2\theta_3 \models E_2$ and $\theta_1\theta_2\theta_3[\sigma_1] = \sigma_1'$ and $\theta_1\theta_2\theta_3[\sigma_2] = \sigma_2'$. Moreover, we know at this point that $\theta_1\theta_2\theta_3\Gamma \vdash e_2 : \rho_3'$ (since all the variables of Γ are already in the domain of θ_1). It follows by induction hypothesis that $infer(E_2, \Gamma, e_2) = E_3, \rho_3$ such that there exists a θ_4 with $\theta_1\theta_2\theta_3\theta_4 \models E_3$ and $\theta_1\theta_2\theta_3\theta_4[\rho_3] \preceq\sqsubseteq \rho_3'$. Next, we have that if $\overline{a} = ftv(\theta_1\theta_2\theta_3\theta_4\rho_3) - ftv(\theta_1\Gamma)$ it is the case that $\vdash^{\mathsf{F}} \forall \overline{a}\,.\,\theta_1\theta_2\theta_3\theta_4\rho_3 \leq \theta_1\theta_2\theta_3\sigma_1 = \theta_1\theta_2\theta_3\theta_4\sigma_1$. By Lemma 6.75 we may call $generalize(\Gamma, E_3, \rho_3)$ to get back E_4, ς_3 ($generalize$ never fails). By Lemma 6.75 we get that $\forall \overline{a}\,.\,\theta_1\theta_2\theta_3\theta_4\rho_3 \in [\![\theta_1\theta_2\theta_3\theta_4\varsigma_3]\!]$, and $\theta_1\theta_2\theta_3\theta_4 \models E_4$, and by Lemma 6.72 it must be that $\theta_1\theta_2\theta_3\theta_4\sigma_1 \in [\![\theta_1\theta_2\theta_3\theta_4\varsigma_3]\!]$. By Lemma 6.55 it must be the case that $E_5 = subsCheck(E_4, \varsigma_3, \sigma_1)$ such that there exists a θ_5 with $\theta_1\theta_2\theta_3\theta_4\theta_5 \models E_5$. Finally, the instantiation step concludes the case by appplying the algorithm clause IAPP.

□

7 Discussion

We present in this section several extensions to FPH, and discuss alternative design choices.

7.1 Bidirectionality

Bidirectional propagation of type annotations may further reduce the amount of required type annotations in FPH. It is relatively straightforward to add bidirectional annotation propagation to the specification of FPH (see e.g. (Peyton Jones et al. 2007)). This bidirectional annotation procedure can be implemented as a separate preprocessing pass, provided that we support open type annotations, and annotated λ-abstractions. Alternatively, this procedure can be implemented by weaving an inference judgement of the form $\Gamma \vdash^{\mathsf{sd}} e :^{\Uparrow} \rho'$ and a checking judgement $\Gamma \vdash^{\mathsf{sd}} e :^{\Downarrow} \rho'$. Necessarily, this bidirectional system is syntax-directed. A special-top level judgement $\Gamma \vdash^{\mathsf{sd}}_{\star} e :^{\Downarrow} \sigma'$ checks an expression *against a polymorphic type* as follows:

$$\frac{\Gamma \vdash^{\mathsf{sd}} e :^{\Downarrow} \rho' \quad \overline{a}\#\Gamma}{\Gamma \vdash^{\mathsf{sd}}_{\star} e : \forall \overline{a}\,.\,\rho'}\ \textsc{skol} \qquad \frac{\Gamma \vdash^{\mathsf{sd}} e :^{\Uparrow} \rho' \quad \overline{a}\#\Gamma \quad \vdash^{\mathsf{F}} \lfloor \forall \overline{a}\,.\,\rho' \rfloor \leq \sigma}{\Gamma \vdash^{\mathsf{sd}}_{\star} e :^{\Downarrow} \boxed{\sigma}}\ \textsc{cbox}$$

Rule SKOL simply removes the top-level quantifiers and checks the expression against the body of the type. Rule CBOX checks an expression against a single box. In this case, we must simply infer a type for the expression, as we cannot use its contents as a type annotation. Consequently, in our main checking judgement $\Gamma \vdash^{\mathsf{sd}} e :^{\Downarrow} \rho'$, we can always assume that ρ' is never a single box. The rule for checking applications then becomes:

$$\frac{\begin{array}{c}\Gamma \vdash^{\mathsf{sd}} e_1 :^{\Uparrow} \sigma' \quad \sigma' \preceq \sqsubseteq^{\rightarrow} \sigma'_1 \rightarrow \sigma'_2 \\ \Gamma \vdash^{\mathsf{sd}}_{\star} e_2 :^{\Downarrow} \sigma'_1 \quad \vdash^{\mathsf{inst}} \sigma'_2 \leq \rho'_2 \quad \lfloor \rho'_2 \rfloor = \lfloor \rho' \rfloor\end{array}}{\Gamma \vdash^{\mathsf{sd}} e_1\ e_2 :^{\Downarrow} \rho'}\ \textsc{app-check}$$

Following previous work, we first infer a type for the function e_1. Notice that e_2 is *checked* against the type σ'_1, which eliminates the need to generalize its type, as happens in the inference-only syntax-directed specification. Since ρ' cannot be a single box, we may simply instantiate σ'_2 to ρ'_2 and check that this type is equal modulo the boxes to the ρ' type that the context requires.

Annotations, no longer reveal polymorphism locally, but rather propagate the annotation down the term structure. The rule ANN-INF below infers a type for an annotated expression $e\texttt{::}\sigma$ by first *checking* e against the annotation σ:

$$\frac{\Gamma \vdash^{\mathsf{sd}}_{\star} e :^{\Downarrow} \sigma \quad \vdash^{\mathsf{inst}} \sigma \leq \rho'}{\Gamma \vdash^{\mathsf{sd}} (e\texttt{::}\sigma) :^{\Uparrow} \rho'}\ \textsc{ann-inf}$$

The rule for inferring types for λ-abstractions is similar to rule SDABS, but the rule for *checking* λ-abstractions allows us now to check a function against a type of the form $\sigma'_1 \rightarrow \sigma'_2$:

$$\frac{\sigma'_1 \sqsubseteq \sigma_1 \quad \Gamma, (x{:}\sigma_1) \vdash^{\mathsf{sd}}_{\star} e :^{\Downarrow} \sigma'_2}{\Gamma \vdash^{\mathsf{sd}} \lambda x\,.\,e :^{\Downarrow} \sigma'_1 \rightarrow \sigma'_2}\ \textsc{abs-check}$$

Notice that σ'_1 must be made box-free before entering the environment, to preserve our invariant that environments are box-free.

With these additions, and assuming that support for open type annotations, we can type functions with more elaborate types than simply $\tau \rightarrow \rho$ types, as the FPH original system does. Recall, for instance, Example 3.7 from Section 3.4.

```
f   :: forall a. a -> [a] -> Int
foo :: [Int -> forall b.b->b]

bog = f (\x y ->y) foo
```

Though `bog` is untypeable (even in a bidirectional system), we can recover it with the (ordinary) annotation:

```
bog = f (\x y -> y :: Int -> forall b. b -> b) foo
```

Special forms for annotated λ-abstractions (Section 3.4) are not necessary in a bidirectional system. Indeed our implementation is a bidirectional version of our basic syntax-directed type system.

7.2 η-conversion and deep instance relations

Unsurprisingly, since FPH is based on System F, it is not stable under η-conversion. In particular, if $f : \sigma \to$ `Int` is in the evnironment, it is not necessarily the case that $\lambda x.f\ x$ is typeable, since x must have a τ-type. Conversely, if an expression $\lambda x.e\ x$ makes a context $\mathcal{C}[(\lambda x.e\ x)]$ typeable, then it is not necessarily the case that $\mathcal{C}[e]$ is typeable, for a more subtle reason. Consider the code below:

```
f   :: Int -> forall a. a -> a
g   :: forall a. a -> [a] -> a
lst :: [forall a. Int -> a -> a]

g1 = g (\x -> f x) lst   -- OK
g2 = g f lst             -- fail!
```

The application in `g2` (untypeable in implicitly typed System F) fails since `lst` requires the instantiation of `g` with type $\forall a.$`Int` $\to a \to a$, whereas `f` has type `Int` $\to \forall a.a \to a$. The FPH system, which is based on System F, is not powerful enough to understand that these two types are isomorphic. Although such conversions are straightforward in predicative systems (Peyton Jones et al. 2007), the presence of impredicativity complicates our ability to support them. In fact, no system with impredicative instantiations proposed to date fully supports η-conversions.

We are currently seeking ways to extend our instance relation to some "deep" version that treats quantifiers to the right of arrows as if they were top-level, but combining that with impredicative instantiations remains a subject of future work.

7.3 Alternative design choices

Our design choices are a compromise between simplicity and expressiveness. In this section we sketch several alternatives.

Removing the $\preceq\!\sqsubseteq$ relation Our use of $\preceq\!\sqsubseteq$ in the type system of Figure 2 is motivated by examples where we need to extract and use some polymorphic value out of a data structure, as in `head ids 3`. If we are willing to sacrifice this convenience, and instead require annotations the relation $\preceq\!\sqsubseteq$ need not be present in our specification, and rule SUBS is unnecessary. The implementation becomes simpler, too. Nevertheless, we believe the extra complexity is worth it because it saves many annotations, and perhaps more importantly, it allows us to type all terms of System F that consist only of applications and variables (Section 3.5).

Typing abstractions with more expressive types Recall that λ-abstractions are typed with box-free types only. This implies that certain transformations, such as *thunking*, may break typeability. For example, consider the following code:

```
f1 :: forall a. (a -> a) -> [a] -> Int
g1 = f (choose id) ids   -- OK

f2 :: forall a b. (b -> a -> a) -> [a] -> Int
g2 = f (\ _ -> choose id) ids -- fails!
```

In the example, while `g1` type checks, simply thunking the application `choose id` breaks typeability, because the type $\boxed{\forall a\,.\,a \to a} \to \boxed{\forall a\,.\,a \to a}$ cannot be unboxed.

An obvious alternative would be to allow arbitrary ρ' types as results of λ-abstractions, and lift our invariant that environments are box-free to allow τ' types as the arguments of abstractions. Though such a modification allows for even fewer type annotations (the bodies of abstractions could use impredicative instantiations and no annotations would be necessary), we do not know of a sound and complete algorithm that could implement such an extension, for at least two reasons.

1. One complication has to do with boxy instantiation $\preceq$. If an argument x enters the environment with type $\boxed{\sigma}$, then in the body of the abstraction boxy instantatiation may allow x to be used at two different types—something that algorithmically cannot be implemented. Perhaps an alternative would be to introduce a different type of *rigid boxes* that do not allow $\preceq$.
2. A second complication has to do with uses of boxy instantiation for the body types of abstractions. If a λ-abstraction is typed with $\tau \to \boxed{\forall a\,.\,\rho}$ then by using $\preceq$ when typing the body of the abstraction it can also be typed with $\tau \to \boxed{\rho}$. Algorithmically, this means that probably we would have to "'flexify" all bounds to the right of arrows. If for a function we have inferred type $\texttt{Int} \to \alpha$ where $\alpha = \forall a\,.\,\rho$, the algorithm would probably have to modify this type to $\texttt{Int} \to \alpha$ where $\alpha \geq [\alpha_\star\ \bot] \Rightarrow \rho$, in order to capture the possibility of $\preceq$ used in the typing derivation of the body of the abstraction.

In general we do not believe that the programming benefits justify the implementation and specification costs—after all in many cases λ-abstractions are `let`-bound, and hence they are forced by the rules for `let`-bound expressions to be box free anyway!

A box-free specification A safe approximation of where type annotations are necessary is at `let`-bindings or λ-abstractions that have to use rich types. Perhaps surprisingly, taking this guideline one step further, if we *always* require annotations in bindings with rich types then we no longer need boxes in the specification *at all*! Consider the basic type system of Figure 2 with the following modifications:

1. Drop all boxy structure from all typing rules, that is, replace all ρ', σ', types with ρ and σ types, and completely remove SUBS and $\preceq\sqsubseteq$. Instantiate with arbitrary σ types in rule INST.
2. Replace rule LET and ABS with their corresponding versions for Damas-Milner types

$$\frac{\Gamma \vdash u : \forall \overline{a}\,.\,\tau \qquad \Gamma, (x{:}\forall \overline{a}\,.\,\tau) \vdash e : \rho}{\Gamma \vdash \texttt{let}\ x = u\ \texttt{in}\ e : \rho}\ \text{LET} \qquad \frac{\Gamma, (x{:}\tau_1) \vdash e : \tau_2}{\Gamma \vdash \lambda x\,.\,e : \tau_1 \to \tau_2}\ \text{ABS}$$

3. Add provision for annotated `let`-bindings and λ-abstractions:

$$\frac{\Gamma \vdash u : \sigma \quad \Gamma, (x{:}\sigma) \vdash e : \rho}{\Gamma \vdash \texttt{let}\ x\texttt{::}\sigma = u\ \texttt{in}\ e : \rho}\ \text{LET-ANN} \qquad \frac{\Gamma, (x{:}\sigma_1) \vdash e : \sigma_2}{\Gamma \vdash (\lambda x\,.\,e :: \sigma_1 \to \sigma_2) : \sigma_1 \to \sigma_2}\ \text{ABS-ANN}$$

The resulting type system enjoys sound and complete type inference, by using essentially the same algorithm as the FPH type system. However, this variation is more demanding in type annotations than the box-based FPH. For instance, one has to annotate *every* `let`-binding that uses rich types, even if its type did not involve any impredicative instantiations. For example:

```
f :: Int -> (forall a. a -> a) -> (forall a. a -> a)
h = f 42      -- fails!
```

The binding for `h` has a rich type an hence must be annotated, although no impredicative instantiation took place.

Given the fact that this simplification is more demanding in type annotations, we believe that it is less suitable for a real-world implementation.

	Specification	**Implementation**	**Placement of annotations / typeable programs**
HM^F	Simple, "minimality" restrictions	Simple	Annotations may be needed on λ-abstractions with rich types and on arguments that must be kept polymorphic
ML^F	Heavyweight, declarative	Heavyweight	Precise, annotations only required for usage of argument variables at two or more types
Boxy Types	Complex, syntax-directed, dark corners	Simple	No clear guidelines, not clear what fragment of System F is typed without annotations
FPH	Simple, declarative	Heavyweight	Precise, annotations on `let`-bindings and λ-abstractions with rich types, types all applicative System F terms and more without annotations

Figure 12: Quick summary of most relevant related works

8 Related work

There are several recent proposals for annotation-driven type inference for first-class polymorphism. The most relevant works are the ML^F language of Rémy and Le Botlan (Le Botlan and Rémy 2003), the HM^F language of Leijen (Leijen 2007a), and the Boxy Types system, proposed by the authors (Vytiniotis et al. 2006). These works differ in simplicity of specification, implementation, placement of type annotations, and expressiveness. We present an extensive comparison below and a quick summary in Table 12.

ML^F and Rigid ML^F The ML^F language of Le Botlan and Rémy (Le Botlan and Rémy 2003; Le Botlan 2004) partly inspired this work. The biggest difference between this language and other approaches is that it extends System F types with constraints of the form $\forall(Q)\tau$ so to enable principal types for all expressions. Therefore, `let`-expansion preserves typeability in ML^F, unlike systems that use only System F types. Because the type language is more expressive, ML^F requires strictly fewer annotations. In ML^F, annotations are necessary only when some variable is *used* at two or more polymorphic types—in contrast, in our language, variables must be annotated when they are *defined* with rich types. For example, the following program

```
f = \x -> x ids
```

needs no annotation in ML^F because x is only used once. FPH requires an annotation on `x`. Hence we are more restrictive.

A big drawback of ML^F is the complexity of its specification: constrained types appear in the declarative type system and the instance relation of ML^F must include them. Our specification is considerably simpler because we do not need a constraint-based instance relation. Furthermore, our low-level implementation is a variation of the ML^F implementation. However, because we do not expose a constraint-based instance relation in the specification, we can formalize our algorithm as directly manipulating sets of System F types. In contrast, ML^F internalizes the subset relation between sets of System F types as a syntactic instance relation, and formalizes type inference with respect to this somewhat complex syntactic instance relation. Le Botlan and Rémy study the set-based interpretation of ML^F in a recent report (Le Botlan and Rémy 2007), which inspired our set-theoretic interpretation of schemes.

Despite the simplifications that FPH provides, there are technical parallels between our work and ML^F. One of the key ideas behind ML^F is that all polymorphic instantiations are "hidden" behind

constrained type variables. Our type system uses anonymous boxes for the same purpose.Furthermore, the *revelation* of implicit polymorphism is achieved in ML^F at type annotation nodes, where explicit type information is present. Similarly, the revelation of polymorphism is achieved in FPH when a boxed type meets an annotation.

Finally, ML^F is a source language and is translated to an explicitly typed intermediate language, such as explicitly typed System F, using coercion terms (Leijen and Löh 2005). Coming up with a typed intermediate language for ML^F that is suitable for a compiler and does not require term-level coercions is still a subject of research. In contrast, because FPH is based on System F, elaborating FPH to System F is straightforward.

A variation of ML^F similar in expressive power to FPH is Leijen's Rigid ML^F (Leijen 2007b). Like FPH, Rigid ML^F does not include constrained types. Instead, it resolves constraints by instantiating flexible bounds at `let`-nodes. However Rigid ML^F is specified using the ML^F instance relation. Consequently, despite the fact that types in the environment are System F types, to reason about typeability one must reason using the ML^F machinery. Additionally, the rules of Rigid ML^F require that when instantiating the types of `let`-bound expressions, the constraint that is used in the typing derivation of the `let`-bound expression is the most general. Requiring programmers to think in terms of most general ML^F constraints may even be more complicated than requiring them to reason with ML^F constraints, as in the original ML^F proposal.

Boxy Types Boxy Types (Vytiniotis et al. 2006) is an earlier proposal by the authors to address type inference for first-class polymorphism. Like this paper, Boxy Types uses boxed System F types to hide polymorphism. Because boxes provide an elegant way to mark impredicativity, we have reused that syntax in this work.

However, boxes play a different role in our previous work. In Boxy Types, boxes merely distinguish the parts of types that were inferred from those that result from some type annotation, combining bidirectional annotation propagation with type inference. In a Boxy Types judgement of the form $\Gamma \vdash e : \rho'$, the ρ' type should be viewed as input to the type-checker, which asks for the boxes of ρ' to get filled in. In this work, the ρ' type is an output, and boxes simply mark where impredicative instantiations took place.

Boxy Types were implemented using a relatively simple algorithm which modestly extends Hindley-Damas-Milner unification with local annotation propagation. Because the algorithm does not manipulate instance constraints, it cannot delay instantiations. Therefore, the type system must make local decisions. In particular, Boxy Types often requires programs to unbox the contents of the boxes too early. For type inference completeness, if information about the contents of a box is not locally available, it must contain a monomorphic type. As a result, the basic Boxy Types system requires many type annotations. Ad-hoc heuristics, such as N-ary applications, and elaborate type subsumption procedures, relieve the annotation burden but further complicate the specification and the predictability of the system.

Although some programs are typeable with Boxy Types and are not typeable (without annotation) in FPH, and vice versa, we believe that the simpler specification of FPH is a dramatic improvement.

HM^F Leijen's HM^F system (Leijen 2007a) is yet another interesting point in the design space. The HM^F system enjoys a particularly simple inference algorithm (a variant of Algorithm W), and one that is certainly simpler than FPH. In exchange, the typing rules are somewhat unconventional in form, and it is somewhat harder to predict exactly where a type annotation is required and where none is needed.

The key feature of HM^F is a clever application rule, where impredicative instantiations are determined by a local match procedure. In the type system, this approach imposes certain "minimality" conditions that require (i) that all types entering the environment are the most general types that can be assigned to programs, and (ii) that all allowed impredicative instantiations of functions are those that "minimize" the polymorphism of the returned application.

The local match procedure means that HM^F takes eager decisions: in general, polymorphic functions get instantiated by default, unless specified otherwise by the programmer. For example, the program `single id` (where `single` has type $\forall a\,.\,a \to [a]$) cannot be typed with type $[\forall a\,.\,a \to a]$. The top-level quantifiers of `id` are instantiated too early, before the local match procedure. Because FPH delays instantiations using constraints, we may type this expression with $[\boxed{\forall a\,.\,a \to a}]$ (but we would still need an annotation to `let`-bind it). In HM^F one may annotate the function `single`, or specify with a *rigid type annotation* that the type of `id` must not be instantiated: `(single (id :: forall a. a -> a))`.[5] Note that HM^F annotations are different than the annotations found, for instance, in Haskell—e.g. `(id :: forall a. a -> a) 42` is rejected.

Leijen observes that local match procedures are, in general, not robust to program transformations. If only a local match were to be used, the application `(cons id) ids` would not typeable, while `(revcons ids) id` would be (where `revcons` has type $\forall a.[a] \to a \to [a]$). Hence, these problems are circumvented in HM^F by using an N-ary application typing rule that uses type information from *all arguments* in an application.

In general, annotations are needed in HM^F on λ-abstractions with rich types and on arguments that must be kept polymorphic. For example, if $f : \forall a\,.\,a \to \ldots$ and $arg : \forall b\,.\,\tau$, an annotation will be needed, $f\ (arg :: \forall b\,.\,\tau)$, to instantiate a with $\forall b\,.\,\tau$. However in some cases, annotation propagation and N-ary applications may make such annotations redundant.

Because HM^F requires most general types in derivations, there are programs typeable in HM^F but not in FPH. For example, `let g = append ids in ...` requires an annotation in FPH, whereas it seamlessly typechecks in HM^F. On the other hand, flexible instantiation allows FPH to type examples such as

```
f :: forall a. [a] -> [a] -> a
g = f (single id) ids
```

where HM^F (even with annotation propagation) fails. Overall, we believe that the placement of required annotations in FPH is somewhat easier to describe than in HM^F. But on the other hand, HM^F posseses a significantly simpler implementation and metatheory.

Other works For completeness, we outline some more distantly connected works. Full type reconstruction for (implicitly typed) System F is undecidable (Wells 1999). Kfoury and Wells stratify System F types by rank (polymorhism on the left of function types), and show undecidability of type reconstruction for System F with types of rank-3 or higher. On the other hand, the rank-2 fragment of System F is decidable (Kfoury and Wells 1994).

Pfenning shows that even *partial type inference* (Pfenning 1988) where only type abstractions and the positions of type applications are known, but not the types of λ-abstraction arguments, for the n-th order polymorphic λ-calculus is equivalent to n-th order unification, which is undecidable. Recent work (Le Botlan and Rémy 2007) shows that one only needs polymorphic function argument annotations (and not type abstractions and type applications) to embed all of System F. Notice that Pfenning's work does not include `let`-bound definitions, which as we have seen, are another cause for lack of principal types in System F.

A different line of work explores type inference for predicative higher-rank polymorphism (Odersky and Läufer 1996; Peyton Jones et al. 2007; Rémy 2005). Odersky and Läufer made the observation that once all polymorphic function arguments are annotated, type inference for predicative higer-rank polymorphism becomes decidable, even in the presence of `let`-bound expressions. Peyton Jones *et al.* explore variations of the Odersky-Läufer type system that support a bidirectional propagation of type annotations. Finally Rémy proposed a clean separation of the bidirectional propagation of type annotations, through a phase called *shape inference*, that is performed before type inference.

[5] A final possibility would be for the annotation $\forall a\,.\,a \to a$ to have been somehow *propagated* to `id`.

9 Future work and conclusions

We have presented a simple, expressive declarative specification for type inference for impredicative polymorphism. We have implemented the system in prototype form; next, we plan to retro-fit the implementation to a full-scale compiler. Our reference implementation is not focused around efficiency, although the unification algorithm is of polynomial time complexity. As future work, we would like to evaluate the efficiency of unification in large programs, and study the interaction with other forms of constraints, such as qualified types. Preliminary work by Leijen and Löh (2005) shows how to combine ML^F-style unification with qualified types, and we expect no significant difficulties to arise.

References

Luis Damas and Robin Milner. Principal type-schemes for functional programs. In *Conference Record of the 9th Annual ACM Symposium on Principles of Programming Languages*, pages 207–12, New York, 1982. ACM Press.

Jacques Garrigue and Didier Rémy. Semi-explicit first-class polymorphism for ML. *Journal of Information and Computation*, 155:134–169, 1999.

AJ Kfoury and JB Wells. A direct algorithm for type inference in the rank-2 fragment of the second-order lambda calculus. In *ACM Symposium on Lisp and Functional Programming*, pages 196–207. ACM, Orlando, Florida, June 1994.

D Le Botlan and D Rémy. MLF: raising ML to the power of System F. In *ACM SIGPLAN International Conference on Functional Programming (ICFP'03)*, pages 27–38, Uppsala, Sweden, September 2003. ACM.

Didier Le Botlan. *MLF : Une extension de ML avec polymorphisme de second ordre et instanciation implicite*. PhD thesis, Ecole Polytechnique, May 2004. 326 pages, also available in english.

Didier Le Botlan and Didier Rémy. Recasting MLF. Research Report 6228, INRIA, Rocquencourt, BP 105, 78 153 Le Chesnay Cedex, France, June 2007.

Daan Leijen. HMF: simple type inference for first-class polymorphism. URL `http://research.microsoft.com/users/daan/download/papers/hmfdraft.pdf`. September 2007a.

Daan Leijen. A type directed translation of MLF to System-F. In *ACM SIGPLAN International Conference on Functional Programming (ICFP'07)*, Freiburg, Germany, 2007b. ACM.

Daan Leijen and Andres Löh. Qualified types for MLF. In *ACM SIGPLAN International Conference on Functional Programming (ICFP'06)*, pages 144–155. ACM Press, 2005.

R Milner. A theory of type polymorphism in programming. *JCSS*, 13(3), December 1978.

M Odersky and K Läufer. Putting type annotations to work. In *23rd ACM Symposium on Principles of Programming Languages (POPL'96)*, pages 54–67. ACM, St Petersburg Beach, Florida, January 1996.

Simon Peyton Jones, Dimitrios Vytiniotis, Stephanie Weirich, and Mark Shields. Practical type inference for arbitrary-rank types. *J. Funct. Program.*, 17(1):1–82, 2007. ISSN 0956-7968. doi: http://dx.doi.org/10.1017/S0956796806006034. URL `http://research.microsoft.com/~simonpj/papers/higher-rank/`.

Frank Pfenning. Partial polymorphic type inference and higher-order unification. In *LFP '88: Proceedings of the 1988 ACM conference on LISP and functional programming*, pages 153–163, New York, NY, USA, 1988. ACM. ISBN 0-89791-273-X. doi: http://doi.acm.org/10.1145/62678.62697.

Didier Rémy. Simple, partial type inference for System F, based on type containment. In *ACM SIGPLAN International Conference on Functional Programming (ICFP'05)*, pages 130–143, Tallinn, Estonia, September 2005. ACM.

Dimitrios Vytiniotis, Stephanie Weirich, and Simon Peyton Jones. Boxy types: Inference for higher-rank types and impredicativity. In *ACM SIGPLAN International Conference on Functional Programming (ICFP'06)*, Portland, Oregon, 2006. ACM Press.

JB Wells. Typability and type checking in system F are equivalent and undecidable. *Ann. Pure Appl. Logic*, 98:111–156, 1999.