Sanitized Type Inference in Context*

Ningning Xie** and Bruno C. d. S. Oliveira

The University of Hong Kong

Abstract. Gundry et al. proposed type inference in context as a general foundation for unification/type inference algorithms. The key idea is based on the notion of information increase. Following this work, a more syntactic foundation for information increase is presented by Dunfield and Krishnaswami to deal with higher rank polymorphism. However, no existing work shows how this idea can be extended naturally to deal with dependencies between existential variables in the presence of dependent types. In this paper, we propose a strategy called *type sanitization* that helps resolve this problem in the framework of type inference in context. We show that type sanitization works on a unification algorithm for a dependently typed system with alpha-equality and first-order constraints. We then further illustrate this strategy to deal with polymorphic subtyping in a higher ranked polymorphic type system.

1 Introduction

Considering unification and/or type inference from a very general perspective helps to establish a common foundation to those kind of problems, which makes it applicable to various programming language features. One of those perspectives is from the point-of-view of information increase. Information increase in type-inference is implemented by having contexts recording all the information including unification variables, with contexts as both the input and the output of the problem. Gundry et al. (2010) presents an implementation of Hindley-Milner (Damas and Milner, 1982; Hindley, 1969) type inference based on this idea to present a methodological understanding of this strategy. Also, Dunfield (2009) uses this strategy to implement a greedy bi-directional typechecking algorithm for inferring polymorphic instances. Later on, Dunfield and Krishnaswami (2013) provide a meta-theoretical foundation for this strategy in a bi-directional algorithm for higher rank polymorphism.

As mentioned by Gundry et al. (2010), a longer-term objective of this strategy is to explain the elaboration of high-level dependently typed programs into fully explicit calculi. However, this objective has not yet been achieved, since unification and/or type inference for dependent type systems is still under-studied, causing difficulties to the application of the type inference in context approach.

Dependent types are increasingly being adopted in many language designs due to their expressiveness (Brady, 2013; Licata and Harper, 2005; McKinna,

* Research Paper. For formal review.

** Student paper.

2006; Norell, 2009; Pasalic et al., 2006; Xi and Pfenning, 1999). However, it is known that complete type inference or unification for many dependent type systems is undecidable. In particular beta-equality, which is commonly employed in dependently type systems, is a major source of difficulty. There is much literature aiming at providing restricted forms of unification for dependent type calculi (Abel and Pientka, 2011; Elliott, 1989; Ziliani and Sozeau, 2015). An important property of the beta-equality based algorithm is that it deals with higher order problems, which is known to be undecidable (Goldfarb, 1981). Therefore in practice, many heuristics are integrated to have acceptable performance. Meanwhile those heuristics make algorithm fairly complicated and even harder to reason about. However beta-equality is not the only problem. Other complications arise from the dependencies introduced by dependent types.

Due to the sophistication of type checking for dependent types in the presence of beta-equality, many recent studies (Kimmell et al., 2012; Sjöberg and Weirich, 2015; Sjöberg et al., 2012; Stump et al., 2009; Sulzmann et al., 2007; van Doorn et al., 2013; Yang et al., 2016) attempt to use explicit casts to manage type-level computation. Type casts are language constructs that control beta-reduction and beta-expansion of the type of an expression. Therefore, type equality in those systems is based on *alpha-equality*, and beta reductions are managed explicitly. While such type systems may lose some convenience over traditional dependent types, there are also some benefits. One particular benefit is that with type casts it is very easy to introduce general recursion without losing various interesting properties (such as decidable type checking).

In this paper, we investigate how to adopt the approach of information increase in a unification algorithm for a dependent type system with explicit type casts. We propose a strategy called *type sanitization* that resolves the dependency problem introduced by dependent types. As we will see, this strategy simplifies the original approach of type inference in context. To show that type sanitization is applicable to other features, we also extend it to *polymorphic type sanitization* that deals with polymorphic subtyping in a higher rank polymorphic type system (Dunfield and Krishnaswami, 2013). Specifically, our main contributions are:

- **A strategy called *type sanitization*** that helps to resolve the dependency between existential variables in the context. This simplifies and at the same time strengthens the approach of type inference in context (Gundry et al., 2010). A key benefit is that information increase can be reasoned in a more syntactic way making the meta-theory easier to study.
- **We show that type sanitization works for dependent types**. This is in contrast to previous work (Dunfield and Krishnaswami, 2013, 2016; Gundry et al., 2010), which does not immediately extend to systems with dependent types.
- **A specification of a unification algorithm** for dependent type systems with control over type-level computations, and first-order constraints. The algorithm is remarkably simple and predictable. We prove that the algorithm is sound and complete.

– **A replacement for higher-rank type instantiation** for an implicitly polymorphic type system with higher rank types (Dunfield and Krishnaswami, 2013). The design of *polymorphic type sanitization* simplifies the subtyping for existential variables, and also removes the problem of duplication in original instantiation, while preserving the completeness and soundness of subtyping.

2 Overview

This section provides background and gives an overview of our work. We discuss the background of type inference in context (Gundry et al., 2010), and also a variant of this approach in a higher rank polymorphic type system (Dunfield and Krishnaswami, 2013). While presenting the key ideas of the work, we also talk about the challenges of each approach, which motivates our work. Then we discuss the key ideas of type sanitization and polymorphic type sanitization as a way to solve those problems.

2.1 Background and Motivation

Type Inference In Context. Gundry et al. (2010) models unification and type inference from a general perspective of information increase. The problem context is a ML-style polymorphic system, based on the invariant that types can only depend on bindings appearing earlier in the context.

Specifically, information and constraints about variables are stored in the context, which is a list maintaining information order. For example[1]:

$\Gamma_1 = \widehat{\alpha}, \widehat{\beta}, x : \widehat{\beta}$

Here $\widehat{\alpha}, \widehat{\beta}$ are existential variables waiting to be solved. Their meanings can be given by solutions, such as:

$\Gamma_2 = \widehat{\alpha}, \widehat{\beta} = \mathit{Int}, x : \widehat{\beta}$

Then the unification problem becomes finding a more informative context that contains solutions for the existential variables so that two expressions are equivalent up to substitution of the solutions. For example, Γ_2 can be the solution context for unifying Int and $\widehat{\beta}$ under Γ_1.

Besides contexts being ordered, a key insight of the approach lies in how to unify existential variables with other types. In this case, unification needs to resolve the dependency between existential variables. Consider unifying $\widehat{\alpha}$ with $\widehat{\beta} \to \widehat{\beta}$ under context $\widehat{\alpha}, \widehat{\beta}, x : \widehat{\beta}$. Here $\widehat{\beta}$ is out of the scope of $\widehat{\alpha}$. The way Gundry et al. solve this problem is to examine variables in the context from the tail to the head, *moving segments of context to the left if necessary*. The process finishes when the existential variable being unified is found. This design is implemented by an additional context that records the context needed to be moved. This can be interpreted from the judgment $\Gamma \shortmid \Xi \vdash \widehat{\alpha} \equiv \tau \dashv \Theta$, which is read as: given input context Γ, Ξ, solve $\widehat{\alpha}$ with τ succeeds and produces an output context Θ.

[1] We adopt our notations and terminologies for the examples

$$\frac{\Gamma \shortmid \Xi \vdash \widehat{\alpha} \equiv \tau \dashv \Theta \qquad v \notin FTV(\widehat{\alpha}, \tau, \Xi)}{\Gamma, v \shortmid \Xi \vdash \widehat{\alpha} \equiv \tau \dashv \Theta} \textsc{Ignore}$$

$$\frac{\Gamma \shortmid \widehat{\beta}, \Xi \vdash \widehat{\alpha} \equiv \tau \dashv \Theta \qquad \widehat{\beta} \in FVT(\tau, \Xi)}{\Gamma, \widehat{\beta} \shortmid \Xi \vdash \widehat{\alpha} \equiv \tau \dashv \Theta} \textsc{Depend} \qquad \frac{}{\Gamma, \widehat{\alpha} \shortmid \Xi \vdash \widehat{\alpha} \equiv \tau \dashv \Gamma, \Xi, \widehat{\alpha} = \tau} \textsc{Define}$$

Fig. 1. Unification between an existential variable and a type (incomplete).

$$\frac{\Gamma_1 \vdash \tau}{\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash \widehat{\alpha} :\leqq \tau \dashv \Gamma_1, \widehat{\alpha} = \tau, \Gamma_2} \textsc{InstLSolve} \qquad \frac{}{\Gamma[\widehat{\alpha}][\widehat{\beta}] \vdash \widehat{\alpha} :\leqq \widehat{\beta} \dashv \Gamma[\widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}]} \textsc{InstLReach}$$

$$\frac{\Gamma[\widehat{\alpha}_2, \widehat{\alpha}_1, \widehat{\alpha} = \widehat{\alpha}_1 \to \widehat{\alpha}_2] \vdash A_1 :\leqq \widehat{\alpha}_1 \dashv \Theta \qquad \Theta \vdash \widehat{\alpha}_2 :\leqq [\Theta](A_2) \dashv \Delta}{\Gamma[\widehat{\alpha}] \vdash \widehat{\alpha} :\leqq A_1 \to A_2 \dashv \Delta} \textsc{InstLArr}$$

$$\frac{}{\Gamma[\widehat{\alpha}][\widehat{\beta}] \vdash \widehat{\beta} :\leqq \widehat{\alpha} \dashv \Gamma[\widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}]} \textsc{InstRReach}$$

Fig. 2. Instantiation between an existential variable and a type (incomplete).

The essential rules involving in this process is given in Figure 1. If a variable is useless for the unification, it is ignored (Ignore). Otherwise, if the variable is needed, but it is out of the scope of $\widehat{\alpha}$, then it is moved to the additional context (Depend). Finally we arrive at the variable we want to unify, which is $\widehat{\alpha}$, we then insert Ξ before $\widehat{\alpha}$, and solve $\widehat{\alpha} = \tau$ (Define). Therefore, the output context for the above unification problem is $\widehat{\beta}, \widehat{\alpha} = \widehat{\beta} \to \widehat{\beta}, x : \widehat{\beta}$. Note that $\widehat{\beta}$ is now placed in the front of $\widehat{\alpha}$.

Challenges. While moving type variables around is a feasible way to resolve the dependency between existential variables, the unpredictable context movements make the information increase hard to formalize and reason about. In this sense, the information increase of contexts is defined in a much *semantic* way: Θ is more informative than Γ, if there exists a substitution S that for every $v \in \Gamma$, we have $\Theta \vdash S(v)$.

This semantic definition makes it hard to prove meta-theory formally, especially when advanced features are involved. For example, in dependent type systems, typing and types/contexts well-formedness are usually coupled, which brings even more complication to the proofs.

Instantiation in Higher Rank Type Systems. Dunfield and Krishnaswami (2013) also use ordered contexts as input and output for type inference for a higher rank polymorphic type system. However, they do it in a more *syntactic* way.

Instead of moving variables to the left in the context, in their subtyping between an existential variable and a type, which they call instantiation, they choose to decompose type constructs so that unification between existential vari-

ables can only happen between single variables. Then there are only two cases to discuss: whether the left existential variable appears first, or the right one appears first. Specifically, Figure 2 shows the key idea of instantiation rules between an existential variable and a type. For space reasons, we only present the rules when the left hand side is an existential variable, but the other case is quite symmetric. We save the formal explanations for notation to Section 3. But we can still get the rough idea there. Notice that in InstLArr, the existential variable $\widehat{\alpha}$ is solved by a function type consisting of two fresh existential variables, and then the function is decomposed to do instantiation successively. Rule InstLReach deals with the case that $\widehat{\alpha}$ appears first, and InstRReach deals with the other case.

In this way, the information increase of contexts, which is named context extension, is formalized in an intuitive and straightforward *syntactic* way, which enables them to prove the meta-theory thoroughly and formally. The complete definition of context extension can be found in the appendix. Our definition presented in Section 3.4 also mimics their definition.

Challenges. While destructing type constructors makes perfect sense in their setting, it cannot deal with dependent types correctly. For example, given the context $\widehat{\alpha}, \widehat{\beta}$, suppose that we want to unify $\widehat{\alpha}$ with a dependent type $\Pi x : \widehat{\beta}.\ x$. Here because $\widehat{\beta}$ appears after $\widehat{\alpha}$, we cannot directly derive $\widehat{\alpha} = \Pi x : \widehat{\beta}.\ x$ which is ill typed. However, if we try to decompose this Pi type, then according to rule InstLArr, it is obvious that $\widehat{\alpha}_2$ should be solved by x. In order to make the solution well typed, we need to put x before $\widehat{\alpha}_2$ into the context. However, this means x will remain in the context, and it is available for any later existential variable that should not have access to x. In essence the strategy for simple function types presented by Dunfield and Krishnaswami (2013) does not work for dependent function types.

Another drawback of decomposition is that it produces duplication. The rules in Figure 2 is repeated for the cases when the existential variable is on the right. For example, there will be a symmetric InstRArr corresponding to InstLArr. Worse, this kind of "duplication" would scales up with the type constructs in the system.

2.2 Our Approach: Type Sanitization

Type sanitization provides another way to resolve the dependency between existential variables. It combines the advantages of the previous two approaches. First, it is a simple and quite predictable process, so that information increase can still be modeled as *syntactic* context extension. Also, it will not decompose types, nor cause any duplication.

To understand how type sanitization works, we revisit the unification problem: given context

$\widehat{\alpha}, \widehat{\beta}, x : \widehat{\beta}$

we want to unify $\widehat{\alpha}$ with $\widehat{\beta} \to \widehat{\beta}$. The problem here is that $\widehat{\beta}$ is out of the scope of $\widehat{\alpha}$. Therefore, we first "sanitize" the type $\widehat{\beta} \to \widehat{\beta}$. The process of type sanitization

will sanitize the existential variables in right hand side that are out of the scope of $\widehat{\alpha}$ by solving them with fresh existential variables added to the front of $\widehat{\alpha}$. Specifically, we will solve $\widehat{\beta}$ with a fresh variable $\widehat{\alpha}_1$, which results in an output context

$\widehat{\alpha}_1, \widehat{\alpha}, \widehat{\beta} = \widehat{\alpha}_1, x : \widehat{\beta}$

Notice that $\widehat{\alpha}_1$ is inserted right before $\widehat{\alpha}$. Now the unification problem becomes unifying $\widehat{\alpha}$ with $\widehat{\alpha}_1 \to \widehat{\alpha}_1$, and $\widehat{\alpha}_1 \to \widehat{\alpha}_1$ is a valid solution for $\widehat{\alpha}$. Therefore, we get a final solution context:

$\widehat{\alpha}_1, \widehat{\alpha} = \widehat{\alpha}_1 \to \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1, x : \widehat{\beta}$.

Interpretation of Type Sanitization. Moving existential variables around in the approach of type inference in context (Gundry et al., 2010), the symmetric rules InstLReach and InstRReach (Dunfield and Krishnaswami, 2013), and the approach of type sanitization all enjoy the same philosophy: *for an unsolved existential variable $\widehat{\alpha}$, the relative order between $\widehat{\alpha}$ and another unsolved existential variable $\widehat{\beta}$ does not matter.*

This seems to go against the design principle that the contexts are ordered lists. However, keeping order is still important for variables *whose order matter*. For instance, for polymorphic types, the order between existential variables ($\widehat{\alpha}$) and type variables (α) is important, so we cannot unify $\widehat{\alpha}$ with α under the context $(\widehat{\alpha}, \alpha)$ since α is not in the scope of $\widehat{\alpha}$. A similar reason exists in dependent type systems: we cannot unify $\widehat{\alpha}$ with x if x appears behind $\widehat{\alpha}$ in the context.

The task of type sanitization is to "move" existential variables to suitable positions in a roundabout way: solving the existential variables with a fresh existential variables and making sure that these new fresh variables appear in a suitable position.

2.3 Unification for Dependent Types

As a first illustration of the utility of the type sanitization, we present a unification algorithm for dependent types with alpha-equality based first-order constrains.

Explicit Casts. Type systems with explicit type casts on type-level computation have been adopted in several dependent type calculi (Kimmell et al., 2012; Sjöberg and Weirich, 2015; Sjöberg et al., 2012; Stump et al., 2009; Sulzmann et al., 2007; van Doorn et al., 2013; Yang et al., 2016). In order to have type-level computations, explicit casts are needed to trigger the computation. In this work we adopt the approach by Yang et al. (2016), where type casts can be triggered by two language constructs: $\mathsf{cast}_{\downarrow}\ e$ that does one-step beta reduction on the type of e, and $\mathsf{cast}_{\uparrow}\ e$ that does one-step beta expansion on the type of e. For example, given

$e : (\lambda x : \mathit{Int}.\ \star)\ 3$

then we have

$\mathsf{cast}_{\downarrow}\ e : \star$

$\mathsf{cast}_{\uparrow}\ (\mathsf{cast}_{\downarrow}\ e) : (\lambda x : \mathit{Int}.\ \star)\ 3$

In type systems with such type casts, type comparison is naturally based on alpha-equality. This simplifies the unification algorithm in the sense that

unification can be mostly structural. However, we still need to deal with the dependency introduced by dependent types carefully, which is mainly reflected in the unification problems between existential variables.

Type Sanitization In Dependent Type System. Type sanitization is applicable to dependently type systems. Consider the previous example, where we unify $\widehat{\alpha}$ with $\Pi x : \widehat{\beta}.\ x$. By a similar process described in Section 2.2, we can sanitize the type to be $\Pi x : \widehat{\alpha}_1.\ x$ without destructing the Pi type, and solve $\widehat{\alpha}$ with $\Pi x : \widehat{\alpha}_1.\ x$.

Based on type sanitization, we come up with a unification algorithm, which is later proved to be sound and complete.

2.4 Polymorphic Type Sanitization For Subtyping

Instead of unification, the instantiation relation in Dunfield and Krishnaswami (2013) is actually aiming at dealing with the polymorphic subtyping relation between existential variables and other types. Here we say that type A is a subtype of B if and only if A is more polymorphic than B. The difficulty of subtyping is that it needs to take unification into account at the same time. For example, given that $\widehat{\alpha}$ is a subtype of Int, then the only possible solution is $\widehat{\alpha} = Int$. And if given $\forall a.a \to a$ is a subtype $\widehat{\alpha}$, then a feasible solution can be $\widehat{\alpha} = Int \to Int$.

The type sanitization we described above only works for unification. If given the context

$\widehat{\alpha}$

and that $\forall \alpha.\alpha \to \alpha$ is a subtype of $\widehat{\alpha}$, how can we sanitize the polymorphic type into a valid solution for an existential variable while solutions can only be monotypes? One observation is that, the most general solution for this subtyping problem is $\widehat{\alpha} = \widehat{\beta} \to \widehat{\beta}$ for fresh $\widehat{\beta}$. Namely, we remove the universal quantifier and replace the variable a with a fresh existential variable that should be in the scope of $\widehat{\alpha}$, which results in the solution context:

$\widehat{\beta}, \widehat{\alpha} = \widehat{\beta} \to \widehat{\beta}$

From this observation, we extend type sanitization to polymorphic type sanitization, which is able to resolve the polymorphic subtyping relation for existential variables. Function types are contra-variant in argument types, and co-variant in return types. For example, we have that:

$(Int \to Int) \to Int \ <: \ (\forall \alpha.\alpha \to \alpha) \to Int$

$\forall \alpha.\alpha \to \alpha \ <: \ Int \to Int$

According to the position the universal quantifier appears, polymorphic type sanitization has two modes, which we call contra-variant mode and co-variant mode respectively. In contra-variant mode, the universal quantifier is replaced by a fresh existential variable, while in co-variant mode, it is put in the context as a common type variable.

We show that the original instantiation relationship in Dunfield and Krishnaswami (2013) can be replaced by our polymorphic type sanitization process, while subtyping remains sound and complete.

$$
\begin{array}{rlrl}
[\emptyset](\sigma) & = \sigma & [\Gamma, x:\tau](\sigma) & = [\Gamma](\sigma) \\
[\Gamma, \widehat{\alpha}](\sigma) & = [\Gamma](\sigma) & [\Gamma, \widehat{\alpha} = \tau](\sigma) & = [\Gamma](\sigma[\widehat{\alpha} \mapsto \tau])
\end{array}
$$

Fig. 3. Context application.

3 Unification and Type Sanitization for Dependent Types

This section introduces a simple dependently typed calculus with alpha-equality and type casts. The main novelties are the unification and type sanitization mechanisms.

3.1 Language Overview

The syntax of the calculus is shown below:

$$
\begin{array}{lrl}
\text{Expressions} & e & ::= x \mid \star \mid e_1\ e_2 \mid \lambda x:\sigma.\ e \mid \Pi x:\sigma_1.\ \sigma_2 \\
 & & \mid\ \mathsf{cast}_{\uparrow}\ e \mid \mathsf{cast}_{\downarrow}\ e \\
\text{Types} & \tau, \sigma & ::= e \mid \widehat{\alpha} \\
\text{Contexts} & \Gamma, \Theta, \Delta & ::= \varnothing \mid \Gamma, x:\sigma \mid \Gamma, \widehat{\alpha} \mid \Gamma, \widehat{\alpha} = \tau \\
\text{Complete Contexts} & \Omega & ::= \varnothing \mid \Omega, x:\sigma \mid \Omega, \widehat{\alpha} = \tau
\end{array}
$$

Expressions. Expressions e include variables x, a single sort $\star$ to represent the type of types, applications $e_1\ e_2$, functions $\lambda x:\sigma.\ e$, Pi types $\Pi x:\sigma_1.\ \sigma_2$, $\mathsf{cast}_{\uparrow}\ e$ that does beta expansion, and $\mathsf{cast}_{\downarrow}\ e$ that does beta reduction.

Types. Types τ, σ are the same as expressions, except that types contain existential variables $\widehat{\alpha}$. The categories of expressions and types are stratified to make sure that existential variables only appears in positions where types are expected.

Contexts. Contexts Γ are an ordered list of variables and existential variables, which can either be unsolved ($\widehat{\alpha}$) or solved by a type τ ($\widehat{\alpha} = \tau$). It is important for a context to be ordered to solve the dependency between variables. Complete contexts Ω only contain variables and solved existential variables.

Hole notation. We use hole notations like $\Gamma[x]$ (Dunfield and Krishnaswami, 2013) to denote that the variable x appears in the context. Sometimes such a hole notation is also written as Γ_1, x, Γ_2. Multiple holes also keep the order. For example, $\Gamma[x][\widehat{\alpha}]$ not only requires the existence of both variables x and $\widehat{\alpha}$, but also requires that x appears before $\widehat{\alpha}$. The hole notation is also used for replacement and modification. For example, $\Gamma[\widehat{\alpha} = \star]$ means the context is mostly unchanged except that $\widehat{\alpha}$ now is solved by $\star$.

Applying Contexts. Since the context records all the solutions of solved existential variables, it can be used as a substitution. Figure 3 defines the substitution process, where all solved existential variables are substituted by their solutions.

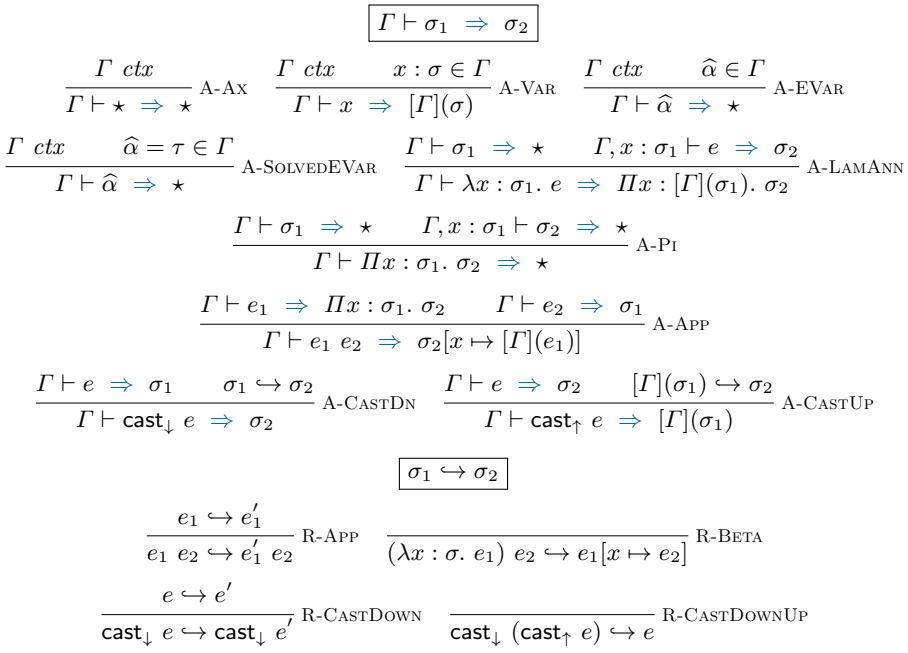

Fig. 4. Typing and semantics.

3.2 Typing in Detail

In order to show that our unification algorithm works correctly, we need to make sure that inputs to the algorithm are well-formed. In a dependent type system, the well-formedness of types and contexts are relying on typing judgments. Therefore, to introduce the well-formedness of type and contexts, we first introduce the typing rules.

Before we give the typing rules, we need to consider: what does it mean for an input type to the unification algorithm to be well-formed? For example, Given a context $(\widehat{\alpha}, x : \widehat{\alpha})$, is the type $((\lambda y : \mathit{Int}.\ \star)\ x)$ well formed? Here, the type requests solving $\widehat{\alpha} = \mathit{Int}$. We do not regard this type well formed, because we keep the invariant: *the constraints contained in the input type have already been solved.* Namely, the unification process only accepts inputs that are already type checked in current context. In this example, this type is well-formed under context $(\widehat{\alpha} = \mathit{Int}, x : \widehat{\alpha})$. This can also help prevent ill-formed contexts that contains conflicting constraints, for example, the context:

$\widehat{\alpha}, x : \widehat{\alpha}, y : ((\lambda x : \mathit{Int}.\ \star)\ x), z : ((\lambda x : \mathit{Bool}.\ \star)\ x)$

contains two constraints that request $\widehat{\alpha}$ to be solved by *Int* and *Bool* respectively, which cannot be satisfied at the same time.

Type System Given this interpretation of well-formedness, the typing rules that serve specifically for well-formedness is shown at the top of Figure 4. The judgment $\Gamma \vdash \sigma_1 \Rightarrow \sigma_2$ is read as: under the context Γ, the type σ_1 has type σ_2.

Rule A-Ax states that $\star$ always has type $\star$. Rule A-Var acquires the type of the variable from the typing context, and applies the context to the type. This reveals another invariant that we keep: *the typing output is always fully substituted under current context.* Rules A-EVar and A-SolvedEVar ensures that existential variables always have type $\star$. Rule A-LamAnn first infers the type $\star$ for the annotation, then puts $x : \sigma_1$ into the typing context to infer the body. To make sure output type is fully substituted, we apply the context to σ_1 in the output Pi type. Rule A-Pi infers the type $\star$ for the argument type σ_1, then puts $x : \sigma_1$ into the typing context to infer σ_2, whose type is also a $\star$. And the result type for a Pi type is $\star$. Rule A-App first infers a function type for e_1, and then infers e_2 to have the argument type. Again, to maintain the invariant, we apply the context to e_1 before substituting x with e_1. Rule A-CastDn infers the type σ_1 of e, that reduce the type σ_1 to σ_2, while rule A-CastUp finds a fully substituted type $[\Gamma](\sigma_1)$ that reduces to σ_2 as the output type. The call by name reduction is defined at the bottom of Figure 4. Due to the design of the rule A-CastUp, the typing rules are non-deterministic, which does not matter for our purpose: the typing is only used in propositions (such as lemmas, theories) but it never appears in the unification and type sanitization algorithms.

Context well-formedness. The first four typing rules have a common precondition $\Gamma \ ctx$, which requests the context is well-formed. The judgment is defined at the top of Figure 5. Rule AC-Empty states that an empty context is always well formed. Rule AC-Var requires x fresh, and the type annotation is typed with $\star$. Rule AC-EVar and AC-SolvedEVar are defined in a similar way.

Type well-formedness. We denote a well-formed type σ as $\Gamma \vdash \sigma \Rightarrow \star$. We will also sometimes write it as $\Gamma \vdash \sigma$. A weaker version of type well-formedness, which is type well-scopedness, written as $\Gamma \vDash \sigma$, is defined at the bottom of Figure 5. Well-scopedness of types only requires all the variables involved in a type are bound in the typing contexts.

3.3 Unification

As we mentioned before, our unification is based on alpha-equality. So in most cases, the unification rules are intuitively structural. The most difficult one which is also the most essential one, is how to unify an existential variable with another type. In this section, we first present the judgment of unification, then we discuss those cases before we present the unification process.

Judgment of unification. Due to our design choice, there are two modes in the unification: expressions, and types. The expression mode ($\delta = e$) does unification between expressions, while the type mode ($\delta = \sigma$) does unification between types. The judgment of unification problem is formalized as:

$$\Gamma \vdash_\delta \sigma_1 \simeq \sigma_2 \dashv \Theta$$

The input of the unification is a context Γ, and two types ($\delta = \sigma$) or two expressions ($\delta = e$). The output of the unification is a new context Θ which extends the original context with probably more new existential variables or more

$$\boxed{\Gamma \ ctx}$$

$$\frac{}{\varnothing \ ctx}\ \text{AC-Empty} \qquad \frac{\Gamma \ ctx \qquad x \notin \Psi \qquad \Gamma \vdash \sigma \ \Rightarrow \ \star}{\Gamma, x : \sigma \ ctx}\ \text{AC-Var}$$

$$\frac{\Gamma \ ctx \qquad \widehat{\alpha} \notin \Psi}{\Gamma, \widehat{\alpha} \ ctx}\ \text{AC-EVar} \qquad \frac{\Gamma \ ctx \qquad \widehat{\alpha} \notin \Gamma \qquad \Gamma \vdash \tau \ \Rightarrow \ \star}{\Gamma, \widehat{\alpha} = \tau \ ctx}\ \text{AC-SolvedEVar}$$

$$\boxed{\Gamma \vDash \sigma}$$

$$\frac{x : \sigma \in \Gamma}{\Gamma \vDash x}\ \text{WS-Var} \qquad \frac{\widehat{\alpha} \in \Gamma}{\Gamma \vDash \widehat{\alpha}}\ \text{WS-EVar} \qquad \frac{\widehat{\alpha} = \tau \in \Gamma}{\Gamma \vDash \widehat{\alpha}}\ \text{WS-SolvedEVar}$$

$$\frac{\Gamma \vDash \sigma_1 \qquad \Gamma, x : \sigma_1 \vDash \sigma_2}{\Gamma \vDash \Pi x : \sigma_1.\ \sigma_2}\ \text{WS-Pi} \qquad \frac{\Gamma \vDash \sigma \qquad \Gamma, x : \sigma \vDash e}{\Gamma \vDash \lambda x : \sigma.\ e}\ \text{WS-LamAnn}$$

$$\frac{\Gamma \vDash e_1 \qquad \Gamma \vDash e_2}{\Gamma \vDash e_1\ e_2}\ \text{WS-App} \qquad \frac{\Gamma \vdash e}{\Gamma \vDash \mathsf{cast}_{\downarrow}\ e}\ \text{WS-CastDn} \qquad \frac{\Gamma \vdash e}{\Gamma \vDash \mathsf{cast}_{\uparrow}\ e}\ \text{WS-CastUp}$$

Fig. 5. Context well-formedness and type well-scopedness.

existing existential variables solved. The formal definition of context extension is discussed in Section 3.4. Following is an example of a unification problem:

$$\widehat{\alpha} \vdash_{\sigma} \widehat{\alpha} \ \simeq \ \mathit{Int} \dashv \widehat{\alpha} = \mathit{Int}$$

where we want to unify $\widehat{\alpha}$ with *Int* under the input context $\widehat{\alpha}$, which results in the output context $\widehat{\alpha} = \mathit{Int}$ that solves $\widehat{\alpha}$ with *Int*.

For a valid unification problem, it must have the invariant: $[\Gamma]\tau_1 = \tau_1$, and $[\Gamma]\tau_2 = \tau_2$. Namely, *the input types must be fully applied under the input context.* So the following is not a valid unification problem input:

$$\widehat{\alpha} = \mathit{Bool} \vdash_{\sigma} \widehat{\alpha} \ \simeq \ \mathit{Int}$$

We assume this invariant is given with the inputs at the beginning, and the unification process would maintain it through the whole formalization.

Process of type sanitization. As we discuss in Section 2, before unifying an existential variable with a type, we first sanitize the type so that the existential variables in the type that are out of the scope are solved by fresh existential variables within the scope. We call this process *type sanitization*, which is formally defined in Figure 6. The judgment $\Gamma[\widehat{\alpha}] \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta$ is interpreted as: under the context Γ, which contains an existential variable $\widehat{\alpha}$, we sanitize all the existential variables in the type τ_1 that appears before $\widehat{\alpha}$, which results in a sanitized type τ_2 and an output context Θ. Computationally, there are three inputs Γ, $\widehat{\alpha}$ and τ_1, with two outputs τ_2 and Θ. We sometimes omit $\widehat{\alpha}$ if it is clear which existential variable is being referred.

The most interesting cases are I-EVarAfter and I-EVarBefore. In I-EVarAfter, because $\widehat{\beta}$ appears after $\widehat{\alpha}$, we create a fresh existential variable $\widehat{\alpha}_1$, which is put before $\widehat{\alpha}$, and solve $\widehat{\beta}$ by $\widehat{\alpha}_1$. In I-EVarBefore, because $\widehat{\beta}$ is in the scope of $\widehat{\alpha}$, we leave the existential variable unchanged. The remaining rules

$$\boxed{\Gamma[\widehat{\alpha}] \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta}$$

$$\frac{}{\Gamma[\widehat{\alpha}][\widehat{\beta}] \vdash \widehat{\beta} \longrightarrow \widehat{\alpha}_1 \dashv \Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}_1]}\ \text{I-EVarAfter}$$

$$\frac{}{\Gamma[\widehat{\beta}][\widehat{\alpha}] \vdash \widehat{\beta} \longrightarrow \widehat{\beta} \dashv \Gamma[\widehat{\beta}][\widehat{\alpha}]}\ \text{I-EVarBefore} \qquad \frac{}{\Gamma \vdash x \longrightarrow x \dashv \Gamma}\ \text{I-Var}$$

$$\frac{}{\Gamma \vdash \star \longrightarrow \star \dashv \Gamma}\ \text{I-Star} \qquad \frac{\Gamma \vdash e_1 \longrightarrow e_3 \dashv \Theta_1 \qquad \Theta_1 \vdash [\Theta_1](e_2) \longrightarrow e_4 \dashv \Theta}{\Gamma \vdash e_1\ e_2 \longrightarrow e_3\ e_4 \dashv \Theta}\ \text{I-App}$$

$$\frac{\Gamma \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta_1 \qquad \Theta_1, x : \tau_1 \vdash [\Theta_1](e_1) \longrightarrow e_2 \dashv \Theta, x : \tau_1}{\Gamma \vdash \lambda x : \tau_1.\ e_1 \longrightarrow \lambda x : \tau_2.\ e_2 \dashv \Theta}\ \text{I-LamAnn}$$

$$\frac{\Gamma \vdash \tau_1 \longrightarrow \tau_3 \dashv \Theta_1 \qquad \Theta_1, x : \tau_1 \vdash [\Theta_1](\tau_2) \longrightarrow \tau_4 \dashv \Theta, x : \tau_1}{\Gamma \vdash \Pi x : \tau_1.\ \tau_2 \longrightarrow \Pi x : \tau_3.\ \tau_4 \dashv \Theta}\ \text{I-Pi}$$

$$\frac{\Gamma \vdash e_1 \longrightarrow e_2 \dashv \Theta}{\Gamma \vdash \mathsf{cast}_{\downarrow}\ e_1 \ \simeq\ \mathsf{cast}_{\downarrow}\ e_2 \dashv \Theta}\ \text{I-CastDn} \qquad \frac{\Gamma \vdash e_1 \longrightarrow e_2 \dashv \Theta}{\Gamma \vdash \mathsf{cast}_{\uparrow}\ e_1 \ \simeq\ \mathsf{cast}_{\uparrow}\ e_2 \dashv \Theta}\ \text{I-CastUp}$$

Fig. 6. Type sanitization.

are structural. Similarly to unification, we always apply intermediate output contexts to the input types to maintain the invariant that the types are fully substituted under current contexts.

The sanitization process is remarkably simple, while it solves exactly what we want: resolve the order of existential variables so that we can focus on the order that really matters.

Process of unification. Based on type sanitization, Figure 7 presents the unification rules. Rule U-AEq corresponds to the case when two types are already alpha-equivalent. Most of the rest rules are structural. The two most subtle cases are rules U-EVarTy and U-TyEVar, which correspond respectively to when the existential variable is on the left and on the right. We go through the first rule. There are three preconditions. First is the occurs check, which is to make sure $\widehat{\alpha}$ does not appear in the free variables of τ_1. Then we use type sanitization to make sure all the existential variables in τ_1 that are out of scope of $\widehat{\alpha}$ are turned into fresh ones that are in the scope of $\widehat{\alpha}$. This process gives us the output type τ_2, and output context $\Theta_1, \widehat{\alpha}, \Theta_2$. Finally, τ_2 could also contain variables whose order matters, so we use $\Theta_1 \vDash \tau_2$ to make sure τ_2 is well scoped. Rule U-TyEVar is symmetric to U-EVarTy. Using well-scopedness instead of well-formedness gets us rid of the dependency on typing.

Example. The derivation of the unification problem $\Gamma, \widehat{\alpha}, \widehat{\beta} \vdash_\sigma \widehat{\alpha} \ \simeq\ \Pi x : \widehat{\beta}.\ x$ is given below. For clarity, we denote $\Theta = \Gamma, \widehat{\alpha}_1, \widehat{\alpha}, \widehat{\beta} = \widehat{\alpha}_1$. And it is easy to verify $\widehat{\alpha} \notin FV(\Pi x : \widehat{\beta}.\ x)$.

$$\dfrac{\dfrac{\dfrac{}{\Gamma, \widehat{\alpha}, \widehat{\beta} \vdash \widehat{\beta} \longmapsto \widehat{\alpha}_1 \dashv \Theta}\ \text{I-Evar2} \qquad \dfrac{}{\Theta \vdash x \longmapsto x \dashv \Theta}\ \text{I-Var}}{\Gamma, \widehat{\alpha}, \widehat{\beta} \vdash \Pi x : \widehat{\beta}.\ x \longmapsto \Pi x : \widehat{\alpha}_1.\ x \dashv \Theta}\ \text{I-Pi} \qquad \dfrac{}{\Gamma, \widehat{\alpha}_1 \vDash \Pi x : \widehat{\alpha}_1.\ x}}{\Gamma, \widehat{\alpha}, \widehat{\beta} \vdash_\sigma \widehat{\alpha} \ \simeq\ \Pi x : \widehat{\beta}.\ x \dashv \Gamma, \widehat{\alpha}_1, \widehat{\alpha} = \Pi x : \widehat{\alpha}_1.\ x, \widehat{\beta} = \widehat{\alpha}_1}\ \text{U-EvarTy}$$

$$\boxed{\Gamma \vdash_\delta \sigma_1 \simeq \sigma_2 \dashv \Theta}$$

$$\frac{}{\Gamma \vdash_\delta \sigma \simeq \sigma \dashv \Gamma}\ \text{U-AEq}$$

$$\frac{\widehat{\alpha} \notin FV(\tau_1) \qquad \Gamma[\widehat{\alpha}] \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta_1, \widehat{\alpha}, \Theta_2 \qquad \Theta_1 \vDash \tau_2}{\Gamma[\widehat{\alpha}] \vdash_\sigma \widehat{\alpha} \simeq \tau_1 \dashv \Theta_1, \widehat{\alpha} = \tau_2, \Theta_2}\ \text{U-EVarTy}$$

$$\frac{\widehat{\alpha} \notin FV(\tau_1) \qquad \Gamma[\widehat{\alpha}] \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta_1, \widehat{\alpha}, \Theta_2 \qquad \Theta_1 \vDash \tau_2}{\Gamma[\widehat{\alpha}] \vdash_\sigma \tau_1 \simeq \widehat{\alpha} \dashv \Theta_1, \widehat{\alpha} = \tau_2, \Theta_2}\ \text{U-TyEVar}$$

$$\frac{\Gamma \vdash_e e_1 \simeq e_3 \dashv \Theta_1 \qquad \Theta_1 \vdash_e [\Theta_1](e_2) \simeq [\Theta_1](e_4) \dashv \Theta}{\Gamma \vdash_\delta e_1\ e_2 \simeq e_3\ e_4 \dashv \Theta}\ \text{U-App}$$

$$\frac{\Gamma \vdash_\sigma \sigma_1 \simeq \sigma_2 \dashv \Theta_1 \qquad \Theta_1, x : \sigma_1 \vdash_e [\Theta_1](e_1) \simeq [\Theta_1](e_2) \dashv \Theta, x : \sigma_1}{\Gamma \vdash_e \lambda x : \sigma_1.\ e_1 \simeq \lambda x : \sigma_2.\ e_2 \dashv \Theta}\ \text{U-LamAnn}$$

$$\frac{\Gamma \vdash_\sigma \sigma_1 \simeq \sigma_3 \dashv \Theta_1 \qquad \Theta_1, x : \sigma_1 \vdash_\sigma [\Theta_1](\sigma_2) \simeq [\Theta_1](\sigma_4) \dashv \Theta, x : \sigma_1}{\Gamma \vdash_\delta \Pi x : \sigma_1.\ \sigma_2 \simeq \Pi x : \sigma_3.\ \sigma_4 \dashv \Theta}\ \text{U-Pi}$$

$$\frac{\Gamma \vdash_e e_1 \simeq e_2 \dashv \Theta}{\Gamma \vdash_\delta \mathsf{cast}_\downarrow\ e_1 \simeq \mathsf{cast}_\downarrow\ e_2 \dashv \Theta}\ \text{U-CastDn} \qquad \frac{\Gamma \vdash_e e_1 \simeq e_2 \dashv \Theta}{\Gamma \vdash_e \mathsf{cast}_\uparrow\ e_1 \simeq \mathsf{cast}_\uparrow\ e_2 \dashv \Theta}\ \text{U-CastUp}$$

Fig. 7. Unification.

3.4 Context Extension

We mentioned that the algorithmic output context extends the input context with new existential variables or more existential variables solved. To accurately capture this kind of information increase, we present the definition of context extension in Figure 8. This definition is similar to the one by Dunfield and Krishnaswami (2013).

The empty context is an extension of itself (CE-Empty). Variables or existential variables are preserved during the extension (CE-Var, CE-EVar), while the solutions for existential variables can be different only if they are equivalent under context application (CE-SolvedEVar). The definition in Dunfield and Krishnaswami (2013) further allows the type annotation to change (in CE-Var), which is not necessary for our algorithm. The extension can also add solutions to unsolved existential variables (CE-Solve), or add new existential variables (CE-Add, CE-AddSolved).

Context application on contexts. Complete contexts Ω are contexts with all existential variables solved, as defined in Section 3.1. Applying a complete context to a well-formed type σ yields a type without existential variables $[\Omega](\sigma)$. Similarly, given $\Gamma \to \Omega$, we can get a context without existential variables $[\Omega](\Gamma)$. The formal definition of context application is in Figure 9.

3.5 Soundness

Our overall framework for proofs is quite similar as Dunfield and Krishnaswami (2013). However, unlike their work which always implicitly assumes that con-

$$\boxed{\Gamma \longrightarrow \Theta}$$

$$\frac{}{\varnothing \longrightarrow \varnothing}\ \textsc{CE-Empty} \qquad \frac{\Gamma \longrightarrow \Delta}{\Gamma, x : \sigma \longrightarrow \Delta, x : \sigma}\ \textsc{CE-Var} \qquad \frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta}{\Gamma, \widehat{\alpha} \longrightarrow \Delta, \widehat{\alpha}}\ \textsc{CE-EVar}$$

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta \qquad [\Delta](\tau_1) = [\Delta](\tau_2)}{\Gamma, \widehat{\alpha} = \tau_1 \longrightarrow \Delta, \widehat{\alpha} = \tau_2}\ \textsc{CE-SolvedEVar}$$

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta \qquad \Delta \vdash \tau}{\Gamma, \widehat{\alpha} \longrightarrow \Delta, \widehat{\alpha} = \tau}\ \textsc{CE-Solve} \qquad \frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta}{\Gamma \longrightarrow \Delta, \widehat{\alpha}}\ \textsc{CE-Add}$$

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta \qquad \Delta \vdash \tau}{\Gamma \longrightarrow \Delta, \widehat{\alpha} = \tau}\ \textsc{CE-AddSolved}$$

Fig. 8. Context extension.

$$\begin{aligned}
[\emptyset](\emptyset) &= \emptyset & [\Omega, x : \tau](\Gamma, x : \tau) &= [\Omega](\Gamma), x : [\Omega](\tau) \\
[\Omega, \widehat{\alpha} = \tau](\Gamma, \widehat{\alpha}) &= [\Omega](\Gamma) & [\Omega, \widehat{\alpha} = \tau_1](\Gamma, \widehat{\alpha} = \tau_2) &= [\Omega](\Gamma) \quad \text{If } [\Omega](\tau_1) = [\Omega](\tau_2) \\
[\Omega, \widehat{\alpha} = \tau_1](\Gamma) &= [\Omega](\Gamma)
\end{aligned}$$

Fig. 9. Context application.

texts and types are well-formed, we put extra efforts on dealing with the well-formedness and the typing explicitly since we have to resolve the dependency carefully.

We proved that our type sanitization strategy and the unification algorithm are sound. First, we show that except for resolving the order problem, type sanitization will not change the type. Namely, the input type and the output type are equivalent after substitution by the output context:

Lemma 1 (Type Sanitization Equivalence). *If* $\Gamma \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta$*, and* $\Gamma \vdash \tau_1 \Rightarrow \sigma$*, then* $[\Theta](\tau_1) = [\Theta](\tau_2)$.

Moreover, if the input type is well-formed under the input context, then the output type is still well-formed under the output context:

Lemma 2 (Type Sanitization Well Formedness). *If* $\Gamma \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta$*, and* $\Gamma \vdash \tau_1 \Rightarrow \sigma$*, then* $\Theta \vdash \tau_2 \Rightarrow [\Theta](\sigma)$.

Having those lemmas related to type sanitization, we can prove that two input types are really unified by the unification algorithm:

Lemma 3 (Unification Equivalence). *If* $\Gamma \vdash_\delta \tau_1 \simeq \tau_2 \dashv \Theta$*, and* $\Gamma \vdash \tau_1 \Rightarrow \sigma_1'$*, and* $\Gamma \vdash \tau_2 \Rightarrow \sigma_2'$*, then* $[\Theta](\tau_1) = [\Theta](\tau_2)$.

3.6 Completeness

We use the notation $\Gamma \vdash \tau_1 = \tau_2$ to mean that $\Gamma \vdash \tau_1 \Rightarrow \sigma$, $\Gamma \vdash \tau_2 \Rightarrow \sigma$ for some σ, and $\tau_1 = \tau_2$.

The completeness of type sanitization is proved by a more general lemma, which can be found in the appendix. Here we present a more readable version:

Corollary 1 (Type Sanitization Completeness Pretty). *Given* $\Gamma \longrightarrow \Omega$, *where* $\Gamma = \Gamma_1, \widehat{\alpha}, \Gamma_2$, *and* $[\Gamma](\sigma_1) = \sigma_1$, *and* $[\Omega](\tau) = \tau$, *and* $\widehat{\alpha} \notin FV(\sigma_1)$, *and* $\Gamma_1 \vDash \tau$. *If* $[\Omega](\Gamma) \vdash \tau = [\Omega](\sigma_1)$, *then there are* Δ, Ω' *such that* $\Omega \longrightarrow \Omega'$, *and* $\Delta \longrightarrow \Omega'$, *and* $\Gamma[\widehat{\alpha}] \vdash \sigma_1 \longrightarrow \sigma_2 \dashv \Delta$, *where* $\Delta = \Delta_1, \widehat{\alpha}, \Delta_2$, *and* $\Delta_1 \vDash \sigma_2$.

Having the completeness of type sanitization, we are ready to prove that our unification algorithm is complete:

Lemma 4 (Unification Completeness). *Given* $\Gamma \longrightarrow \Omega$, *and* $[\Gamma](\sigma_1) = \sigma_1$, *and* $[\Gamma](\sigma_2) = \sigma_2$. *If* $[\Omega](\Gamma) \vdash [\Omega](\sigma_1) = [\Omega](\sigma_2) \Rightarrow \tau$, *then there are* Δ, Ω' *such that* $\Omega \longrightarrow \Omega'$, *and* $\Delta \longrightarrow \Omega'$, *and* $\Gamma \vdash_\delta \sigma_1 \simeq \sigma_2 \dashv \Delta$ *for any* δ *suitable.*

4 Higher Rank Polymorphic Type System

In this section, we adopt the type sanitization strategy to a higher rank polymorphic type system from Dunfield and Krishnaswami (2013). We show that type sanitization can be further extended to polymorphic type sanitization to deal with subtyping, which can be used to replace the instantiation relation in original system while preserving the completeness and subtyping.

4.1 Language

The syntax of Dunfield and Krishnaswami (2013) is given below. Notice that notations from Section 3 are reused here. We will always make it clear from the context which system is being referred.

Type	$A, B ::= 1 \mid \alpha \mid \widehat{\alpha} \mid \forall \alpha.A \mid A \to B$
Monotype	$\sigma, \tau ::= 1 \mid \alpha \mid \widehat{\alpha} \mid \sigma \to \tau$
Contexts	$\Gamma, \Theta, \Delta ::= \varnothing \mid \Gamma, \alpha \mid \Gamma, x : A \mid \Gamma, \widehat{\alpha} \mid \Gamma, \widehat{\alpha} = \tau \mid \Gamma, \blacktriangleright_{\widehat{\alpha}}$
Complete Contexts	$\Omega ::= \varnothing \mid \Omega, \alpha \mid \Omega, x : A \mid \Omega, \widehat{\alpha} = \tau \mid \Omega, \blacktriangleright_{\widehat{\alpha}}$

Types. Types A, B include unit type 1, type variables α, existential variables $\widehat{\alpha}$, polymorphic types $\forall \alpha.A$ and function types $A \to B$.

Monotypes σ, τ are a special kind of types without universal quantifiers.

Contexts. Contexts Γ are an ordered list of type variables α, variables $x : A$, existential variables $\widehat{\alpha}$ and $\widehat{\alpha} = \tau$, and special markers $\blacktriangleright_{\widehat{\alpha}}$ for scoping reasons. All existential variables in complete contexts Ω are solved.

4.2 Subtyping

The original definition of algorithmic subtyping is given in Figure 10. The judgment $\Gamma \vdash A <: B \dashv \Theta$ is interpreted as: given the context Γ, A is a subtype of B under the output context Θ. Due to the space limitation, we only present when the left hand side (InstL) or the right hand side (InstR) is an existential variable. In those cases, the subtyping leaves all the work to the instantiation judgment ($:\leqq$), some of which we have seen in Section 2. The complete definition can be found in the appendix.

$$\boxed{\Gamma \vdash A \ <: \ B \dashv \Theta}$$

$$\frac{\widehat{\alpha} \notin FV(A) \qquad \Gamma[\widehat{\alpha}] \vdash \widehat{\alpha} :\leqq A \dashv \Theta}{\Gamma[\widehat{\alpha}] \vdash \widehat{\alpha} \ <: \ A \dashv \Theta} \textsc{InstL} \qquad \frac{\widehat{\alpha} \notin FV(A) \qquad \Gamma[\widehat{\alpha}] \vdash A :\leqq \widehat{\alpha} \dashv \Theta}{\Gamma[\widehat{\alpha}] \vdash A \ <: \ \widehat{\alpha} \dashv \Theta} \textsc{InstR}$$

Fig. 10. Algorithmic Subtyping (Original, incomplete).

$$\boxed{\Gamma \vdash A \ <: \ B \dashv \Theta}$$

$$\frac{\widehat{\alpha} \notin FV(A) \qquad \Gamma[\widehat{\alpha}] \vdash^{-} A \longrightarrow \sigma \dashv \Theta, \widehat{\alpha}, \Delta \qquad \Theta \vdash \sigma}{\Gamma[\widehat{\alpha}] \vdash \widehat{\alpha} \ <: \ A \dashv \Theta, \widehat{\alpha} = \sigma, \Delta} \textsc{SaL}$$

$$\frac{\widehat{\alpha} \notin FV(A) \qquad \Gamma[\widehat{\alpha}] \vdash^{+} A \longrightarrow \sigma \dashv \Theta, \widehat{\alpha}, \Delta \qquad \Theta \vdash \sigma}{\Gamma[\widehat{\alpha}] \vdash A \ <: \ \widehat{\alpha} \dashv \Theta, \widehat{\alpha} = \sigma, \Delta} \textsc{SaR}$$

$$\boxed{\Gamma[\widehat{\alpha}] \vdash^{s} A \longrightarrow \sigma \dashv \Theta}$$

$$\frac{\Gamma[\widehat{\beta}, \widehat{\alpha}] \vdash^{+} A[\alpha \mapsto \widehat{\beta}] \longrightarrow \sigma \dashv \Theta}{\Gamma[\widehat{\alpha}] \vdash^{+} \forall \alpha.A \longrightarrow \sigma \dashv \Theta} \textsc{I-All-Plus} \qquad \frac{\Gamma, \alpha \vdash^{-} A \longrightarrow \sigma \dashv \Theta, \alpha}{\Gamma \vdash^{-} \forall \alpha.A \longrightarrow \sigma \dashv \Theta} \textsc{I-All-Minus}$$

$$\frac{\Gamma \vdash^{-s} A_1 \longrightarrow \sigma_1 \dashv \Theta_1 \qquad \Theta_1 \vdash^{s} [\Theta_1](A_2) \longrightarrow \sigma_2 \dashv \Theta}{\Gamma \vdash^{s} A_1 \to A_2 \longrightarrow \sigma_1 \to \sigma_2 \dashv \Theta} \textsc{I-Pi-Poly}$$

$$\frac{}{\Gamma \vdash^{s} 1 \longrightarrow 1 \dashv \Gamma} \textsc{I-Unit} \qquad \frac{}{\Gamma[\widehat{\alpha}][\widehat{\beta}] \vdash^{s} \widehat{\beta} \longrightarrow \widehat{\alpha}_1 \dashv \Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}_1]} \textsc{I-EVarAfter-Poly}$$

$$\frac{}{\Gamma[\alpha] \vdash^{s} \alpha \longrightarrow \alpha \dashv \Gamma[\alpha]} \textsc{I-TVar} \qquad \frac{}{\Gamma[\widehat{\beta}][\widehat{\alpha}] \vdash^{s} \widehat{\beta} \longrightarrow \widehat{\beta} \dashv \Gamma[\widehat{\beta}][\widehat{\alpha}]} \textsc{I-EVarBefore-Poly}$$

Fig. 11. New subtyping rules, polymorphic type sanitization.

4.3 Polymorphic Type Sanitization

The goal of polymorphic type sanitization is to replace the instantiation relationship with even simpler rules.

Therefore, we replace the rules InstL and InstR with new rules SaL and SaR shown at the top of Figure 11. Namely, the subtyping leaves the job to polymorphic type sanitization to resolve the order problem with existential variables and also sanitize the input type to a monotype. A final check ensures that the sanatized type is well-formed under the context before $\widehat{\alpha}$. If the type being sanitized appears on the right (as SaL), we say it appears co-variantly, and we say it appears contra-variantly if it appears on the left (as SaR). This corresponds to the fact that a monotype can be a subtype of a polymorphic type only if all the universal quantifiers that appear in the body are contra-variant in the type.

The rules for polymorphic type sanitization are shown at the bottom of Figure 11. According to whether the type appears co-variantly ($s = -$) or contra-variantly ($s = +$), we have two modes. The judgment $\Gamma[\widehat{\alpha}] \vdash^{s} A \longrightarrow \sigma \dashv \Theta$ is interpreted as: under typing context Γ which contains $\widehat{\alpha}$, sanitize a possibly

polymorphic type A to a monotype σ according to the mode s, with output context Θ. Computationally, there are four inputs ($\Gamma, s, \widehat{\alpha}$ and A), and two outputs (σ and Θ).

The only difference between these two modes is how to sanitize polymorphic types. If a polymorphic type appears contra-variantly (I-All-Plus), it means a monotype would make the final type more polymorphic. Therefore, we replace the universal binder α with a fresh existential variable $\widehat{\beta}$ and put it before $\widehat{\alpha}$. Otherwise, in rule I-All-Minus, we put α in the context and sanitize A. Notice that the result σ might not be well-formed under the output context Θ, since α is discarded in the output context. Rule I-Pi-Poly is where the mode is flipped.

Polymorphic type sanitization does nothing if it is a unit type (I-Unit) or a type variable (I-TVar). Rule I-EVarAfter-Poly and I-EVarBefore-Poly deals with existential variables, and creates fresh existential variables if the input existential variable appears after $\widehat{\alpha}$, as we have seen in type sanitization in Section 3.3.

Example The derivation of the subtyping problem $\widehat{\alpha} \vdash \widehat{\alpha} \mathrel{<:} (\forall\alpha.\alpha \to \alpha) \to 1$ is given below. For clarity, we omit some detailed process, and denote $\Theta = \widehat{\beta}, \widehat{\alpha}$.

$$\dfrac{\dfrac{\dfrac{\Theta \vdash^{+} \widehat{\beta} \to \widehat{\beta} \longrightarrow \widehat{\beta} \to \widehat{\beta} \dashv \Theta}{\widehat{\alpha} \vdash^{+} \forall\alpha.\alpha \to \alpha \longrightarrow \widehat{\beta} \to \widehat{\beta} \dashv \Theta}\,\text{I-All-Plus} \quad \dfrac{}{\Theta \vdash^{-} 1 \longrightarrow 1 \dashv \Theta}\,\text{I-Unit}}{\widehat{\alpha} \vdash^{-} (\forall\alpha.\alpha \to \alpha) \to 1 \longrightarrow (\widehat{\beta} \to \widehat{\beta}) \to 1 \dashv \Theta}\,\text{I-Pi-Poly}}{\widehat{\alpha} \vdash \widehat{\alpha} \mathrel{<:} (\forall\alpha.\alpha \to \alpha) \to 1 \dashv \widehat{\beta}, \widehat{\alpha} = (\widehat{\beta} \to \widehat{\beta}) \to 1}\,\text{SaL}$$

4.4 Meta-theory

The soundness and completeness of subtyping relies on the soundness and completeness of instantiation in Dunfield and Krishnaswami (2013). To prove polymorphic type sanitization works correctly, there is no need to re-prove the lemmas about subtyping. Instead, we prove that polymorphic type sanitization leads to exactly the same result as instantiation.

For soundness, we prove that under contra-variant mode ($s = +$), the input type is a declarative subtype of the output type after substituted by a complete context, while under co-variant mode ($s = -$), the input type is a declarative supertype:

Lemma 5 (Polymorphic Type Sanitization Soundness). *Given $\Theta \longrightarrow \Omega$, and $[\Gamma](A) = A$:*

- *If $\Gamma[\widehat{\alpha}] \vdash^{+} A \longrightarrow \sigma \dashv \Theta$, then $[\Omega](\Theta) \vdash [\Omega](A) \leq [\Omega](\sigma)$.*
- *If $\Gamma[\widehat{\alpha}] \vdash^{-} A \longrightarrow \sigma \dashv \Theta$, then $[\Omega](\Theta) \vdash [\Omega](\sigma) \leq [\Omega](A)$.*

For completeness, we first prove that for a possibly polymorphic type $[\Omega](A)$, if there is a monotype τ that is more polymorphic than it, there is a polymorphic type sanitization result σ of A, and it is the most general subtype in the sense that we can recover the τ from applying a complete context to σ.

Lemma 6 (Polymorphic Type Sanitization Completeness). *Given* $\Gamma \longrightarrow \Omega$, *where* $\Gamma = \Gamma_1, \widehat{\alpha}, \Gamma_2$, *and* $[\Gamma](A) = A$, *and* $[\Omega](\tau) = \tau$, *and* $\widehat{\alpha} \notin FV(A)$, *and* $\Gamma_1 \vdash \tau$:

- *If* $[\Omega](\Gamma) \vdash \tau \leq [\Omega](A)$, *then there are* Δ, Ω' *such that* $\Omega \longrightarrow \Omega'$, *and* $\Delta \longrightarrow \Omega'$, *and* $\Gamma[\widehat{\alpha}] \vdash^{-} A \longrightarrow \sigma \dashv \Delta$, *where* $\Delta = \Delta_1, \widehat{\alpha}, \Delta_2$, *and* $\Delta_1 \vdash \sigma$, *and* $[\Omega'](\sigma) = \tau$.
- *If* $[\Omega](\Gamma) \vdash [\Omega](A) \leq \tau$, *then there are* Δ, Ω' *such that* $\Omega \longrightarrow \Omega'$, *and* $\Delta \longrightarrow \Omega'$, *and* $\Gamma[\widehat{\alpha}] \vdash^{+} A \longrightarrow \sigma \dashv \Delta$, *where* $\Delta = \Delta_1, \widehat{\alpha}, \Delta_2$, *and* $\Delta_1 \vdash \sigma$, *and* $[\Omega'](\sigma) = \tau$.

Based on the completeness of polymorphic type sanitization, we can prove exactly the same completeness lemma as the instantiation to show that subtyping for existential variables is complete:

Corollary 2 (Polymorphic Type Sanitization Completeness w.r.t Subtyping). *Given* $\Gamma \longrightarrow \Omega$, *and* $[\Gamma](A) = A$, *and* $\widehat{\alpha} \notin FV(A)$, *and* $\widehat{\alpha} \in unsolved(\Gamma)$:

- *If* $[\Omega](\Gamma) \vdash [\Omega](\widehat{\alpha}) \leq [\Omega](A)$, *then there are* Δ, Ω' *such that* $\Omega \longrightarrow \Omega'$, *and* $\Delta \longrightarrow \Omega'$, *and* $\Gamma[\widehat{\alpha}] \vdash \widehat{\alpha} <: A \dashv \Delta$.
- *If* $[\Omega](\Gamma) \vdash [\Omega](A) \leq [\Omega](\widehat{\alpha})$, *then there are* Δ, Ω' *such that* $\Omega \longrightarrow \Omega'$, *and* $\Delta \longrightarrow \Omega'$, *and* $\Gamma[\widehat{\alpha}] \vdash A <: \widehat{\alpha} \dashv \Delta$.

5 Discussion

5.1 Type Inference Algorithm For Dependent Types

In Section 3, we use the unification algorithm for dependent types to show that type sanitization is applicable to advanced features. However, notice that the typing rules presented in Section 3.2 work specifically for well-formedness of types. In other words, it is *not* an algorithmic type system containing type inference. Therefore, readers may wonder: how can we design a type inference algorithm for practical usage based on the unification algorithm, and what is the relation between the type inference algorithm and the well-formedness typing rules? In this section, we discuss these two questions.

Unification and Type Inference. In a program, programmers may omit some type annotations, and the type system will try to infer them. A typical example is unannotated lambdas. For example

$\lambda x.\ x + 1$

There is no annotation for the binder x, but the type system is able to recover that $x : Int$ from the expression $x + 1$. This is done by generating a fresh existential variable $\widehat{\alpha}$ for x, and then using the unification algorithm to unify $\widehat{\alpha}$ with Int. In summary, type inference generates existential variables that are waiting to be solved, while unification is in charge of finding the solutions according to the type constraints.

Therefore, we can talk about unification separately from type inference. And type inference for dependent types is also a challenging topic, where the unification algorithm would definitely help with. Of course, in a type system with unannotated lambdas, we also need to extend the unification to deal with the new lambdas, which is not so complicated.

Type Inference and Well-formedness. Even if we are given an algorithmic type system containing type inference, the well-formedness system still works correctly as long as the definition of well-formedness not changed. Namely, any input type to the unification algorithm is already fully type-checked, therefore it will introduce no more new constraints. However, in our unification algorithm, well-formedness is only used for propositions, so we leave it non-deterministic for simplicity. If it is used in the algorithm, we would need to change it to an deterministic and decidable relation.

5.2 Future Work

One possible future work is to apply the strategy of type sanitization to type systems with more advanced features. For example, the polymorphic type sanitization presented in Section 4 works specifically for subtyping between polymorphic types. We can try to extend type sanitization to other kinds of subtyping, for instance, nominal subtyping and/or intersection types (Coppo et al., 1981; Pottinger, 1980).

Also, since type sanitization resolves the unification problem between dependent types, another possible future work is to come up with a complete type inference algorithm for dependent types based on alpha-equality and first-order constraints. Extending the type sanitization to see whether it could play any role in extended setting, such as beta-equality or higher-order constraints is also a feasible future work.

6 Related Work

6.1 Type Inference in Context

We have discussed the work by Gundry et al. (2010) and Dunfield and Krishnaswami (2013) in Section 2, which in some sense can both be understood in terms of the proofs systems of Miller (1992). They adopt different strategies to resolve the order problem of existential variables, with different emphasizes. Gundry et al. (2010) supports ML-style polymorphism, and as they mention the longer-term goal is to elaborate high-level dependently typed programs into fully explicit calculi. However they do not present the algorithm nor prove it. Dunfield and Krishnaswami (2013) use the strategy specifically for higher rank polymorphism, while as we have seen their strategy cannot be applied to dependent types.

6.2 Unification for Dependent Types

Unification for dependent types has been a challenging topic for years. While our unification only solves first-order constraints based on alpha-equality, existing literature takes some advanced features into account. The price to pay for such features is that the algorithm is usually complicated to understand, and also the meta-theory is hard to prove. Elliott (1989) develop a higher order unification algorithm for dependent function types in the spirit of Huet (1975), though the problem is undecidable. Reed (2009) comes up with a constraint simplification algorithm that works on dynamic pattern fragment of higher-order unification in a dependent type system. Later, Abel and Pientka (2011) propose a constraint-based unification algorithm that solves a richer class of patterns. Ziliani and Sozeau (2015) formalize the unification algorithm used in the language Coq (Coq Development Team, 2012), which is quite sophisticated and the correctness proof is lacking.

6.3 Type Inference for Higher Rank Type Systems

Type inference involving higher ranks has been well studied in recent years. Jones et al. (2007) develop an approach using traditional bi-directional type checking, which is built upon Odersky and Läufer (1996). In this system, subtyping and unification are separated, and unification is only between monotypes. Dunfield and Krishnaswami (2013) build a simple and concise algorithm for higher ranked polymorphism, which is also based on traditional bidirectional type checking. The instantiation in their system is introduced in Section 2. These two systems are predicative, in the sense that universal quantifiers can only be instantiated with monotypes. There are also impredictive systems, including ML^F (Le Botlan and Rémy, 2003, 2009; Rémy and Yakobowski, 2008), the HML system (Leijen, 2009) and the FPH system (Vytiniotis et al., 2008).

7 Conclusion

In this paper, we propose a new strategy *type sanitization*, and present two applications of this strategy. The first application is type sanitization in a unification algorithm for a dependent type system with alpha-equality based first order constrains. And the second one is in a subtyping algorithm for a higher rank polymorphic type system with the extended *polymorphic type sanitization*.

We expect type sanitization is applicable to more type systems with other advanced features, for example, intersection types and more advanced forms of dependent types.

Bibliography

Andreas Abel and Brigitte Pientka. Higher-order dynamic pattern unification for dependent types and records. In *International Conference on Typed Lambda Calculi and Applications*, pages 10–26. Springer, 2011.

Edwin Brady. Idris, a general-purpose dependently typed programming language: Design and implementation. *Journal of Functional Programming*, 23 (05):552–593, 2013.

Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Functional characters of solvable terms. *Mathematical Logic Quarterly*, 27(2-6):45–58, 1981.

Coq Development Team. The *Coq* proof assistant reference manual, 2012.

Luis Damas and Robin Milner. Principal type-schemes for functional programs. In *Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '82, pages 207–212. ACM, 1982. ISBN 0-89791-065-6.

Joshua Dunfield. Greedy bidirectional polymorphism. In *Proceedings of the 2009 ACM SIGPLAN workshop on ML*, pages 15–26. ACM, 2009.

Joshua Dunfield and Neelakantan R. Krishnaswami. Complete and easy bidirectional typechecking for higher-rank polymorphism. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, ICFP '13, pages 429–442. ACM, 2013. ISBN 978-1-4503-2326-0.

Joshua Dunfield and Neelakantan R. Krishnaswami. Sound and complete bidirectional typechecking for higher-rank polymorphism with existentials and indexed types. `http://arxiv.org/abs/1601.05106`, January 2016.

Conal Elliott. Higher-order unification with dependent function types. In *Rewriting Techniques and Applications*, pages 121–136. Springer, 1989.

Warren D Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13(2):225–230, 1981.

Adam Gundry, Conor McBride, and James McKinna. Type inference in context. In *Proceedings of the third ACM SIGPLAN workshop on Mathematically structured functional programming*, pages 43–54. ACM, 2010.

J. Roger Hindley. The principal type-scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60, 1969.

Gerard P. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1(1):27–57, 1975.

Simon Peyton Jones, Dimitrios Vytiniotis, Stephanie Weirich, and Mark Shields. Practical type inference for arbitrary-rank types. *Journal of functional programming*, 17(01):1–82, 2007.

Garrin Kimmell, Aaron Stump, Harley D Eades III, Peng Fu, Tim Sheard, Stephanie Weirich, Chris Casinghino, Vilhelm Sjöberg, Nathan Collins, and Ki Yung Ahn. Equational reasoning about programs with general recursion and call-by-value semantics. In *Proceedings of the sixth workshop on Programming languages meets program verification*, pages 15–26. ACM, 2012.

Didier Le Botlan and Didier Rémy. Mlf: Raising ml to the power of system f. In *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming*, ICFP '03, pages 27–38. ACM, 2003. ISBN 1-58113-756-7.

Didier Le Botlan and Didier Rémy. Recasting mlf. *Information and Computation*, 207(6):726–785, 2009.

Daan Leijen. Flexible types: Robust type inference for first-class polymorphism. In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '09, pages 66–77. ACM, 2009. ISBN 978-1-60558-379-2.

Daniel R Licata and Robert Harper. A formulation of dependent ml with explicit equality proofs. 2005.

James McKinna. Why dependent types matter. In *Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '06, pages 1–1. ACM, 2006. ISBN 1-59593-027-2.

Dale Miller. Unification under a mixed prefix. *Journal of symbolic computation*, 14(4):321–358, 1992.

Ulf Norell. Dependently typed programming in agda. In *Advanced Functional Programming*, pages 230–266. Springer, 2009.

Martin Odersky and Konstantin Läufer. Putting type annotations to work. In *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '96, pages 54–67. ACM, 1996. ISBN 0-89791-769-3.

Emir Pasalic, Jeremy Siek, and Walid Taha. Concoqtion: Mixing dependent types and hindley-milner type inference. 2006.

Garrel Pottinger. A type assignment for the strongly normalizable λ-terms. *To HB Curry: essays on combinatory logic, lambda calculus and formalism*, pages 561–577, 1980.

Jason Reed. Higher-order constraint simplification in dependent type theory. In *Proceedings of the Fourth International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice*, pages 49–56. ACM, 2009.

Didier Rémy and Boris Yakobowski. From ml to mlf: Graphic type constraints with efficient type inference. In *Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming*, ICFP '08, pages 63–74. ACM, 2008. ISBN 978-1-59593-919-7.

Vilhelm Sjöberg and Stephanie Weirich. Programming up to congruence. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 369–382. ACM, 2015. ISBN 978-1-4503-3300-9.

Vilhelm Sjöberg, Chris Casinghino, Ki Yung Ahn, Nathan Collins, Harley D Eades III, Peng Fu, Garrin Kimmell, Tim Sheard, Aaron Stump, and Stephanie Weirich. Irrelevance, heterogeneous equality, and call-by-value dependent type systems. *arXiv preprint arXiv:1202.2923*, 2012.

Aaron Stump, Morgan Deters, Adam Petcher, Todd Schiller, and Timothy Simpson. Verified programming in guru. In *Proceedings of the 3rd workshop on Programming languages meets program verification*, pages 49–58. ACM, 2009.

Martin Sulzmann, Manuel MT Chakravarty, Simon Peyton Jones, and Kevin Donnelly. System f with type equality coercions. In *Proceedings of the 2007 ACM SIGPLAN international workshop on Types in languages design and implementation*, pages 53–66. ACM, 2007.

Floris van Doorn, Herman Geuvers, and Freek Wiedijk. Explicit convertibility proofs in pure type systems. In *Proceedings of the Eighth ACM SIGPLAN international workshop on Logical frameworks & meta-languages: theory & practice*, pages 25–36. ACM, 2013.

Dimitrios Vytiniotis, Stephanie Weirich, and Simon Peyton Jones. Fph: First-class polymorphism for haskell. In *Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming*, ICFP '08, pages 295–306. ACM, 2008. ISBN 978-1-59593-919-7.

Hongwei Xi and Frank Pfenning. Dependent types in practical programming. In *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 214–227. ACM, 1999.

Yanpeng Yang, Xuan Bi, and Bruno C d S Oliveira. Unified syntax with iso-types. In *Asian Symposium on Programming Languages and Systems*, pages 251–270. Springer, 2016.

Beta Ziliani and Matthieu Sozeau. A unification algorithm for coq featuring universe polymorphism and overloading. In *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming*, ICFP 2015, pages 179–191. ACM, 2015. ISBN 978-1-4503-3669-7.

A Appendix Overview

In this section, we give an overview of the all the appendixes.

A.1 Appendix B

Due to the space limitation, in the paper we only give some incomplete definitions. In this appendix, we present the full version of them.

A.2 Appendix C

This section gives the proofs for the dependent type system. Here we list some important lemmas.

Type Sanitization

- Lemma 42. Type Sanitization Extension.
  If $\Gamma \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta$, and $\Gamma \vdash \tau_1 \Rightarrow \sigma$, then $\Gamma \longrightarrow \Theta$.
- Lemma 43. Type Sanitization Equivalence.
  If $\Gamma \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta$, and $\Gamma \vdash \tau_1 \Rightarrow \sigma$, then $[\Theta](\tau_1) = [\Theta](\tau_2)$.
- Lemma 44. Type Sanitization Well Formedness.
  If $\Gamma \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta$, and $\Gamma \vdash \tau_1 \Rightarrow \sigma$, then $\Theta \vdash \tau_2 \Rightarrow [\Theta](\sigma)$.
- Lemma 46. Type Sanitization Completeness.
  Given $\Gamma \longrightarrow \Omega$, where $\Gamma = \Gamma_1, \widehat{\alpha}, \Gamma_2, \Gamma_0$, and Γ_0 only contains variables, and $[\Gamma](\sigma_1) = \sigma_1$, and $[\Omega](\tau) = \tau$, and $\widehat{\alpha} \notin FV(\sigma_1)$, and $\Gamma_1, \Gamma_0 \vDash \tau$. If $[\Omega](\Gamma) \vdash \tau = [\Omega](\sigma_1)$, then there are Δ, Ω' such that $\Omega \longrightarrow \Omega'$, and $\Delta \longrightarrow \Omega'$, and $\Gamma[\widehat{\alpha}] \vdash \sigma_1 \longrightarrow \sigma_2 \dashv \Delta$, where $\Delta = \Delta_1, \widehat{\alpha}, \Delta_2, \Gamma_0$, and $\Delta_1, \Gamma_0 \vDash \sigma_2$.
- Corollary 4. Type Sanitization Completeness Pretty.
  Given $\Gamma \longrightarrow \Omega$, where $\Gamma = \Gamma_1, \widehat{\alpha}, \Gamma_2$, and $[\Gamma](\sigma_1) = \sigma_1$, and $[\Omega](\tau) = \tau$, and $\widehat{\alpha} \notin FV(\sigma_1)$, and $\Gamma_1 \vDash \tau$. If $[\Omega](\Gamma) \vdash \tau = [\Omega](\sigma_1)$, then there are Δ, Ω' such that $\Omega \longrightarrow \Omega'$, and $\Delta \longrightarrow \Omega'$, and $\Gamma[\widehat{\alpha}] \vdash \sigma_1 \longrightarrow \sigma_2 \dashv \Delta$, where $\Delta = \Delta_1, \widehat{\alpha}, \Delta_2$, and $\Delta_1 \vDash \sigma_2$.
- Lemma 47. Type Sanitization Completeness w.r.t. Unification.
  Given $\Gamma \longrightarrow \Omega$, where $\Gamma = \Gamma_1, \widehat{\alpha}, \Gamma_2$, and $[\Gamma](\sigma_1) = \sigma_1$, and $\widehat{\alpha} \notin FV(\sigma_1)$.
  - If $[\Omega](\Gamma) \vdash [\Omega](\widehat{\alpha}) = [\Omega](\sigma_1)$, then there are Δ, Ω' such that $\Omega \longrightarrow \Omega'$, and $\Delta \longrightarrow \Omega'$, and $\Gamma[\widehat{\alpha}] \vdash \widehat{\alpha} \simeq \sigma_1 \dashv \Delta$.
  - If $[\Omega](\Gamma) \vdash [\Omega](\sigma_1) = [\Omega](\widehat{\alpha})$, then there are Δ, Ω' such that $\Omega \longrightarrow \Omega'$, and $\Delta \longrightarrow \Omega'$, and $\Gamma[\widehat{\alpha}] \vdash \sigma_1 \simeq \widehat{\alpha} \dashv \Delta$.

Unification

- Lemma 48. Unification Extension.

  - If $\Gamma \vdash_e e_1 \simeq e_2 \dashv \Theta$, and $\Gamma \vdash e_1 \Rightarrow \sigma_1'$, and $\Gamma \vdash e_2 \Rightarrow \sigma_2'$, then $\Gamma \longrightarrow \Theta$.

- If $\Gamma \vdash_\sigma \tau_1 \simeq \tau_2 \dashv \Theta$, and $\Gamma \vdash \tau_1 \Rightarrow \star$, and $\Gamma \vdash \tau_2 \Rightarrow \star$, then $\Gamma \longrightarrow \Theta$.

– Lemma 49. Unification Equivalence.
If $\Gamma \vdash_\delta \tau_1 \simeq \tau_2 \dashv \Theta$, and $\Gamma \vdash \tau_1 \Rightarrow \sigma_1'$, and $\Gamma \vdash \tau_2 \Rightarrow \sigma_2'$, then $[\Theta](\tau_1) = [\Theta](\tau_2)$.
– Lemma 50. Unification Completeness.
Given $\Gamma \longrightarrow \Omega$, and $[\Gamma](\sigma_1) = \sigma_1$, and $[\Gamma](\sigma_2) = \sigma_2$. If $[\Omega](\Gamma) \vdash [\Omega](\sigma_1) = [\Omega](\sigma_2) \Rightarrow \tau$, then there are Δ, Ω' such that $\Omega \longrightarrow \Omega'$, and $\Delta \longrightarrow \Omega'$, and $\Gamma \vdash_\delta \sigma_1 \simeq \sigma_2 \dashv \Delta$ for any δ suitable.

A.3 Appendix C

This section gives the proofs for the higher rank type system. Here we list some important lemmas.

Polymorphic Type Sanitization

– Lemma 69. Polymorphic Type Sanitization Extension.
If $\Gamma \vdash^s A \longrightarrow \sigma \dashv \Theta$, then $\Gamma \longrightarrow \Theta$.
– Lemma 70. Polymorphic Type Sanitization Soundness.
Given $\Theta \longrightarrow \Omega$, and $[\Gamma](A) = A$:
- If $\Gamma[\widehat{\alpha}] \vdash^+ A \longrightarrow \sigma \dashv \Theta$, then $[\Omega](\Theta) \vdash [\Omega](A) \leq [\Omega](\sigma)$.
- If $\Gamma[\widehat{\alpha}] \vdash^- A \longrightarrow \sigma \dashv \Theta$, then $[\Omega](\Theta) \vdash [\Omega](\sigma) \leq [\Omega](A)$.

– Lemma 71. Polymorphic Type Sanitization Completeness.
Given $\Gamma \longrightarrow \Omega$, where $\Gamma = \Gamma_1, \widehat{\alpha}, \Gamma_2$, and $[\Gamma](A) = A$, and $[\Omega](\tau) = \tau$, and $\widehat{\alpha} \notin FV(A)$, and $\Gamma_1 \vdash \tau$:
- If $[\Omega](\Gamma) \vdash \tau \leq [\Omega](A)$, then there are Δ, Ω' such that $\Omega \longrightarrow \Omega'$, and $\Delta \longrightarrow \Omega'$, and $\Gamma[\widehat{\alpha}] \vdash^- A \longrightarrow \sigma \dashv \Delta$, where $\Delta = \Delta_1, \widehat{\alpha}, \Delta_2$, and $\Delta_1 \vdash \sigma$, and $[\Omega'](\sigma) = \tau$.
- If $[\Omega](\Gamma) \vdash [\Omega](A) \leq \tau$, then there are Δ, Ω' such that $\Omega \longrightarrow \Omega'$, and $\Delta \longrightarrow \Omega'$, and $\Gamma[\widehat{\alpha}] \vdash^+ A \longrightarrow \sigma \dashv \Delta$, where $\Delta = \Delta_1, \widehat{\alpha}, \Delta_2$, and $\Delta_1 \vdash \sigma$, and $[\Omega'](\sigma) = \tau$.

– Corollary 5. Polymorphic Type Sanitization Completeness w.r.t Subtyping.
Given $\Gamma \longrightarrow \Omega$, and $[\Gamma](A) = A$, and $\widehat{\alpha} \notin FV(A)$, and $\widehat{\alpha} \in unsolved(\Gamma)$:
- If $[\Omega](\Gamma) \vdash [\Omega](\widehat{\alpha}) \leq [\Omega](A)$, then there are Δ, Ω' such that $\Omega \longrightarrow \Omega'$, and $\Delta \longrightarrow \Omega'$, and $\Gamma[\widehat{\alpha}] \vdash \widehat{\alpha} <: A \dashv \Delta$.
- If $[\Omega](\Gamma) \vdash [\Omega](A) \leq [\Omega](\widehat{\alpha})$, then there are Δ, Ω' such that $\Omega \longrightarrow \Omega'$, and $\Delta \longrightarrow \Omega'$, and $\Gamma[\widehat{\alpha}] \vdash A <: \widehat{\alpha} \dashv \Delta$.

B Complete Definitions

In this section, we give the full definitions from Dunfield and Krishnaswami (2013) that are omitted in the paper.

$$\boxed{\Gamma \vdash A \ <: \ B \dashv \Theta}$$

$$\frac{}{\Gamma[\alpha] \vdash \alpha \ <: \ \alpha \dashv \Gamma[\alpha]}\ \textsc{Var} \qquad \frac{}{\Gamma \vdash 1 \ <: \ 1 \dashv \Gamma}\ \textsc{Unit} \qquad \frac{}{\Gamma[\widehat{\alpha}] \vdash \widehat{\alpha} \ <: \ \widehat{\alpha} \dashv \Gamma[\widehat{\alpha}]}\ \textsc{Exvar}$$

$$\frac{\Gamma \vdash B_1 \ <: \ A_1 \dashv \Theta \qquad \Theta \vdash [\Theta](A_2) \ <: \ [\Theta](B_2) \dashv \Delta}{\Gamma \vdash A_1 \to A_2 \ <: \ B_1 \to B_2 \dashv \Delta} \to$$

$$\frac{\Gamma, \blacktriangleright_{\widehat{\alpha}}, \widehat{\alpha} \vdash A[\widehat{\alpha} \mapsto \alpha] \ <: \ B \dashv \Delta, \blacktriangleright_{\widehat{\alpha}}, \Theta}{\Gamma \vdash \forall \alpha.A \ <: \ B \dashv \Delta}\ \forall\text{L} \qquad \frac{\Gamma, \alpha \vdash A \ <: \ B \dashv \Delta, \alpha, \Theta}{\Gamma \vdash A \ <: \ \forall \alpha.B \dashv \Delta}\ \forall\text{R}$$

$$\frac{\widehat{\alpha} \notin FV(A) \qquad \Gamma[\widehat{\alpha}] \vdash \widehat{\alpha} :\leqq A \dashv \Theta}{\Gamma[\widehat{\alpha}] \vdash \widehat{\alpha} \ <: \ A \dashv \Theta}\ \textsc{InstL} \qquad \frac{\widehat{\alpha} \notin FV(A) \qquad \Gamma[\widehat{\alpha}] \vdash A :\leqq \widehat{\alpha} \dashv \Theta}{\Gamma[\widehat{\alpha}] \vdash A \ <: \ \widehat{\alpha} \dashv \Theta}\ \textsc{InstR}$$

Fig. 12. Algorithmic Subtyping (Original).

Subtyping The original definition of algorithmic subtyping is given in Figure 12 (Dunfield and Krishnaswami, 2013). This is the complete definition for Figure 10. The judgment $\Gamma \vdash A \ <: \ B \dashv \Theta$ is interpreted as: given the context Γ, A is a subtype of B under the output context Θ. The subtyping relation is reflexive (Var, Exvar and Unit). The function type is contra-variant on the argument type, and co-variant on the return type ($\to$). Here the intermediate context Θ is applied to A_2 and B_2 to make sure the input types are fully substituted under the input context. In $\forall$L, new existential variable $\widehat{\alpha}$ is created to represent the universal quantifier. There is a marker before $\widehat{\alpha}$ so that we can throw away all tailing contexts that are out of scope after the subtyping. Similarly, rule $\forall$R puts a new type variable in the context and throws away all the contexts after the type variable.

When the left hand side (InstL) or the right hand side (InstR) is an existential variable, the subtyping leaves all the work to the instantiation judgment ($:\leqq$), which is given at the bottom of Figure 13.

Instantiation The original definition of instantiation is given in Figure 13 (Dunfield and Krishnaswami, 2013). This is the complete definition for Figure 2. There are two judgments corresponding to whether the existential variable is on the left or right.

The judgment $\Gamma \vdash \widehat{\alpha} :\leqq A \dashv \Theta$ is interpreted as: given the context Γ, instantiate $\widehat{\alpha}$ so that it is a subtype of A. If A is already a monotype and it is well-formed under context Γ_1, then we directly solve $\widehat{\alpha} = \tau$ (InstLSolve). If A is another existential variable that appears after $\widehat{\alpha}$, then we solve $\widehat{\beta} = \widehat{\alpha}$ (InstLReach). If A is a function type, it means $\widehat{\alpha}$ must be a function. Therefore we generate two fresh existential variables $\widehat{\alpha}_1, \widehat{\alpha}_2$, and continue to instantiate A_1 with $\widehat{\alpha}_1$, and $\widehat{\alpha}_2$ with $[\Theta](A_2)$ (InstLArr). If A is a polymorphic type, and we put the type variable β into the context and continue to instantiate $\widehat{\alpha}$ with B

$$\boxed{\Gamma \vdash \widehat{\alpha} :\leqq A \dashv \Theta}$$

$$\frac{\Gamma_1 \vdash \tau}{\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash \widehat{\alpha} :\leqq \tau \dashv \Gamma_1, \widehat{\alpha} = \tau, \Gamma_2} \textsc{InstLSolve}$$

$$\frac{}{\Gamma[\widehat{\alpha}][\widehat{\beta}] \vdash \widehat{\alpha} :\leqq \widehat{\beta} \dashv \Gamma[\widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}]} \textsc{InstLReach}$$

$$\frac{\Gamma[\widehat{\alpha}_2, \widehat{\alpha}_1, \widehat{\alpha} = \widehat{\alpha}_1 \to \widehat{\alpha}_2] \vdash A_1 :\leqq \widehat{\alpha}_1 \dashv \Theta \qquad \Theta \vdash \widehat{\alpha}_2 :\leqq [\Theta](A_2) \dashv \Delta}{\Gamma[\widehat{\alpha}] \vdash \widehat{\alpha} :\leqq A_1 \to A_2 \dashv \Delta} \textsc{InstLArr}$$

$$\frac{\Theta[\widehat{\alpha}], \beta \vdash \widehat{\alpha} :\leqq B \dashv \Theta, \beta, \Delta}{\Gamma[\widehat{\alpha}] \vdash \widehat{\alpha} :\leqq \forall \beta.B \dashv \Theta} \textsc{InstLAllR}$$

$$\boxed{\Gamma \vdash A :\leqq \widehat{\alpha} \dashv \Theta}$$

$$\frac{\Gamma_1 \vdash \tau}{\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash \tau :\leqq \widehat{\alpha} \dashv \Gamma_1, \widehat{\alpha} = \tau, \Gamma_2} \textsc{InstRSolve}$$

$$\frac{}{\Gamma[\widehat{\alpha}][\widehat{\beta}] \vdash \widehat{\beta} :\leqq \widehat{\alpha} \dashv \Gamma[\widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}]} \textsc{InstRReach}$$

$$\frac{\Gamma[\widehat{\alpha}_2, \widehat{\alpha}_1, \widehat{\alpha} = \widehat{\alpha}_1 \to \widehat{\alpha}_2] \vdash \widehat{\alpha}_1 :\leqq A_1 \dashv \Theta \qquad \Theta \vdash [\Theta](A_2) :\leqq \widehat{\alpha}_2 \dashv \Delta}{\Gamma[\widehat{\alpha}] \vdash A_1 \to A_2 :\leqq \widehat{\alpha} \dashv \Delta} \textsc{InstLArr}$$

$$\frac{\Theta[\widehat{\alpha}], \blacktriangleright_{\widehat{\beta}}, \widehat{\beta} \vdash B[\beta \mapsto \widehat{\beta}] :\leqq \widehat{\alpha} \dashv \Theta, \blacktriangleright_{\widehat{\beta}}, \Delta}{\Gamma[\widehat{\alpha}] \vdash \forall \beta.B :\leqq \widehat{\alpha} \dashv \Theta} \textsc{InstRAllL}$$

Fig. 13. Instantiation (Original).

(InstLAllR). Notice that since β is not in the scope of $\widehat{\alpha}$ the context, if β appears in B, then the instantiation will fail.

The other judgment $\Gamma \vdash A :\leqq \widehat{\alpha} \dashv \Theta$ plays a symmetric role. The only difference is in InstRAllL, where we put a fresh existential variable $\widehat{\beta}$ in the context. For example $\widehat{\alpha} \vdash \forall \beta.\beta \to \beta :\leqq \widehat{\alpha}$ has solution $\widehat{\beta} \to \widehat{\beta}$.

Context Extension We give the definition of context extension in Dunfield and Krishnaswami (2013) in Figure 14. Our definition in Figure 8 mimics the original definition. Therefore we save the explanation here.

$$\boxed{\Gamma \longrightarrow \Delta}$$

$$\frac{}{\varnothing \longrightarrow \varnothing}\ \textsc{Empty} \qquad \frac{\Gamma \longrightarrow \Delta \qquad [\Delta](A) = [\Delta](A')}{\Gamma, x : A \longrightarrow \Delta, x : A'}\ \textsc{Var} \qquad \frac{\Gamma \longrightarrow \Delta}{\Gamma, \alpha \longrightarrow \Delta, \alpha}\ \textsc{TVar}$$

$$\frac{\Gamma \longrightarrow \Delta}{\Gamma, \widehat{\alpha} \longrightarrow \Delta, \widehat{\alpha}}\ \textsc{EVar} \qquad \frac{\Gamma \longrightarrow \Delta \qquad [\Delta](\tau_1) = [\Delta](\tau_2)}{\Gamma, \widehat{\alpha} = \tau_1 \longrightarrow \Delta, \widehat{\alpha} = \tau_2}\ \textsc{SolvedEVar}$$

$$\frac{\Gamma \longrightarrow \Delta}{\Gamma, \widehat{\alpha} \longrightarrow \Delta, \widehat{\alpha} = \tau}\ \textsc{CE-Solve} \qquad \frac{\Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta, \widehat{\alpha}}\ \textsc{CE-Add} \qquad \frac{\Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta, \widehat{\alpha} = \tau}\ \textsc{CE-AddSolved}$$

$$\frac{\Gamma \longrightarrow \Delta}{\Gamma, \blacktriangleright_{\widehat{\alpha}} \longrightarrow \Delta, \blacktriangleright_{\widehat{\alpha}}}\ \textsc{CE-Marker}$$

Fig. 14. Context Extension.

C Dependent Type System

C.1 Properties of Context Application

Lemma 7 (Context Application is Idempotent). *If* Γ *ctx , then* $[\Gamma]([\Gamma](\sigma)) = [\Gamma](\sigma)$.

Proof. By induction on the well formedness of context. Case AC-Empty, AC-Var, and AC-EVar are trivial. We discuss AC-SolvedEVar below.

– Case
$$\frac{\Gamma\ ctx \qquad \widehat{\alpha} \notin \Gamma \qquad \Gamma \vdash \tau\ \Rightarrow\ \star}{\Gamma, \widehat{\alpha} = \tau\ ctx}\ \textsc{AC-SolvedEVar}$$

$[\Gamma, \widehat{\alpha} = \tau]([\Gamma, \widehat{\alpha} = \tau](\sigma)) = [\Gamma, \widehat{\alpha} = \tau]([\Gamma](\sigma[\widehat{\alpha} \mapsto \tau]))$	By definition
$\Gamma \vdash \tau\ \Rightarrow\ \star$	Given
$\widehat{\alpha} \notin \Gamma$	Given
$\widehat{\alpha} \notin FV(\tau)$	By above propositions
$\widehat{\alpha} \notin FV(\sigma[\widehat{\alpha} \mapsto \tau])$	Follows directly
$\widehat{\alpha} \notin FV([\Gamma](\sigma[\widehat{\alpha} \mapsto \tau]))$	Follows directly
$[\Gamma, \widehat{\alpha} = \tau]([\Gamma](\sigma[\widehat{\alpha} \mapsto \tau])) = [\Gamma]([\Gamma](\sigma[\widehat{\alpha} \mapsto \tau]))$	Substitute fresh variable
$= [\Gamma](\sigma[\widehat{\alpha} \mapsto \tau])$	By induction hypothesis
$= [\Gamma, \widehat{\alpha} = \tau](\sigma)$	By definition of context application

□

Lemma 8 (Reduction Preserves Fully Substitution). *If* Γ *ctx , and* $[\Gamma](\sigma) = \sigma$*, and* $\sigma \hookrightarrow \tau$*, then* $[\Gamma](\tau) = \tau$.

Proof. Follows directly from the definition of context substitution and reduction. □

Lemma 9 (Context Application Over Reduction). *If* $\sigma_1 \hookrightarrow \sigma_2$*, and* $[\Gamma](\sigma_1) \hookrightarrow [\Gamma](\sigma_2)$.

Proof. By induction on the reduction derivation.

– Case
$$\frac{e_1 \hookrightarrow e_1'}{e_1\ e_2 \hookrightarrow e_1'\ e_2}\ \textsc{R-App}$$

– Case

$$\frac{}{(\lambda x : \sigma.\ e_1)\ e_2 \hookrightarrow e_1[x \mapsto e_2]}\ \textsc{R-Beta}$$

$[\Gamma]((\lambda x : \sigma.\ e_1)\ e_2)$
$= (\lambda x : [\Gamma](\sigma).\ [\Gamma](e_1))\ [\Gamma](e_2)$ By definition of substitution
$\hookrightarrow ([\Gamma](e_1))[x \mapsto [\Gamma](e_2)]$ By R-Beta
$= [\Gamma](e_1[x \mapsto e_2])$ By property of substitution

– Case

$$\frac{e \hookrightarrow e'}{\mathsf{cast}_{\downarrow}\ e \hookrightarrow \mathsf{cast}_{\downarrow}\ e'}\ \textsc{R-CastDown}$$

Follows directly from induction hypothesis.

– Case

$$\frac{}{\mathsf{cast}_{\downarrow}\ (\mathsf{cast}_{\uparrow}\ e) \hookrightarrow e}\ \textsc{R-CastDownUp}$$

Follows directly from induction hypothesis.

□

Lemma 10 (Output is Fully Substituted). *If* $\Gamma \vdash \sigma_1 \Rightarrow \sigma_2$, *then* $[\Gamma](\sigma_2) = \sigma_2$.

Proof. By induction on typing derivation.

– Case A-Ax, A-EVar, A-SolvedEVar, A-Pi follows directly from $[\Gamma](\star) = \star$.

– Case

$$\frac{\Gamma\ ctx \qquad x : \sigma \in \Gamma}{\Gamma \vdash x \Rightarrow [\Gamma](\sigma)}\ \textsc{A-Var}$$

$[\Gamma]([\Gamma](\sigma)) = [\Gamma](\sigma)$ By Lemma 7

– Case

$$\frac{\Gamma \vdash \sigma_1 \Rightarrow \star \qquad \Gamma, x : \sigma_1 \vdash e \Rightarrow \sigma_2}{\Gamma \vdash \lambda x : \sigma_1.\ e \Rightarrow \Pi x : [\Gamma](\sigma_1).\ \sigma_2}\ \textsc{A-LamAnn}$$

$[\Gamma](\Pi x : [\Gamma](\sigma_1).\ \sigma_2) = \Pi x : [\Gamma]([\Gamma](\sigma_1)).\ [\Gamma](\sigma_2)$ Follows from definition of context substitution
$= \Pi x : [\Gamma](\sigma_1).\ [\Gamma](\sigma_2)$ By Lemma 7
$[\Gamma, x : \sigma_1](\sigma_2) = \sigma_2$ By hypothesis
$[\Gamma](\sigma_2) = \sigma_2$ Follows from definition of context substitution
$[\Gamma](\Pi x : [\Gamma](\sigma_1).\ \sigma_2) = \Pi x : [\Gamma](\sigma_1).\ \sigma_2$ By above equalities

– Case

$$\frac{\Gamma \vdash e_1 \;\Rightarrow\; \Pi x : \sigma_1.\ \sigma_2 \qquad \Gamma \vdash e_2 \;\Rightarrow\; \sigma_1}{\Gamma \vdash e_1\ e_2 \;\Rightarrow\; \sigma_2[x \mapsto [\Gamma](e_1)]}\ \text{A-App}$$

$[\Gamma](\Pi x : \sigma_1.\ \sigma_2) = \Pi x : \sigma_1.\ \sigma_2$	By hypothesis
$[\Gamma](\sigma_2) = \sigma_2$	Follows directly from above
$[\Gamma]([\Gamma](e)) = [\Gamma](e)$	By Lemma 7
$[\Gamma](\sigma_2[x \mapsto [\Gamma](e_1)])$	
$= [\Gamma](\sigma_2)[x \mapsto [\Gamma]([\Gamma](e_1))]$	Context application is distributed over substitution
$= \sigma_2[x \mapsto [\Gamma](e_1)]$	By above equalities

– Case

$$\frac{\Gamma \vdash e \;\Rightarrow\; \sigma_1 \qquad \sigma_1 \hookrightarrow \sigma_2}{\Gamma \vdash \mathsf{cast}_{\downarrow}\ e \;\Rightarrow\; \sigma_2}\ \text{A-CastDn}$$

$[\Gamma](\sigma_1) = \sigma_1$ By hypothesis
$[\Gamma](\sigma_2) = \sigma_2$ By Lemma 8

– Case

$$\frac{\Gamma \vdash e \;\Rightarrow\; \sigma_2 \qquad [\Gamma](\sigma_1) \hookrightarrow \sigma_2}{\Gamma \vdash \mathsf{cast}_{\uparrow}\ e \;\Rightarrow\; [\Gamma](\sigma_1)}\ \text{A-CastUp}$$

Follows directly from Lemma 7

□

Lemma 11 (Context Application In Context). *If* $\Gamma_1, y : \tau, \Gamma_2 \vdash \sigma_1 \;\Rightarrow\; \sigma_2$, *and* $\Gamma_1, y : [\Gamma_1](\tau), \Gamma_2\ ctx$, *then* $\Gamma_1, y : [\Gamma_1](\tau), \Gamma_2 \vdash \sigma_1 \;\Rightarrow\; \sigma_2$.

Proof. By induction on the typing relation.

– Case A-AX, A-EVar and A-SolvedEVar follows directly.
– Case

$$\frac{\Gamma\ ctx \qquad x : \sigma \in \Gamma}{\Gamma \vdash x \;\Rightarrow\; [\Gamma](\sigma)}\ \text{A-Var}$$

If $x = y$, then result type is $[\Gamma]([\Gamma](\tau)) = [\Gamma](\tau)$ by Lemma 7. Otherwise the result is the same.
– The rest cases follows directly from the hypothesis.

□

Lemma 12 (Reverse Context Application In Context). *If* $\Gamma_1, y : [\Gamma_1](\tau), \Gamma_2 \vdash \sigma_1 \;\Rightarrow\; \sigma_2$, *and* $\Gamma_1, y : \tau, \Gamma_2\ ctx$, *then* $\Gamma_1, y : \tau, \Gamma_2 \vdash \sigma_1 \;\Rightarrow\; \sigma_2$.

Proof. By induction on the typing relation.

– Case A-AX, A-EVar and A-SolvedEVar follows directly.

– Case

$$\frac{\Gamma \ ctx \qquad x : \sigma \in \Gamma}{\Gamma \vdash x \ \Rightarrow \ [\Gamma](\sigma)} \text{ A-Var}$$

If $x = y$, then result type is $[\Gamma](\tau) = [\Gamma]([\Gamma](\tau))$ by Lemma 7. Otherwise the result is the same.

– The rest cases follows directly from the hypothesis.

□

Definition 1 (Typing Size). , *The size of typing derivation* $\Gamma \vdash \sigma \ \Rightarrow \ \tau$, *defined based on typing process in Figure* **??**, *written as* $|\Gamma \vdash \sigma \ \Rightarrow \ \tau|$, *is defined as:*

$$\begin{aligned}
|\Gamma \vdash \star \ \Rightarrow \ \star| &= 1 \\
|\Gamma \vdash x \ \Rightarrow \ [\Gamma](\sigma)| &= 1 \\
|\Gamma \vdash \widehat{\alpha} \ \Rightarrow \ \star| &= 1 \text{ if } \widehat{\alpha} \in \Gamma \\
|\Gamma \vdash \widehat{\alpha} \ \Rightarrow \ \star| &= 1 + |\Gamma_1 \vdash \tau \ \Rightarrow \ \star| \text{ if } \Gamma = \Gamma_1, \widehat{\alpha} = \tau, \Gamma 2 \\
|\Gamma \vdash \lambda x : \sigma_1.\ e \ \Rightarrow \ \Pi x : [\Gamma](\sigma_1).\ \sigma_2| &= 1 + |\Gamma \vdash \sigma_1 \ \Rightarrow \ \star| + |\Gamma, x : \sigma_1 \vdash e \ \Rightarrow \ \sigma_2| \\
|\Gamma \vdash \Pi x : \sigma_1.\ \sigma_2 \ \Rightarrow \ \star| &= 1 + |\Gamma \vdash \sigma_1 \ \Rightarrow \ \star| + |\Gamma, x : \sigma_1 \vdash \sigma_2 \ \Rightarrow \ \star| \\
|\Gamma \vdash \mathsf{cast}_{\downarrow} \ e \ \Rightarrow \ \sigma_2| &= 1 + |\Gamma \vdash e \ \Rightarrow \ \sigma_1| \\
|\Gamma \vdash \mathsf{cast}_{\uparrow} \ e \ \Rightarrow \ [\Gamma](\sigma_1)| &= 1 + |\Gamma \vdash e \ \Rightarrow \ \sigma_2|
\end{aligned}$$

Lemma 13 (Context Application Preserves Typing). *If* $\Gamma \vdash \sigma_1 \ \Rightarrow \ \sigma_2$, *then* $\Gamma \vdash [\Gamma](\sigma_1) \ \Rightarrow \ \sigma_2$.

Proof. By induction on the typing size.

– Case

$$\frac{\Gamma \ ctx}{\Gamma \vdash \star \ \Rightarrow \ \star} \text{ A-Ax}$$

$[\Gamma](\star) = \star$ Directly from the definition of context application
$\Gamma \vdash \star \ \Rightarrow \ \star$ By A-Ax

– Case AVar, AEVar are similar as AAx, which follows from definition directly.

– Case

$$\frac{\Gamma \ ctx \qquad \widehat{\alpha} = \tau \in \Gamma}{\Gamma \vdash \widehat{\alpha} \ \Rightarrow \ \star} \text{ A-SolvedEVar}$$

let $\Gamma = \Gamma_1, \widehat{\alpha} = \tau, \Gamma_2$
$[\Gamma](\widehat{\alpha}) = [\Gamma_1](\tau)$ Follows from definition of context application
$\Gamma_1 \vdash [\Gamma_1](\tau) \ \Rightarrow \ \star$ By hypothesis
$\Gamma_1, \widehat{\alpha} = \tau, \Gamma_2 \vdash [\Gamma_1](\tau) \ \Rightarrow \ \star$ By Lemma 16

– Case

$$\frac{\Gamma \vdash \sigma_1 \;\Rightarrow\; \star \qquad \Gamma, x : \sigma_1 \vdash e \;\Rightarrow\; \sigma_2}{\Gamma \vdash \lambda x : \sigma_1.\ e \;\Rightarrow\; \Pi x : [\Gamma](\sigma_1).\ \sigma_2}\ \textsc{A-LamAnn}$$

$[\Gamma](\lambda x : \sigma_1.\ e) = \lambda x : [\Gamma](\sigma_1).\ [\Gamma](e)$	Follows from definition of context application
$\Gamma \vdash [\Gamma](\sigma_1) \;\Rightarrow\; \star$	By hypothesis
$\Gamma, x : \sigma_1 \vdash [\Gamma, x : \sigma_1](e) \;\Rightarrow\; \sigma_2$	By hypothesis
$\Gamma, x : \sigma_1 \vdash [\Gamma](e) \;\Rightarrow\; \sigma_2$	By definition of context application
$\Gamma, x : [\Gamma](\sigma_1) \vdash [\Gamma](e) \;\Rightarrow\; \sigma_2$	By Lemma 11

– Case

$$\frac{\Gamma \vdash \sigma_1 \;\Rightarrow\; \star \qquad \Gamma, x : \sigma_1 \vdash \sigma_2 \;\Rightarrow\; \star}{\Gamma \vdash \Pi x : \sigma_1.\ \sigma_2 \;\Rightarrow\; \star}\ \textsc{A-Pi}$$

$[\Gamma](\Pi x : \sigma_1.\ \sigma_2) = \Pi x : [\Gamma](\sigma_1).\ [\Gamma](\sigma_2)$	Follows from definition of context application
$\Gamma \vdash [\Gamma](\sigma_1) \;\Rightarrow\; \star$	By hypothesis
$\Gamma, x : \sigma_1 \vdash [\Gamma, x : \sigma_1](\sigma_2) \;\Rightarrow\; \star$	By hypothesis
$\Gamma, x : \sigma_1 \vdash [\Gamma](\sigma_2) \;\Rightarrow\; \star$	By definition of context application
$\Gamma, x : [\Gamma](\sigma_1) \vdash [\Gamma](\sigma_2) \;\Rightarrow\; \star$	By Lemma 11

– Case A-App, A-CastDn, A-CastUp follows directly from the hypothesis.

□

Lemma 14 (Reverse Context Application Preserves Typing). *If* $\Gamma \vdash [\Gamma](\sigma_1) \;\Rightarrow\; \sigma_2$, *then* $\Gamma \vdash \sigma_1 \;\Rightarrow\; \sigma_2$.

Proof. By induction on the typing derivation.

Then we analyze the shape of σ_1. Because when we do inversion, there is always one case that $\sigma_1 = \widehat{\alpha}$ for some $\widehat{\alpha}$, we deal with the case first and then ignore all the cases when σ is an existential variable.

– Case $\sigma = \widehat{\alpha}$. According to $\widehat{\alpha}$ is solved or not in Γ, we have two subcases.
  - $\Gamma = \Gamma_1, \widehat{\alpha}, \Gamma_2$. Then $[\Gamma](\widehat{\alpha}) = \widehat{\alpha}$. Therefore the goal follows trivially.
  - $\Gamma = \Gamma_1, \widehat{\alpha} = \tau, \Gamma_2$.

$\Gamma\ ctx$	By Lemma 15
$\Gamma \vdash \widehat{\alpha} \;\Rightarrow\; \star$	By A-SolvedEVar

– Case $[\Gamma](\sigma_1) = \star$

$\sigma_1 = \star$	By inversion
$\Gamma \vdash \star \;\Rightarrow\; \star$	By A-Ax

– Case $[\Gamma](\sigma_1) = x$

$\sigma_1 = x$ By inversion
$\Gamma \vdash x \Rightarrow \sigma_2$ By A-Var

– Case $[\Gamma](\sigma_1) = \widehat{\alpha}$ Then we have $\sigma_1 = \widehat{\beta}$ for some $\widehat{\beta}$. We can prove it as in the first case.
– Case $[\Gamma](\sigma_1) = \lambda x : \tau_1.\ e_1$

$\sigma_1 = \lambda x : \tau_2.\ e_2$	By inversion
$[\Gamma](\tau_2) = \tau_1$	As above
$[\Gamma](e_2) = e_1$	As above
$\Gamma \vdash \tau_1 \Rightarrow \star$	By inversion
$\Gamma, x : \tau_1 \vdash e_1 \Rightarrow \sigma$	As above
$\sigma_2 = \Pi x : [\Gamma](\tau_1).\ \sigma$	Given
$\Gamma \vdash \tau_2 \Rightarrow \star$	By induction hypothesis
$\Gamma, x : \tau_1 \vdash e_2 \Rightarrow \sigma$	By induction hypothesis
$\Gamma, x : [\Gamma](\tau_2) \vdash e_2 \Rightarrow \sigma$	By substituting the equality
$\Gamma, x : \tau_2 \vdash e_2 \Rightarrow \sigma$	By Lemma 12
$\Gamma \vdash \sigma_1 \Rightarrow \Pi x : [\Gamma](\tau_2).\ \sigma$	By A-LamAnn
$[\Gamma](\tau)_1 = \tau_1$	By 7
$\Gamma \vdash \sigma_1 \Rightarrow \sigma_2$	By substituting the equalities

– Case A-Pi and A-App are similar as last case.
– Case $\sigma_1 = \mathsf{cast}_{\downarrow}\ e_1$

$\sigma_1 = \mathsf{cast}_{\downarrow}\ e_2$	By inversion
$[\Gamma](e_2) = e_1$	As above
$\Gamma \vdash e_1 \Rightarrow \sigma$	By inversion
$\sigma \hookrightarrow \sigma_2$	As above
$\Gamma \vdash e_2 \Rightarrow \sigma$	By induction hypothesis
$\Gamma \vdash \mathsf{cast}_{\downarrow}\ e_2 \Rightarrow \sigma_2$	By A-CastDn

– Case $\sigma_1 = \mathsf{cast}_{\uparrow}\ e_1$ Similar as last case.

□

C.2 Properties of Declarative Typing

Lemma 15 (Typing Context Well Formedness). *If $\Gamma \vdash \tau \Rightarrow \sigma$, then Γ ctx .*

Proof. By a straightforward induction on the typing derivation. □

Notice since we have this lemma hold, if we have $\Gamma \vdash \sigma_1 \Rightarrow \sigma_2$ in the context, we will implicitly assume we already have Γ *ctx* . Therefore, to apply lemmas that request the well-formedness of contexts, which is automatically hold given

the typing derivation, we will sometimes not explicitly provide that the context is well formed.

Lemma 16 (Typing Weakening). *If* $\Gamma_1, \Gamma_2 \vdash \sigma_1 \Rightarrow \sigma_2$, *and* $\Gamma_1, \Theta, \Gamma_2$ *ctx* , *then* $\Gamma_1, \Theta, \Gamma_2 \vdash \sigma_1 \Rightarrow \sigma_2$.

Proof. By induction on the typing derivation.

- Case A-Ax, A-Var, A-EVar and A-SolvedEVar follows directly.
- Case

$$\frac{\Gamma \vdash \sigma_1 \Rightarrow \star \qquad \Gamma, x : \sigma_1 \vdash e \Rightarrow \sigma_2}{\Gamma \vdash \lambda x : \sigma_1.\ e \Rightarrow \Pi x : [\Gamma](\sigma_1).\ \sigma_2}\ \text{A-LamAnn}$$

$\Gamma_1, \Theta, \Gamma_2 \vdash \sigma_1 \Rightarrow \star$	By hypothesis
$\Gamma_1, \Theta, \Gamma_2, x : \sigma_1$ *ctx*	By AC-Var
$\Gamma_1, \Theta, \Gamma_2, x : \sigma_1 \vdash e \Rightarrow \sigma_2$	By hypothesis

- Case A-Pi is similar as A-LamAnn.
- Case A-App, A-CastDn and A-CastUp follows directly from hypothesis. □

Lemma 17 (Typing Strengthening). *If* $\Gamma_1, \Gamma_2, \Gamma_3 \vdash \tau \Rightarrow \sigma$, *and* Γ_1, Γ_3 *ctx* , *and* $\Gamma_1, \Gamma_3 \vDash \tau$, *then* $\Gamma_1, \Gamma_3 \vdash \tau \Rightarrow \sigma$.

Proof. By induction on the typing derivation.

- Case A-Ax Follows trivially.
- Case

$$\frac{\Gamma\ \textit{ctx} \qquad x : \sigma \in \Gamma}{\Gamma \vdash x \Rightarrow [\Gamma](\sigma)}\ \text{A-Var}$$

$\Gamma_1, \Gamma_3 \vDash x$	Given
$x : \sigma \in \Gamma_1, \Gamma_3$	By inversion
$\Gamma_1, \Gamma_3 \vdash x \Rightarrow [\Gamma_1, \Gamma_3](\sigma)$	By A-Var
Γ_1, Γ_3 *ctx*	Given
Γ_1, Γ_3	does not contain any existential variable in Γ_2
$[\Gamma_1, \Gamma_3](\sigma) = [\Gamma_1, \Gamma_2, \Gamma_3](\sigma)$	Follows directly

-

$$\frac{\Gamma\ \textit{ctx} \qquad \widehat{\alpha} \in \Gamma}{\Gamma \vdash \widehat{\alpha} \Rightarrow \star}\ \text{A-EVar}$$

$\Gamma_1, \Gamma_3 \vDash \widehat{\alpha}$	Given
$\widehat{\alpha} \in \Gamma_1, \Gamma_3$	By inversion
$\Gamma_1, \Gamma_3 \vdash \widehat{\alpha} \Rightarrow \star$	By A-EVar

–

$$\frac{\Gamma\ ctx \qquad \widehat{\alpha} = \tau \in \Gamma}{\Gamma \vdash \widehat{\alpha}\ \Rightarrow\ \star}\ \textsc{A-SolvedEVar}$$

$\Gamma_1, \Gamma_3 \vDash \widehat{\alpha}$	Given
$\widehat{\alpha} = \tau \in \Gamma_1, \Gamma_3$	By inversion
$\Gamma_1, \Gamma_3 \vdash \widehat{\alpha}\ \Rightarrow\ \star$	By A-EVar

–

$$\frac{\Gamma \vdash \sigma_1\ \Rightarrow\ \star \qquad \Gamma, x : \sigma_1 \vdash e\ \Rightarrow\ \sigma_2}{\Gamma \vdash \lambda x : \sigma_1.\ e\ \Rightarrow\ \Pi x : [\Gamma](\sigma_1).\ \sigma_2}\ \textsc{A-LamAnn}$$

$\Gamma_1, \Gamma_3 \vDash \sigma_1$	Given
$\Gamma_1, \Gamma_3 \vdash \sigma_1\ \Rightarrow\ \star$	By induction hypothesis
$\Gamma_1, \Gamma_3, x : \sigma_1\ ctx$	By AC-Var
$\Gamma_1, \Gamma_3, x : \sigma_1 \vDash e$	Given
$\Gamma_1, \Gamma_3, x : \sigma_1 \vdash e\ \Rightarrow\ \sigma_2$	By induction hypothesis
$\Gamma_1, \Gamma_3 \vdash \lambda x : \sigma_1.\ e\ \Rightarrow\ \Pi x : [\Gamma_1, \Gamma_3](\sigma_1).\ \sigma_2$	By A-LamAnn
$\Gamma_1, \Gamma_3\ ctx$	Given
Γ_1, Γ_3	does not contain any existential variable in Γ_2
$[\Gamma_1, \Gamma_3](\sigma_1) = [\Gamma_1, \Gamma_2, \Gamma_3](\sigma_1)$	Follows directly

–

$$\frac{\Gamma \vdash \sigma_1\ \Rightarrow\ \star \qquad \Gamma, x : \sigma_1 \vdash \sigma_2\ \Rightarrow\ \star}{\Gamma \vdash \Pi x : \sigma_1.\ \sigma_2\ \Rightarrow\ \star}\ \textsc{A-Pi}$$

Similar as Case A-LamAnn.

–

$$\frac{\Gamma \vdash e_1\ \Rightarrow\ \Pi x : \sigma_1.\ \sigma_2 \qquad \Gamma \vdash e_2\ \Rightarrow\ \sigma_1}{\Gamma \vdash e_1\ e_2\ \Rightarrow\ \sigma_2[x \mapsto [\Gamma](e_1)]}\ \textsc{A-App}$$

$\Gamma_1, \Gamma_3 \vDash e_1$	Given
$\Gamma_1, \Gamma_3 \vdash e_1\ \Rightarrow\ \Pi x : \sigma_1.\ \sigma_2$	By induction hypothesis
$\Gamma_1, \Gamma_3 \vDash e_2$	Given
$\Gamma_1, \Gamma_3 \vdash e_2\ \Rightarrow\ \sigma_1$	By induction hypothesis
$\Gamma_1, \Gamma_3 \vdash e_1\ e_2\ \Rightarrow\ \sigma_2[x \mapsto [\Gamma_1, \Gamma_3](e_1)]$	By A-App
$\Gamma_1, \Gamma_3 \vDash e_1$	Given
$\Gamma_1, \Gamma_3\ ctx$	Given
Γ_1, Γ_3, e_1	does not contain any existential variable in Γ_2
$[\Gamma_1, \Gamma_3](e_1) = [\Gamma_1, \Gamma_2, \Gamma_3](\sigma_1)$	Follows directly

–

$$\frac{\Gamma \vdash e\ \Rightarrow\ \sigma_1 \qquad \sigma_1 \hookrightarrow \sigma_2}{\Gamma \vdash \mathsf{cast}_{\downarrow}\ e\ \Rightarrow\ \sigma_2}\ \textsc{A-CastDn}$$

$\Gamma_1, \Gamma_3 \vDash e$	Given
$\Gamma_1, \Gamma_3 \vdash e \Rightarrow \sigma_1$	By induction hypothesis
$\sigma_1 \hookrightarrow \sigma_2$	Given
$\Gamma_1, \Gamma_3 \vdash \mathsf{cast}_{\downarrow}\ e \Rightarrow \sigma_2$	By A-CastDn

–

$$\frac{\Gamma \vdash e \Rightarrow \sigma_2 \qquad [\Gamma](\sigma_1) \hookrightarrow \sigma_2}{\Gamma \vdash \mathsf{cast}_{\uparrow}\ e \Rightarrow [\Gamma](\sigma_1)}\ \text{A-CastUp}$$

Similar as Case A-CastDn.

□

Lemma 18 (Typing Variable Exchange). *If $\Gamma, x : \sigma_1, y : \sigma_2, \Theta \vdash \tau_1 \Rightarrow \tau_2$, and $\Gamma, y : \sigma_2, x : \sigma_1, \Theta\ ctx$, then $\Gamma, y : \sigma_2, x : \sigma_1, \Theta \vdash \tau_1 \Rightarrow \tau_2$ with the same size of typing derivation.*

Proof. By straightforward induction on the typing derivation. □

Lemma 19 (Typing Substitution). *If $\Gamma \vdash \tau \Rightarrow [\Gamma](\sigma_1)$, and $\Gamma, x : \sigma_1 \vdash \tau' \Rightarrow \sigma_2$, then $\Gamma \vdash \tau'[x \mapsto \tau] \Rightarrow \sigma_2$.*

Proof. By induction on the size of second typing derivation.

– Case A-Ax, A-EVar and A-SolvedEVar follows from typing rules directly.
– Case

$$\frac{\Gamma\ ctx \qquad x : \sigma \in \Gamma}{\Gamma \vdash x \Rightarrow [\Gamma](\sigma)}\ \text{A-Var}$$

If two x's are the same, then $\sigma_2 = [\Gamma](\sigma_1)$. The goal follows directly from given conditions. If two variables are not the same, then it follows directly from A-Var.
– Case

$$\frac{\Gamma \vdash \sigma_1 \Rightarrow \star \qquad \Gamma, x : \sigma_1 \vdash e \Rightarrow \sigma_2}{\Gamma \vdash \lambda x : \sigma_1.\ e \Rightarrow \Pi x : [\Gamma](\sigma_1).\ \sigma_2}\ \text{A-LamAnn}$$

Let the expression be $\lambda y : \sigma'.\ e$ to avoid name conflicts.

$\Gamma \vdash \sigma'[x \mapsto \tau] \Rightarrow \star$	By induction hypothesis
$\Gamma, x : \sigma_1, y : \sigma'[x \mapsto \tau] \vdash e \Rightarrow \sigma_3$	Given, notice we use σ_3 to avoid name conflicts
$\Gamma \vdash \sigma_1 \Rightarrow \star$	Given
$\Gamma, y : \sigma'[x \mapsto \tau] \vdash \sigma_1 \Rightarrow \star$	By Lemma 16
$\Gamma, y : \sigma'[x \mapsto \tau], x : \sigma_1\ ctx$	By AC-Var
$\Gamma, y : \sigma'[x \mapsto \tau], x : \sigma_1 \vdash e \Rightarrow \sigma_3$	By Lemma 18
$\Gamma \vdash \tau \Rightarrow [\Gamma](\sigma_1)$	Given
$\Gamma, y : \sigma_1[x \mapsto \tau] \vdash \tau \Rightarrow [\Gamma](\sigma_1)$	By Lemma 16
$\Gamma, y : \sigma_1[x \mapsto \tau] \vdash e[x \mapsto \tau] \Rightarrow \sigma_3$	By induction hypothesis

– Case A-Pi is similar as A-LamAnn.
– Rest cases follow directly from hypothesis.

□

C.3 Properties of Context Extension

Lemma 20 (Declaration Preservation). *If $\Gamma \longrightarrow \Delta$, and u is a variable declared in Γ, then u is declared in Δ.*

Proof. By a straightforward induction on the context application. □

Lemma 21 (Reverse Declaration Preservation). *If $\Gamma \longrightarrow \Theta$, and u is a variable that $u \notin \Theta$, then $u \notin \Gamma$.*

Proof. By a straightforward induction on the context application. □

Lemma 22 (Declaration Order Preservation). *If $\Gamma \longrightarrow \Delta$, and u is declared to the left of v in Γ, then u is declared to the left of v in the Δ.*

Proof. By induction on the context application.

– Case

$$\frac{}{\varnothing \longrightarrow \varnothing}\ \text{CE-Empty}$$

Impossible case.
– Case

$$\frac{\Gamma \longrightarrow \Delta}{\Gamma, x : \sigma \longrightarrow \Delta, x : \sigma}\ \text{CE-Var}$$

- SubCase $v = x$.

$u \in \Gamma$	Given
$u \in \Delta$	By Lemma 20

So u is to the left of x in Γ, x

- SubCase $v \neq x$.

u is declared to the left of v in Γ	Given
u is declared to the left of v in Δ	By induction hypothesis

u is declared to the left of v in $\Delta, x : \sigma$

– Case CE-EVar, CE-SolvedEVar, CE-Solve are similar as Case CE-Var.
– Case

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta}{\Gamma \longrightarrow \Delta, \widehat{\alpha}}\ \text{CE-Add}$$

u is declared to the left of v in Δ By induction hypothesis
So u is declared to the left of v in $\Delta, \widehat{\alpha}$

– Case

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta \qquad \Delta \vdash \tau}{\Gamma \longrightarrow \Delta, \widehat{\alpha} = \tau}\ \text{CE-AddSolved}$$

u is declared to the left of v in Δ By induction hypothesis
So u is declared to the left of v in $\Delta, \widehat{\alpha} = \tau$

□

Lemma 23 (Reverse Declaration Order Preservation). *If $\Gamma \longrightarrow \Delta$, and u and v are both declared in Γ, and u is declared to the left of v in Δ, then u is declared to the left of v in the Γ.*

Proof. We can prove this lemma by contradiction. Suppose u is declared to the right of v in Γ, then by Lemma 22, we have that u is declared to the right of v in Γ. Because we already have u is declared to the left of v, we have a contradiction. Therefore, u is declared to the left of v in Γ.

□

Lemma 24 (Substitution Extension Invariance). *If $\Gamma \vdash \sigma \Rightarrow \sigma'$, and $\Gamma \longrightarrow \Delta$, then $[\Delta](\sigma) = [\Delta]([\Gamma](\sigma))$, and $[\Delta](\sigma) = [\Gamma]([\Delta](\sigma))$.*

Proof. We first prove $[\Delta](\sigma) = [\Gamma]([\Delta](\sigma))$, then prove $[\Delta](\sigma) = [\Delta]([\Gamma](\sigma))$.

Part 1 We do induction on the context extension.

– Case

$$\frac{}{\varnothing \longrightarrow \varnothing}\ \text{CE-Empty}$$

Trivial case.

– Case

$$\frac{\Gamma \longrightarrow \Delta}{\Gamma, x : \sigma \longrightarrow \Delta, x : \sigma}\ \text{CE-Var}$$

We use σ_1 to replace the σ in CE-Var.

$[\Delta, x : \sigma_1](\sigma)$
$= [\Delta](\sigma)$ Follows by definition of context application
$= [\Gamma]([\Delta](\sigma))$ By induction hypothesis
$= [\Gamma, x : \sigma_1]([\Delta, x : \sigma_1](\sigma))$ By definition of context application

– Case

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta}{\Gamma, \widehat{\alpha} \longrightarrow \Delta, \widehat{\alpha}}\ \text{CE-EVar}$$

Similar as Case CE-Var.

– Case

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta \qquad [\Delta](\tau_1) = [\Delta](\tau_2)}{\Gamma, \widehat{\alpha} = \tau_1 \longrightarrow \Delta, \widehat{\alpha} = \tau_2}\ \text{CE-SolvedEVar}$$

$[\Delta, \widehat{\alpha} = \tau_2](\sigma)$
$= [\Delta](\sigma[\widehat{\alpha} \mapsto \tau_2])$ By definition of context application
$= [\Gamma]([\Delta](\sigma[\widehat{\alpha} \mapsto \tau_2]))$ By induction hypothesis
$= [\Gamma, \widehat{\alpha} = \tau_1]([\Delta](\sigma[\widehat{\alpha} \mapsto \tau_2]))$ $\widehat{\alpha} \notin FV([\Delta](\sigma[\widehat{\alpha} \mapsto \tau_2]))$
$= [\Gamma, \widehat{\alpha} = \tau_1]([\Delta, \widehat{\alpha} = \tau_2](\sigma))$ By definition of context application

– Case

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta \qquad \Delta \vdash \tau}{\Gamma, \widehat{\alpha} \longrightarrow \Delta, \widehat{\alpha} = \tau}\ \text{CE-Solve}$$

$[\Delta, \widehat{\alpha} = \tau](\sigma)$
$= [\Delta](\sigma[\widehat{\alpha} \mapsto \tau])$ By definition of context application
$= [\Gamma]([\Delta](\sigma[\widehat{\alpha} \mapsto \tau]))$ By induction hypothesis
$= [\Gamma, \widehat{\alpha}]([\Delta](\sigma[\widehat{\alpha} \mapsto \tau]))$ By definition of context application
$= [\Gamma, \widehat{\alpha}]([\Delta, \widehat{\alpha} = \tau](\sigma))$ By definition of context application

– Case

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta}{\Gamma \longrightarrow \Delta, \widehat{\alpha}}\ \text{CE-Add}$$

$[\Delta, \widehat{\alpha}](\sigma)$
$= [\Delta](\sigma)$ By definition of context application
$= [\Gamma]([\Delta](\sigma))$ By induction hypothesis
$= [\Gamma]([\Delta, \widehat{\alpha}](\sigma))$ By definition of context application

– Case

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta \qquad \Delta \vdash \tau}{\Gamma \longrightarrow \Delta, \widehat{\alpha} = \tau}\ \text{CE-AddSolved}$$

$[\Delta, \widehat{\alpha} = \tau](\sigma)$
$= [\Delta](\sigma[\widehat{\alpha} \mapsto \tau])$ By definition of context application
$= [\Gamma]([\Delta](\sigma[\widehat{\alpha} \mapsto \tau]))$ By induction hypothesis
$= [\Gamma]([\Delta, \widehat{\alpha} = \tau](\sigma))$ By definition of context application

Part 2 By induction on the size of the typing derivation.

– Case

$$\frac{\Gamma \ ctx}{\Gamma \vdash \star \ \Rightarrow \ \star} \text{ A-Ax}$$

Follows directly by for all context Γ, $[\Gamma](\star) = \star$.

– Case

$$\frac{\Gamma \ ctx \qquad x : \sigma \in \Gamma}{\Gamma \vdash x \ \Rightarrow \ [\Gamma](\sigma)} \text{ A-Var}$$

Follows directly by for all context Γ, $[\Gamma](x) = x$.

– Case

$$\frac{\Gamma \ ctx \qquad \widehat{\alpha} \in \Gamma}{\Gamma \vdash \widehat{\alpha} \ \Rightarrow \ \star} \text{ A-EVar}$$

$[\Gamma](\widehat{\alpha}) = \widehat{\alpha}$
$[\Delta]([\Gamma](\widehat{\alpha})) = [\Delta](\widehat{\alpha})$ Follows directly

– Case

$$\frac{\Gamma \ ctx \qquad \widehat{\alpha} = \tau \in \Gamma}{\Gamma \vdash \widehat{\alpha} \ \Rightarrow \ \star} \text{ A-SolvedEVar}$$

$\Gamma \longrightarrow \Delta$	Given
$\widehat{\alpha} = \tau \in \Gamma$	Given
$\Delta = \Delta_1, \widehat{\alpha} = \tau_2, \Delta_2$	By Lemma 29
$[\Delta_1](\tau) = [\Delta_1](\tau_2)$	As above
$[\Delta](\tau) = [\Delta](\tau_2)$	Since $\widehat{\alpha}, \Delta_2$ should be fresh contexts for τ, τ_2
$[\Delta](\widehat{\alpha})$	
$= [\Delta](\tau_2)$	By definition of context application
$= [\Delta](\tau)$	Known
$= [\Delta]([\Gamma](\tau))$	By induction hypothesis
$= [\Delta]([\Gamma](\widehat{\alpha}))$	By definition of context application

– Case

$$\frac{\Gamma \vdash \sigma_1 \ \Rightarrow \ \star \qquad \Gamma, x : \sigma_1 \vdash e \ \Rightarrow \ \sigma_2}{\Gamma \vdash \lambda x : \sigma_1.\ e \ \Rightarrow \ \Pi x : [\Gamma](\sigma_1).\ \sigma_2} \text{ A-LamAnn}$$

$[\Delta](\sigma) = [\Delta]([\Gamma](\sigma_1))$	By induction hypothesis
$\Gamma \longrightarrow \Delta$	Given
$\Gamma, x : \sigma_1 \longrightarrow \Delta, x : \sigma_1$	By CE-Var
$[\Delta, x : \sigma_1](e) = [\Delta, x : \sigma_1]([\Gamma, x : \sigma_1](e))$	By induction hypothesis
$[\Delta](e) = [\Delta]([\Gamma](e))$	By definition of context application

– Case

$$\frac{\Gamma \vdash \sigma_1 \;\Rightarrow\; \star \qquad \Gamma, x : \sigma_1 \vdash \sigma_2 \;\Rightarrow\; \star}{\Gamma \vdash \Pi x : \sigma_1.\ \sigma_2 \;\Rightarrow\; \star}\ \text{A-Pi}$$

Similar as Case A-LamAnn.

– Case

$$\frac{\Gamma \vdash e_1 \;\Rightarrow\; \Pi x : \sigma_1.\ \sigma_2 \qquad \Gamma \vdash e_2 \;\Rightarrow\; \sigma_1}{\Gamma \vdash e_1\ e_2 \;\Rightarrow\; \sigma_2[x \mapsto [\Gamma](e_1)]}\ \text{A-App}$$

Follows directly from induction hypothesis.

– Case

$$\frac{\Gamma \vdash e \;\Rightarrow\; \sigma_1 \qquad \sigma_1 \hookrightarrow \sigma_2}{\Gamma \vdash \mathsf{cast}_{\downarrow}\ e \;\Rightarrow\; \sigma_2}\ \text{A-CastDn}$$

Follows directly from induction hypothesis.

– Case

$$\frac{\Gamma \vdash e \;\Rightarrow\; \sigma_2 \qquad [\Gamma](\sigma_1) \hookrightarrow \sigma_2}{\Gamma \vdash \mathsf{cast}_{\uparrow}\ e \;\Rightarrow\; [\Gamma](\sigma_1)}\ \text{A-CastUp}$$

Follows directly from induction hypothesis.

□

Lemma 25 (Extension Equality Preservation). *If* $\Gamma \vdash \sigma_1 \;\Rightarrow\; \tau_1$, *and* $\Gamma \vdash \sigma_2 \;\Rightarrow\; \tau_2$, *and* $[\Gamma](\sigma_1) = [\Gamma](\sigma_2)$, *and* $\Gamma \longrightarrow \Delta$, *then* $[\Delta](\sigma_1) = [\Delta](\sigma_2)$.

Proof.

$[\Delta](\sigma_1)$
$= [\Delta]([\Gamma](\sigma_1))$ By Lemma 24
$= [\Delta]([\Gamma](\sigma_2))$ Given
$= [\Delta](\sigma_2)$ By Lemma 24

□

Lemma 26 (Reflexivity of Context Extension). *If* $\Gamma\ ctx$, *then* $\Gamma \longrightarrow \Gamma$.

Proof. By induction on the well-formedness of context.

– Case

$$\frac{}{\varnothing\ ctx}\ \text{AC-Empty}$$

Follows directly from CE-Empty.

– Case

$$\frac{\Gamma\ ctx \qquad x \notin \Psi \qquad \Gamma \vdash \sigma \;\Rightarrow\; \star}{\Gamma, x : \sigma\ ctx}\ \text{AC-Var}$$

$\Gamma \longrightarrow \Gamma$ By induction hypothesis
$\Gamma, x : \sigma \longrightarrow \Gamma, x : \sigma$ By CE-Var

– Case

$$\frac{\Gamma\ ctx \qquad \widehat{\alpha} \notin \Psi}{\Gamma, \widehat{\alpha}\ ctx}\ \textsc{ac-evar}$$

$\Gamma \longrightarrow \Gamma$	By induction hypothesis
$\widehat{\alpha} \notin \Gamma$	Given
$\Gamma, \widehat{\alpha} \longrightarrow \Gamma, \widehat{\alpha}$	By CE-EVar

– Case

$$\frac{\Gamma\ ctx \qquad \widehat{\alpha} \notin \Gamma \qquad \Gamma \vdash \tau\ \Rightarrow\ \star}{\Gamma, \widehat{\alpha} = \tau\ ctx}\ \textsc{ac-SolvedEVar}$$

$\Gamma \longrightarrow \Gamma$	By induction hypothesis
$\widehat{\alpha} \notin \Gamma$	Given
$\Gamma, \widehat{\alpha} = \tau \longrightarrow \Gamma, \widehat{\alpha} = \tau$	By CE-SolvedEVar

□

Lemma 27 (Transitivity of Context Extension). *Given Θ, Γ, Δ are all well-formed contexts, if $\Theta \longrightarrow \Gamma$, and $\Gamma \longrightarrow \Delta$, then $\Theta \longrightarrow \Delta$.*

Proof. By induction on the derivation $\Gamma \longrightarrow \Delta$.

– Case

$$\frac{}{\varnothing \longrightarrow \varnothing}\ \textsc{ce-Empty}$$

Our goal $\Theta \longrightarrow \varnothing$ is given.

– Case

$$\frac{\Gamma \longrightarrow \Delta}{\Gamma, x : \sigma \longrightarrow \Delta, x : \sigma}\ \textsc{ce-Var}$$

$\Theta \longrightarrow \Gamma, x : \sigma$	Given
$\Theta = \Theta', x : \sigma$	By inversion
$\Theta' \longrightarrow \Gamma$	By inversion
$\Theta' \longrightarrow \Delta$	By induction hypothesis
$\Theta', x : \sigma \longrightarrow \Delta, x : \sigma$	By CE-Var
$\Theta \longrightarrow \Delta, x : \sigma$	Namely

– Case

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta}{\Gamma, \widehat{\alpha} \longrightarrow \Delta, \widehat{\alpha}}\ \textsc{ce-EVar}$$

We are given $\Theta \longrightarrow \Gamma, \widehat{\alpha}$. By inversion, we have two subcases.

- SubCase

$$\frac{\Theta \longrightarrow \Gamma \qquad \widehat{\alpha} \notin \Gamma}{\Theta, \widehat{\alpha} \longrightarrow \Gamma, \widehat{\alpha}}\ \textsc{ce-EVar}$$

$\Theta \longrightarrow \Gamma$ Given
$\Theta \longrightarrow \Delta$ By induction hypothesis
$\Theta, \widehat{\alpha} \longrightarrow \Delta, \widehat{\alpha}$ By CE-EVar

- SubCase

$$\frac{\Theta \longrightarrow \Gamma \qquad \widehat{\alpha} \notin \Gamma}{\Theta \longrightarrow \Gamma, \widehat{\alpha}} \text{ CE-Add}$$

$\Theta \longrightarrow \Gamma$ Given
$\Theta \longrightarrow \Delta$ By induction hypothesis
$\Theta \longrightarrow \Delta, \widehat{\alpha}$ By CE-Add

– Case

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta \qquad [\Delta](\tau_1) = [\Delta](\tau_2)}{\Gamma, \widehat{\alpha} = \tau_1 \longrightarrow \Delta, \widehat{\alpha} = \tau_2} \text{ CE-SolvedEVar}$$

We are given $\Theta \longrightarrow \Gamma, \widehat{\alpha} = \tau_1$. By inversion, we have three subcases.

- SubCase

$$\frac{\Theta \longrightarrow \Gamma \qquad \widehat{\alpha} \notin \Gamma \qquad [\Gamma](\tau_1) = [\Gamma](\tau_3)}{\Theta, \widehat{\alpha} = \tau_3 \longrightarrow \Gamma, \widehat{\alpha} = \tau_1} \text{ CE-SolvedEVar}$$

$\Theta \longrightarrow \Delta$ By induction hypothesis
$\Theta, \widehat{\alpha} = \tau_3\ ctx$ Given
$\Theta \vdash \tau_3$ By inversion
$\Gamma \vdash \tau_3$ By Corollary 32
$[\Delta](\tau_3)$
$= [\Delta]([\Gamma](\tau_3))$ By Lemma 24
$= [\Delta]([\Gamma](\tau_1))$ Given
$= [\Delta](\tau_1)$ By Lemma 24
$= [\Delta](\tau_2)$ Known
$\Theta, \widehat{\alpha} = \tau_3 \longrightarrow \Delta, \widehat{\alpha} = \tau_2$ By CE-SolvedEVar

- SubCase

$$\frac{\Theta \longrightarrow \Gamma \qquad \widehat{\alpha} \notin \Gamma \qquad \Gamma \vdash \tau}{\Theta, \widehat{\alpha} \longrightarrow \Gamma, \widehat{\alpha} = \tau} \text{ CE-Solve}$$

$\Theta \longrightarrow \Gamma$ Given
$\Theta \longrightarrow \Delta$ By induction hypothesis
$\Gamma \vdash \tau$ Given
$\Gamma\ ctx$ By Lemma 15
$\Delta\ ctx$ By Lemma 32
$\Delta \vdash \tau$ By Corollary 3

$\Theta \longrightarrow \Delta, \widehat{\alpha} = \tau$ By CE-AddSolved

– Case

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta \qquad \Delta \vdash \tau}{\Gamma, \widehat{\alpha} \longrightarrow \Delta, \widehat{\alpha} = \tau}\ \text{CE-Solve}$$

We are given $\Theta \longrightarrow \Gamma, \widehat{\alpha}$. By induction, we have two subcases.

- SubCase

$$\frac{\Theta \longrightarrow \Gamma \qquad \widehat{\alpha} \notin \Gamma}{\Theta, \widehat{\alpha} \longrightarrow \Gamma, \widehat{\alpha}}\ \text{CE-EVar}$$

$\Theta \longrightarrow \Gamma$ Given
$\Theta \longrightarrow \Delta$ By induction hypothesis
$\Theta, \widehat{\alpha} \longrightarrow \Delta, \widehat{\alpha} = \tau$ By CE-Solve

- SubCase

$$\frac{\Theta \longrightarrow \Gamma \qquad \widehat{\alpha} \notin \Gamma}{\Theta \longrightarrow \Gamma, \widehat{\alpha}}\ \text{CE-Add}$$

$\Theta \longrightarrow \Gamma$ Given
$\Theta \longrightarrow \Delta$ By induction hypothesis
$\Theta \longrightarrow \Delta, \widehat{\alpha} = \tau$ By CE-AddSolved

– Case

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta}{\Gamma \longrightarrow \Delta, \widehat{\alpha}}\ \text{CE-Add}$$

$\Theta \longrightarrow \Gamma$ Given
$\Theta \longrightarrow \Delta$ By induction hypothesis
$\Theta \longrightarrow \Delta, \widehat{\alpha}$ By CE-Add

– Case

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta \qquad \Delta \vdash \tau}{\Gamma \longrightarrow \Delta, \widehat{\alpha} = \tau}\ \text{CE-AddSolved}$$

$\Theta \longrightarrow \Gamma$ Given
$\Theta \longrightarrow \Delta$ By induction hypothesis
$\Theta \longrightarrow \Delta, \widehat{\alpha} = \tau$ By CE-Add

□

Definition 2 (Softness). *A Context Γ is soft if and only if it consists only of $\widehat{\alpha}$ and $\widehat{\alpha} = \tau$ declarations.*

Lemma 28 (Right Softness). *If $\Gamma \longrightarrow \Delta$, and Θ is soft, and Δ, Θ is well formed, then $\Gamma \longrightarrow \Delta, \Theta$.*

Proof. By induction on Θ, and apply rule CE-Add and CE-AddSolved as needed. □

Lemma 29 (Extension Order).

- *If $\Gamma_L, y : \sigma, \Gamma_R \longrightarrow \Delta$, then $\Delta = \Delta_L, y : \sigma, \Delta_R$, and $\Gamma_L \longrightarrow \Delta_L$, and Δ_R is soft if and only if Γ_R is soft.*
- *If $\Gamma_L, \widehat{\alpha}, \Gamma_R \longrightarrow \Delta$, then $\Delta = \Delta_L, \Theta, \Delta_R$, and $\Gamma_L \longrightarrow \Delta_L$, and Θ is either $\widehat{\alpha}$ or $\widehat{\alpha} = \tau$ for some τ, and Δ_R is soft if and only if Γ_R is soft.*
- *If $\Gamma_L, \widehat{\alpha} = \sigma_1, \Gamma_R \longrightarrow \Delta$, then $\Delta = \Delta_L, \widehat{\alpha} = \sigma_2, \Delta_R$, and $\Gamma_L \longrightarrow \Delta_L$, and $[\Delta_L](\sigma_1) = [\Delta_L](\sigma_2)$, and Δ_R is soft if and only if Γ_R is soft.*

Proof. **Part 1** By induction on the context extension $\Gamma_L, y : \sigma, \Gamma_R \longrightarrow \Delta$.

- Case

$$\frac{}{\varnothing \longrightarrow \varnothing}\ \text{CE-Empty}$$

  Impossible case.
- Case

$$\frac{\Gamma \longrightarrow \Delta}{\Gamma, x : \sigma \longrightarrow \Delta, x : \sigma}\ \text{CE-Var}$$

  Depending on whether $x = y$, we have two subcases.
  - SubCase $x = y$. Therefore $\Gamma_R = \Delta_R = \varnothing$, and $\Gamma_L = \Gamma, \Delta_L = \Delta$. All goal follows directly.
  - SubCase $x \neq y$. Then we have

$$\frac{\Gamma, y : \sigma, \Gamma'_R \longrightarrow \Delta}{\Gamma, y : \sigma, \Gamma'_R, x : \sigma \longrightarrow \Delta, x : \sigma}\ \text{CE-Var}$$

$\Delta = \Delta_L, y : \sigma, \Delta'_R$	By induction hypothesis
$\Gamma_L \longrightarrow \Delta_L$	By induction hypothesis
$\Gamma'_R, x : \sigma$	is not soft
$\Delta'_R, x : \sigma$	is not soft

- Case

$$\frac{\Gamma_L, y : \sigma, \Gamma'_R \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta}{\Gamma_L, y : \sigma, \Gamma'_R, \widehat{\alpha} \longrightarrow \Delta, \widehat{\alpha}}\ \text{CE-EVar}$$

$\Delta = \Delta_L, y : \sigma, \Delta'_R$	By induction hypothesis
$\Gamma_L \longrightarrow \Delta_L$	By induction hypothesis
Γ'_R is soft iff Δ'_R is soft	By induction hypothesis
$\Gamma'_R, \widehat{\alpha}$ is soft iff $\Delta'_R, \widehat{\alpha}$ is soft	Follows directly

– Case

$$\frac{\Gamma_L, y : \sigma, \Gamma'_R \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta \qquad [\Delta](\tau_1) = [\Delta](\tau_2)}{\Gamma_L, y : \sigma, \Gamma'_R, \widehat{\alpha} = \tau_1 \longrightarrow \Delta, \widehat{\alpha} = \tau_2}\ \textsc{CE-SolvedEVar}$$

$\Delta = \Delta_L, y : \sigma, \Delta'_R$	By induction hypothesis
$\Gamma_L \longrightarrow \Delta_L$	By induction hypothesis
Γ'_R is soft iff Δ'_R is soft	By induction hypothesis
$\Gamma'_R, \widehat{\alpha} = \tau$ is soft iff $\Delta'_R, \widehat{\alpha} = \tau$ is soft	Follows directly

– Case

$$\frac{\Gamma_L, y : \sigma, \Gamma'_R \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta \qquad \Delta \vdash \tau}{\Gamma_L, y : \sigma, \Gamma'_R, \widehat{\alpha} \longrightarrow \Delta, \widehat{\alpha} = \tau}\ \textsc{CE-Solve}$$

$\Delta = \Delta_L, y : \sigma, \Delta'_R$	By induction hypothesis
$\Gamma_L \longrightarrow \Delta_L$	By induction hypothesis
Γ'_R is soft iff Δ'_R is soft	By induction hypothesis
$\Gamma'_R, \widehat{\alpha}$ is soft iff $\Delta'_R, \widehat{\alpha} = \tau$ is soft	Follows directly

– Case

$$\frac{\Gamma_L, y : \sigma, \Gamma_R \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta}{\Gamma_L, y : \sigma, \Gamma_R \longrightarrow \Delta, \widehat{\alpha}}\ \textsc{CE-Add}$$

$\Delta = \Delta_L, y : \sigma, \Delta'_R$	By induction hypothesis
$\Gamma_L \longrightarrow \Delta_L$	By induction hypothesis
Γ_R is soft iff Δ'_R is soft	By induction hypothesis
Γ_R is soft iff $\Delta'_R, \widehat{\alpha}$ is soft	Follows directly

– Case

$$\frac{\Gamma_L, y : \sigma, \Gamma_R \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta \qquad \Delta \vdash \tau}{\Gamma_L, y : \sigma, \Gamma_R \longrightarrow \Delta, \widehat{\alpha} = \tau}\ \textsc{CE-AddSolved}$$

$\Delta = \Delta_L, y : \sigma, \Delta'_R$	By induction hypothesis
$\Gamma_L \longrightarrow \Delta_L$	By induction hypothesis
Γ_R is soft iff Δ'_R is soft	By induction hypothesis
Γ_R is soft iff $\Delta'_R, \widehat{\alpha} = \tau$ is soft	Follows directly

Part 2 Similar to Part 1.
Part 3 Similar to Part 1.

□

Lemma 30 (Extension Weakening). *If* $\Gamma \vdash \tau_1 \Rightarrow \tau_2$, *and* $\Gamma \longrightarrow \Delta$, *and* Δ *ctx* , *then* $\Delta \vdash \tau_1 \Rightarrow [\Delta](\tau_2)$.

Proof. By induction on the typing derivation.

– Case

$$\frac{\Gamma \ ctx}{\Gamma \vdash \star \ \Rightarrow \ \star} \text{ A-Ax}$$

Follows directly by A-Ax.

– Case

$$\frac{\Gamma \ ctx \qquad x : \sigma \in \Gamma}{\Gamma \vdash x \ \Rightarrow \ [\Gamma](\sigma)} \text{ A-Var}$$

$x : \sigma \in \Gamma$	Given
$x : \sigma \in \Delta$	By Lemma 29
$\Delta \vdash x \ \Rightarrow \ [\Delta](\sigma)$	By A-Var
$\tau_2 = [\Gamma](\sigma)$	Given
$[\Delta](\sigma)$	
$= [\Delta]([\Gamma](\sigma))$	By Lemma 24
$= [\Delta](\tau_2)$	Substitute above equality

– Case

$$\frac{\Gamma \ ctx \qquad \widehat{\alpha} \in \Gamma}{\Gamma \vdash \widehat{\alpha} \ \Rightarrow \ \star} \text{ A-EVar}$$

By Lemma 29, we have either $\widehat{\alpha}$ or $\widehat{\alpha} = \tau$ in Δ. Then by rule A-EVar or A-SolvedEVar, we have $\Delta \vdash \widehat{\alpha} \ \Rightarrow \ \star$ directly.

– Case

$$\frac{\Gamma \ ctx \qquad \widehat{\alpha} = \tau \in \Gamma}{\Gamma \vdash \widehat{\alpha} \ \Rightarrow \ \star} \text{ A-SolvedEVar}$$

By Lemma 29, and rule A-SolveEVar, we have $\Delta \vdash \widehat{\alpha} \ \Rightarrow \ \star$ directly.

– Case

$$\frac{\Gamma \vdash \sigma_1 \ \Rightarrow \ \star \qquad \Gamma, x : \sigma_1 \vdash e \ \Rightarrow \ \sigma_2}{\Gamma \vdash \lambda x : \sigma_1.\ e \ \Rightarrow \ \Pi x : [\Gamma](\sigma_1).\ \sigma_2} \text{ A-LamAnn}$$

$\Delta \vdash \sigma_1 \ \Rightarrow \ \star$	By induction hypothesis
$\Gamma, x : \sigma_1 \longrightarrow \Delta, x : \sigma_1$	By CE-Var
$\Delta, x : \sigma_1 \vdash e \ \Rightarrow \ [\Delta, x : \sigma_1](\sigma_2)$	By induction hypothesis
$\Delta, x : \sigma_1 \vdash e \ \Rightarrow \ [\Delta](\sigma_2)$	By definition of context substitution
$\Delta \vdash \lambda x : \sigma_1.\ e \ \Rightarrow \ \Pi x : [\Delta](\sigma_1).\ [\Delta](\sigma_2)$	By A-LamAnn
$[\Delta](\sigma_1) = [\Delta]([\Gamma](\sigma_1))$	By Lemma 24
$\Delta \vdash \lambda x : \sigma_1.\ e \ \Rightarrow \ \Pi x : [\Delta]([\Gamma](\sigma_1)).\ [\Delta](\sigma_2)$	Substitute above equality
$\Delta \vdash \lambda x : \sigma_1.\ e \ \Rightarrow \ [\Delta](\Pi x : [\Gamma](\sigma_1).\ \sigma_2)$	Follows directly

– Case

$$\frac{\Gamma \vdash \sigma_1 \;\Rightarrow\; \star \qquad \Gamma, x : \sigma_1 \vdash \sigma_2 \;\Rightarrow\; \star}{\Gamma \vdash \Pi x : \sigma_1.\ \sigma_2 \;\Rightarrow\; \star}\ \text{A-Pi}$$

$\Delta \vdash \sigma_1 \;\Rightarrow\; \star$	By induction hypothesis
$\Gamma, x : \sigma_1 \longrightarrow \Delta, x : \sigma_1$	By CE-Var
$\Delta, x : \sigma_1 \vdash \sigma_2 \;\Rightarrow\; \star$	By induction hypothesis
$\Delta \vdash \Pi x : \sigma_1.\ \sigma_2 \;\Rightarrow\; \star$	By A-Pi

– Case

$$\frac{\Gamma \vdash e_1 \;\Rightarrow\; \Pi x : \sigma_1.\ \sigma_2 \qquad \Gamma \vdash e_2 \;\Rightarrow\; \sigma_1}{\Gamma \vdash e_1\ e_2 \;\Rightarrow\; \sigma_2[x \mapsto [\Gamma](e_1)]}\ \text{A-App}$$

$\Delta \vdash e_1 \;\Rightarrow\; [\Delta](\Pi x : \sigma_1.\ \sigma_2)$	By induction hypothesis
$\Delta \vdash e_1 \;\Rightarrow\; \Pi x : [\Delta](\sigma_1).\ [\Delta](\sigma_2)$	Follows directly
$\Delta \vdash e_2 \;\Rightarrow\; [\Delta](\sigma_1)$	By induction hypothesis
$\Delta \vdash e_1\ e_2 \;\Rightarrow\; ([\Delta](\sigma_2))[x \mapsto [\Delta](e_1)]$	By A-App
$([\Delta](\sigma_2))[x \mapsto [\Delta](e_1)]$	
$= ([\Delta](\sigma_2))[x \mapsto [\Delta]([\Gamma](e_1))]$	By Lemma 24
$= [\Delta](\sigma_2[x \mapsto [\Gamma](e_1)])$	Property of substitution

– Case

$$\frac{\Gamma \vdash e \;\Rightarrow\; \sigma_1 \qquad \sigma_1 \hookrightarrow \sigma_2}{\Gamma \vdash \mathsf{cast}_{\downarrow}\ e \;\Rightarrow\; \sigma_2}\ \text{A-CastDn}$$

$\Delta \vdash e \;\Rightarrow\; [\Delta](\sigma_1)$	By induction hypothesis
$[\Delta](\sigma_1) \hookrightarrow [\Delta](\sigma_2)$	By Lemma 9
$\Delta \vdash \mathsf{cast}_{\downarrow}\ e \;\Rightarrow\; [\Delta](\sigma_2)$	By A-CastDn

– Case

$$\frac{\Gamma \vdash e \;\Rightarrow\; \sigma_2 \qquad [\Gamma](\sigma_1) \hookrightarrow \sigma_2}{\Gamma \vdash \mathsf{cast}_{\uparrow}\ e \;\Rightarrow\; [\Gamma](\sigma_1)}\ \text{A-CastUp}$$

$\Delta \vdash e \;\Rightarrow\; [\Delta](\sigma_2)$	By induction hypothesis
$[\Delta](\sigma_1)$	
$= [\Delta]([\Gamma](\sigma_1))$	By Lemma 24
$\hookrightarrow [\Delta](\sigma_2)$	By Lemma 9
$\Delta \vdash \mathsf{cast}_{\uparrow}\ e \;\Rightarrow\; [\Delta](\sigma_1)$	By A-CastUp

□

Corollary 3 (Extension Weakening Well Formedness). *If $\Gamma \vdash \tau$, and $\Gamma \longrightarrow \Delta$, and Δ ctx , then $\Delta \vdash \tau$.*

Proof. Follows directly by Lemma 30 since $[\Delta](\star) = \star$. □

Lemma 31 (Extension Weakening Well Scopedness). *If $\Gamma \vDash \tau$, and $\Gamma \longrightarrow \Delta$, then $\Delta \vDash \tau$.*

Proof. By a straightforward induction on the context extension. □

Lemma 32 (Context Extension Preserves Context Well Formedness). *If Γ ctx , and $\Gamma \longrightarrow \Delta$, then Δ ctx .*

Proof. By induction on the context extension.

– Case

$$\frac{}{\varnothing \longrightarrow \varnothing}\ \textsc{CE-Empty}$$

Here we have $\Gamma = \Delta$, so our goal is given.

– Case

$$\frac{\Gamma \longrightarrow \Delta}{\Gamma, x : \sigma \longrightarrow \Delta, x : \sigma}\ \textsc{CE-Var}$$

Γ *ctx*	Given
$\Gamma \vdash \sigma$	Given
$x \notin \Gamma$	By hypothesis
Δ *ctx*	By induction hypothesis
$\Delta \vdash \sigma$	By Corollary 3
$x \notin \Delta$	Context extension add no variables
$\Delta, x : \sigma$ *ctx*	By AC-Var

– Case

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta}{\Gamma, \widehat{\alpha} \longrightarrow \Delta, \widehat{\alpha}}\ \textsc{CE-EVar}$$

Γ *ctx*	Given
Δ *ctx*	By induction hypothesis
$\widehat{\alpha} \notin \Delta$	Given
$\Delta, \widehat{\alpha}$ *ctx*	By AC-EVar

– Case

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta \qquad [\Delta](\tau_1) = [\Delta](\tau_2)}{\Gamma, \widehat{\alpha} = \tau_1 \longrightarrow \Delta, \widehat{\alpha} = \tau_2}\ \textsc{CE-SolvedEVar}$$

Γ *ctx*	Given
$\Gamma \vdash \tau_1$	Given
Δ *ctx*	By induction hypothesis

$\Delta \vdash \tau_1$ By Corollary 3
$\Delta \vdash [\Delta](\tau_1)$ By Lemma 13
$\Delta \vdash [\Delta](\tau_2)$ Known
$\Delta \vdash \tau_2$ By Lemma 14
$\Delta, \widehat{\alpha} = \tau$ By AC-SolvedEVar

– Case

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta \qquad \Delta \vdash \tau}{\Gamma, \widehat{\alpha} \longrightarrow \Delta, \widehat{\alpha} = \tau}\ \text{CE-Solve}$$

Γ *ctx* Given
Δ *ctx* By induction hypothesis
$\Delta, \widehat{\alpha} = \tau$ By AC-SolvedEVar

– Case

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta}{\Gamma \longrightarrow \Delta, \widehat{\alpha}}\ \text{CE-Add}$$

Γ *ctx* Given
Δ *ctx* By induction hypothesis
$\Delta, \widehat{\alpha}$ By AC-EVar

– Case

$$\frac{\Gamma \longrightarrow \Delta \qquad \widehat{\alpha} \notin \Delta \qquad \Delta \vdash \tau}{\Gamma \longrightarrow \Delta, \widehat{\alpha} = \tau}\ \text{CE-AddSolved}$$

Γ *ctx* Given
Δ *ctx* By induction hypothesis
$\Delta, \widehat{\alpha} = \tau$ By AC-EVar

□

Notice since this lemma holds, many preconditions in previous lemmas are hold automatically. For example, Lemma 30 requests Δ is a well-formed context, which is automatically satisfied given Γ is well-formed and this lemma. Therefore, while applying those kind of lemmas, we will ignore this kind of "self-hold" preconditions.

Furthermore, from now on, we will sometimes implicitly assume the contexts we mentioned are well-formed. Since we have already dealt with the dependency between well-formedness and typing through previous lemmas.

Lemma 33 (Solution Admissibility for Extension). *If* $\Gamma_L, \widehat{\alpha}, \Gamma_R$ *ctx , and* $\Gamma_L \vdash \tau$*, then* $\Gamma_L, \widehat{\alpha}, \Gamma_R \longrightarrow \Gamma_L, \widehat{\alpha} = \tau, \Gamma_R$.

Proof. By induction on Γ_R.

– Case $\varnothing$. Follows directly by Lemma 26 and CE-Solve.

– Case $\Gamma_R = \Gamma'_R, x : \sigma$

$\Gamma_L, \widehat{\alpha}, \Gamma'_R \longrightarrow \Gamma_L, \widehat{\alpha} = \tau, \Gamma'_R$ By induction hypothesis
$\Gamma_L, \widehat{\alpha}, \Gamma'_R, x : \sigma \longrightarrow \Gamma_L, \widehat{\alpha} = \tau, \Gamma'_R, x : \sigma$ By CE-Var

– Case $\Gamma_R = \Gamma'_R, \widehat{\beta}$

$\Gamma_L, \widehat{\alpha}, \Gamma'_R \longrightarrow \Gamma_L, \widehat{\alpha} = \tau, \Gamma'_R$ By induction hypothesis
$\Gamma_L, \widehat{\alpha}, \Gamma'_R, \widehat{\beta} \longrightarrow \Gamma_L, \widehat{\alpha} = \tau, \Gamma'_R, \widehat{\beta}$ By CE-EVar

– Case $\Gamma_R = \Gamma'_R, \widehat{\beta} = \sigma$

$\Gamma_L, \widehat{\alpha}, \Gamma'_R \longrightarrow \Gamma_L, \widehat{\alpha} = \tau, \Gamma'_R$ By induction hypothesis
$\Gamma_L, \widehat{\alpha}, \Gamma'_R \vdash \sigma \;\Rightarrow\; \star$ Given
$\Gamma_L, \widehat{\alpha} = \tau, \Gamma'_R \vdash \sigma \;\Rightarrow\; \star$ By Lemma 30
$\Gamma_L, \widehat{\alpha}, \Gamma'_R, \widehat{\beta} = \sigma \longrightarrow \Gamma_L, \widehat{\alpha} = \tau, \Gamma'_R, \widehat{\beta} = \sigma$ By CE-SolvedEVar

□

Lemma 34 (Unsolved Variable Addition for Extension). *If Γ_L, Γ_R ctx , and $\widehat{\alpha} \notin \Gamma_L, \Gamma_R$, then $\Gamma_L, \Gamma_R \longrightarrow \Gamma_L, \widehat{\alpha}, \Gamma_R$.*

Proof. By induction on Γ_R.

– Case $\varnothing$.

$\Gamma_L \longrightarrow \Gamma_L$ By Lemma 26
$\Gamma_L \longrightarrow \Gamma_L, \widehat{\alpha}$ By CE-Add

– Case $\Gamma_R = \Gamma'_R, x : \sigma$

$\Gamma_L, \Gamma'_R \longrightarrow \Gamma_L, \widehat{\alpha}, \Gamma'_R$ By induction hypothesis
$\Gamma_L, \Gamma'_R, x : \sigma \longrightarrow \Gamma_L, \widehat{\alpha}, \Gamma'_R, x : \sigma$ By CE-Var

– Case $\Gamma_R = \Gamma'_R, \widehat{\beta}$

$\Gamma_L, \Gamma'_R \longrightarrow \Gamma_L, \widehat{\alpha}, \Gamma'_R$ By induction hypothesis
$\Gamma_L, \Gamma'_R, \widehat{\beta} \longrightarrow \Gamma_L, \widehat{\alpha}, \Gamma'_R, \widehat{\beta}$ By CE-EVar

– Case $\Gamma_R = \Gamma'_R, \widehat{\beta} = \sigma$

$\Gamma_L, \Gamma'_R \longrightarrow \Gamma_L, \widehat{\alpha}, \Gamma'_R$ By induction hypothesis
$\Gamma_L, \Gamma'_R \vdash \sigma \;\Rightarrow\; \star$ Given
$\Gamma_L, \widehat{\alpha}, \Gamma'_R \vdash \sigma \;\Rightarrow\; \star$ By Lemma 30
$\Gamma_L, \Gamma'_R, \widehat{\beta} = \sigma \longrightarrow \Gamma_L, \widehat{\alpha}, \Gamma'_R, \widehat{\beta} = \sigma$ By CE-SolvedEVar

□

Lemma 35 (Solved Variable Addition for Extension). *If Γ_L, Γ_R ctx , and $\widehat{\alpha} \notin \Gamma_L, \Gamma_R$, and $\Gamma_L \vdash \tau$, then $\Gamma_L, \Gamma_R \longrightarrow \Gamma_L, \widehat{\alpha} = \tau, \Gamma_R$.*

Proof.

$\Gamma_L, \Gamma_R \longrightarrow \Gamma_L, \widehat{\alpha}, \Gamma_R$	By Lemma 34
$\Gamma_L, \widehat{\alpha}, \Gamma_R \longrightarrow \Gamma_L, \widehat{\alpha} = \tau, \Gamma_R$	By Lemma 33
$\Gamma_L, \Gamma_R \longrightarrow \Gamma_L, \widehat{\alpha} = \tau, \Gamma_R$	By Lemma 27

□

Lemma 36 (Parallel Admissibility). *If $\Gamma_L \longrightarrow \Delta_L$, and $\Gamma_L, \Gamma_R \longrightarrow \Delta_L, \Delta_R$, and Γ_R has no variable overlapped with Δ_L, and $\widehat{\alpha} \notin \Gamma_L, \Gamma_R, \Delta_L, \Delta_R$, then:*

- *$\Gamma_L, \widehat{\alpha}, \Gamma_R \longrightarrow \Delta_L, \widehat{\alpha}, \Delta_R$.*
- *If $\Delta_L \vdash \tau$, then $\Gamma_L, \widehat{\alpha}, \Gamma_R \longrightarrow \Delta_L, \widehat{\alpha} = \tau, \Delta_R$.*
- *If $\Gamma_L \vdash \tau$, and $\Gamma_L \vdash \tau_1$, and $\Delta_L \vdash \tau_2$, and $[\Delta_L](\tau_1) = [\Delta_L](\tau_2)$, then $\Gamma_L, \widehat{\alpha} = \tau_1, \Gamma_R \longrightarrow \Delta_L, \widehat{\alpha} = \tau_2, \Delta_R$.*
- *If $\Gamma_L \vdash \tau$, then $\Gamma_L, x : \tau, \Gamma_R \longrightarrow \Delta_L, x : \tau, \Delta_R$.*

Proof. **Part 1** By induction on Δ_R.

- Case $\Delta_R = \varnothing$. Since Γ_R has no variable overlapped with Δ_L, we have $\Gamma_R = \varnothing$. Therefore, $\Gamma_L \longrightarrow \Delta_L$. By applying CE-EVar we are done.
- Case $\Delta_R = \Delta'_R, \widehat{\beta}$. We have $\Gamma_L, \Gamma_R \longrightarrow \Delta_L, \Delta'_R, \widehat{\beta}$. By inversion, we have two subcases.
  - SubCase $\Gamma_R = \Gamma'_R, \widehat{\beta}$, and $\Gamma_L, \Gamma'_R \longrightarrow \Delta_L, \Delta'_R$. Notice here we assume Γ_R is a non-empty context. Since if it is empty, then $\widehat{\beta} \in \Gamma_L$. Then according to Lemma 20, we have $\widehat{\beta} \in \Delta_L$, which should not be right because we implicitly assume Δ_L, Δ_R is well-formed. By induction hypothesis, we have $\Gamma_L, \widehat{\alpha}, \Gamma'_R \longrightarrow \Delta_L, \widehat{\alpha}, \Delta'_R$. Then by CE-EVar we have $\Gamma_L, \widehat{\alpha}, \Gamma_R \longrightarrow \Delta_L, \widehat{\alpha}, \Delta_R$.
  - SubCase $\Gamma_L, \Gamma_R \longrightarrow \Delta_L, \Delta_R$. Then by induction hypothesis we have $\Gamma_L, \widehat{\alpha}, \Gamma_R \longrightarrow \Delta_L, \widehat{\alpha}, \Delta_R$. By CE-Add we are done.
- All rest cases are similar as last case.

Part 2 All rest parts are similar as Part 1.

□

Lemma 37 (Parallel Extension Solution). *If $\Gamma_L, \widehat{\alpha}, \Gamma_R \longrightarrow \Delta_L, \widehat{\alpha} = \tau_2, \Delta_R$, and $\Gamma_L \vdash \tau_1$, and $[\Delta_L](\tau_1) = [\Delta_L](\tau_2)$, then $\Gamma_L, \widehat{\alpha} = \tau_1, \Gamma_R \longrightarrow \Delta_L, \widehat{\alpha} = \tau_2, \Delta_R$.*

Proof. By induction on Δ_R.

Similar to the proof of Lemma 36. □

Lemma 38 (Stability of Complete Contexts). *If $\Gamma \longrightarrow \Omega$, then $[\Omega](\Omega) = [\Omega](\Gamma)$.*

Proof. By induction on the derivation $\Gamma \longrightarrow \Omega$.

– Case

$$\frac{}{\varnothing \longrightarrow \varnothing}\ \textsc{CE-Empty}$$

Holds trivially.

– Case

$$\frac{\Gamma' \longrightarrow \Omega'}{\Gamma', x : \sigma \longrightarrow \Omega', x : \sigma}\ \textsc{CE-Var}$$

$[\Omega](\Omega)$
$= [\Omega'](\Omega'), x : \sigma$ By definition of context application
$= [\Omega'](\Gamma'), x : \sigma$ By induction hypothesis
$= [\Omega](\Gamma)$ By definition of context application

– Case

$$\frac{\Gamma' \longrightarrow \Omega' \qquad \widehat{\alpha} \notin \Omega'}{\Gamma', \widehat{\alpha} \longrightarrow \Omega', \widehat{\alpha}}\ \textsc{CE-EVar}$$

Similar as Case CE-Var.

– Case

$$\frac{\Gamma' \longrightarrow \Omega' \qquad \widehat{\alpha} \notin \Omega' \qquad [\Omega'](\tau_1) = [\Omega'](\tau_2)}{\Gamma', \widehat{\alpha} = \tau_1 \longrightarrow \Omega', \widehat{\alpha} = \tau_2}\ \textsc{CE-SolvedEVar}$$

Similar as Case CE-Var.

– Case

$$\frac{\Gamma' \longrightarrow \Omega' \qquad \widehat{\alpha} \notin \Omega' \qquad \Omega' \vdash \tau}{\Gamma', \widehat{\alpha} \longrightarrow \Omega', \widehat{\alpha} = \tau}\ \textsc{CE-Solve}$$

$[\Omega](\Omega)$
$= [\Omega'](\Omega')$ By definition of context application
$= [\Omega'](\Gamma')$ By induction hypothesis
$= [\Omega](\Gamma)$ By definition of context application

– Case

$$\frac{\Gamma' \longrightarrow \Omega' \qquad \widehat{\alpha} \notin \Omega'}{\Gamma' \longrightarrow \Omega', \widehat{\alpha}}\ \textsc{CE-Add}$$

Impossible since Ω is a complete context.

– Case

$$\frac{\Gamma' \longrightarrow \Omega' \qquad \widehat{\alpha} \notin \Omega' \qquad \Omega' \vdash \tau}{\Gamma' \longrightarrow \Omega', \widehat{\alpha} = \tau}\ \textsc{CE-AddSolved}$$

$[\Omega](\Omega)$
$= [\Omega'](\Omega')$ By definition of context application
$= [\Omega'](\Gamma')$ By induction hypothesis

$= [\Omega](\Gamma)$ By definition of context application

□

Lemma 39 (Finishing Types). *If* $\Omega \vdash \sigma \Rightarrow \tau$, *and* $\Omega \longrightarrow \Omega'$, *then* $[\Omega](\sigma) = [\Omega'](\sigma)$.

Proof.

$[\Omega'](\sigma)$
$= [\Omega']([\Omega](\sigma))$ By Lemma 24
$[\Omega](\sigma)$ contains no existential variables
$[\Omega']([\Omega](\sigma))$
$= [\Omega](\sigma)$ Follows directly

□

Lemma 40 (Finishing Completions). *If* $\Omega \longrightarrow \Omega'$, *then* $[\Omega](\Omega) = [\Omega'](\Omega')$.

Proof. By a straightforward induction on the derivation $\Omega \longrightarrow \Omega'$.
The case analysis is like Lemma 38. □

Lemma 41 (Confluence of Completeness). *If* $\Delta_1 \longrightarrow \Omega$, *and* $\Delta_2 \longrightarrow \Omega$, *then* $[\Omega](\Delta_1) = [\Omega](\Delta_2)$.

Proof.

$[\Omega](\Delta_1)$
$= [\Omega](\Omega)$ By Lemma 38
$= [\Omega](\Delta_2)$ By Lemma 38

□

C.4 Properties of Type Sanitization

Lemma 42 (Type Sanitization Extension). *If* $\Gamma \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta$, *and* $\Gamma \vdash \tau_1 \Rightarrow \sigma$, *then* $\Gamma \longrightarrow \Theta$.

Proof. By induction on type sanitization.

– Case

$$\frac{}{\Gamma[\widehat{\alpha}][\widehat{\beta}] \vdash \widehat{\beta} \longrightarrow \widehat{\alpha}_1 \dashv \Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}_1]}\ \textsc{i-EVarAfter}$$

$\Gamma[\widehat{\alpha}][\widehat{\beta}] \longrightarrow \Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta}]$ By Lemma 34
$\Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta}] \longrightarrow \Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}_1]$ By Lemma 33
$\Gamma[\widehat{\alpha}][\widehat{\beta}] \longrightarrow \Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}_1]$ By Lemma 27

– Case

$$\frac{}{\Gamma[\widehat{\beta}][\widehat{\alpha}] \vdash \widehat{\beta} \longrightarrow \widehat{\beta} \dashv \Gamma[\widehat{\beta}][\widehat{\alpha}]}\ \text{I-EVarBefore}$$

Follows directly by Lemma 26.

– Case

$$\frac{}{\Gamma \vdash x \longrightarrow x \dashv \Gamma}\ \text{I-Var}$$

Follows directly by Lemma 26.

– Case

$$\frac{}{\Gamma \vdash \star \longrightarrow \star \dashv \Gamma}\ \text{I-Star}$$

Follows directly by Lemma 26.

– Case

$$\frac{\Gamma \vdash e_1 \longrightarrow e_3 \dashv \Theta_1 \qquad \Theta_1 \vdash [\Theta_1](e_2) \longrightarrow e_4 \dashv \Theta}{\Gamma \vdash e_1\ e_2 \longrightarrow e_3\ e_4 \dashv \Theta}\ \text{I-App}$$

$\Gamma \longrightarrow \Theta_1$	By induction hypothesis
$\Gamma \vdash e_1\ e_2 \Rightarrow \sigma$	Given
$\Gamma \vdash e_2 \Rightarrow \sigma'$	By inversion
$\Theta_1 \vdash e_2 \Rightarrow [\Theta_1](\sigma')$	By Lemma 30
$\Theta_1 \vdash [\Theta_1](e_2) \Rightarrow \sigma'$	By Lemma 13
$\Theta_1 \longrightarrow \Theta$	By induction hypothesis
$\Gamma \longrightarrow \Theta$	By Lemma 27

– Case

$$\frac{\Gamma \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta_1 \qquad \Theta_1, x : \tau_1 \vdash [\Theta_1](e_1) \longrightarrow e_2 \dashv \Theta, x : \tau_1}{\Gamma \vdash \lambda x : \tau_1.\ e_1 \longrightarrow \lambda x : \tau_2.\ e_2 \dashv \Theta}\ \text{I-LamAnn}$$

$\Gamma \longrightarrow \Theta_1$	By induction hypothesis
$\Gamma \vdash \lambda x : \tau_1.\ e_1 \Rightarrow \sigma$	Given
$\Gamma, x : \tau_1 \vdash e_1 \Rightarrow \sigma'$	By inversion
$\Gamma, x : \tau_1 \longrightarrow \Theta_1, x : \tau_1$	By CE-Var
$\Theta_1, x : \tau_1 \vdash e_1 \Rightarrow [\Theta_1, x : \tau_1](\sigma')$	By Lemma 30
$\Theta_1, x : \tau_1 \vdash [\Theta_1, x : \tau_1](e_1) \Rightarrow [\Theta_1, x : \tau_1](\sigma')$	By Lemma 13
$\Theta_1, x : \tau_1 \vdash [\Theta_1](e_1) \Rightarrow [\Theta_1](\sigma')$	By definition of context application
$\Theta_1, x : \tau_1 \longrightarrow \Theta, x : \tau_1$	By induction hypothesis
$\Theta_1 \longrightarrow \Theta$	By inversion
$\Gamma \longrightarrow \Theta$	By Lemma 27

– Case

$$\frac{\Gamma \vdash \tau_1 \longrightarrow \tau_3 \dashv \Theta_1 \qquad \Theta_1, x : \tau_1 \vdash [\Theta_1](\tau_2) \longrightarrow \tau_4 \dashv \Theta, x : \tau_1}{\Gamma \vdash \Pi x : \tau_1.\ \tau_2 \longrightarrow \Pi x : \tau_3.\ \tau_4 \dashv \Theta} \text{ I-Pi}$$

Similar as Case I-LamAnn.

– Case

$$\frac{\Gamma \vdash e_1 \longrightarrow e_2 \dashv \Theta}{\Gamma \vdash \mathsf{cast}_{\downarrow}\ e_1 \ \simeq\ \mathsf{cast}_{\downarrow}\ e_2 \dashv \Theta} \text{ I-CastDn}$$

Follows directly from induction hypothesis.

– Case

$$\frac{\Gamma \vdash e_1 \longrightarrow e_2 \dashv \Theta}{\Gamma \vdash \mathsf{cast}_{\uparrow}\ e_1 \ \simeq\ \mathsf{cast}_{\uparrow}\ e_2 \dashv \Theta} \text{ I-CastUp}$$

Follows directly from induction hypothesis.

□

Lemma 43 (Type Sanitization Equivalence). *If* $\Gamma \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta$, *and* $\Gamma \vdash \tau_1 \ \Rightarrow\ \sigma$, *then* $[\Theta](\tau_1) = [\Theta](\tau_2)$.

Proof. By induction on type sanitization.

– Case

$$\frac{}{\Gamma[\widehat{\alpha}][\widehat{\beta}] \vdash \widehat{\beta} \longrightarrow \widehat{\alpha}_1 \dashv \Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}_1]} \text{ I-EVarAfter}$$

$[\Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}_1]](\widehat{\beta}) = \widehat{\alpha}_1$ By definition of context substitution
$[\Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}_1]](\widehat{\alpha}_1) = \widehat{\alpha}_1$ By definition of context substitution

– Case

$$\frac{}{\Gamma[\widehat{\beta}][\widehat{\alpha}] \vdash \widehat{\beta} \longrightarrow \widehat{\beta} \dashv \Gamma[\widehat{\beta}][\widehat{\alpha}]} \text{ I-EVarBefore}$$

Follows trivially.

– Case

$$\frac{}{\Gamma \vdash x \longrightarrow x \dashv \Gamma} \text{ I-Var}$$

Follows trivially.

– Case

$$\frac{}{\Gamma \vdash \star \longrightarrow \star \dashv \Gamma} \text{ I-Star}$$

Follows trivially.

– Case

$$\frac{\Gamma \vdash e_1 \longrightarrow e_3 \dashv \Theta_1 \qquad \Theta_1 \vdash [\Theta_1](e_2) \longrightarrow e_4 \dashv \Theta}{\Gamma \vdash e_1\ e_2 \longrightarrow e_3\ e_4 \dashv \Theta} \text{ I-App}$$

$[\Theta_1](e_1) = [\Theta_1](e_3)$ By induction hypothesis

$\Gamma \vdash e_1\ e_2 \Rightarrow \sigma$	Given
$\Gamma \vdash e_1 \Rightarrow \sigma'$	By inversion
$\Gamma \vdash e_2 \Rightarrow \sigma''$	By inversion
$\Gamma \longrightarrow \Theta_1$	By Lemma 42
$\Theta_1 \vdash e_2 \Rightarrow [\Theta_1](\sigma'')$	By Lemma 30
$\Theta_1 \vdash [\Theta_1](e_2) \Rightarrow [\Theta_1](\sigma'')$	By Lemma 13
$\Theta_1 \longrightarrow \Theta$	By Lemma 42
$[\Theta]([\Theta_1](e_2)) = [\Theta](e_4)$	By induction hypothesis
$[\Theta](e_2) = [\Theta](e_4)$	By Lemma 24
$[\Theta](e_3)$	
$= [\Theta]([\Theta_1](e_3))$	By Lemma 24
$= [\Theta]([\Theta_1](e_1))$	By above equalities
$= [\Theta](e_1)$	By Lemma 24

– Case

$$\frac{\Gamma \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta_1 \qquad \Theta_1, x : \tau_1 \vdash [\Theta_1](e_1) \longrightarrow e_2 \dashv \Theta, x : \tau_1}{\Gamma \vdash \lambda x : \tau_1.\ e_1 \longrightarrow \lambda x : \tau_2.\ e_2 \dashv \Theta}\ \textsc{I-LamAnn}$$

$\Gamma \vdash \lambda x : \tau_1.\ e_1 \Rightarrow \sigma$	Given
$\Gamma \vdash \tau_1 \Rightarrow \star$	By inversion
$\Gamma \longrightarrow \Theta_1$	By Lemma 42
$[\Theta_1](\tau_1) = [\Theta_1](\tau_2)$	By induction hypothesis
$\Gamma, x : \tau_1 \longrightarrow \Theta_1, x : \tau_1$	By CE-Var
$\Gamma, x : \tau_1 \vdash e_1 \Rightarrow \sigma'$	By inversion
$\Theta_1, x : \tau_1 \vdash e_1 \Rightarrow [\Theta_1, x : \tau_1](\sigma')$	By Lemma 30
$\Theta_1, x : \tau_1 \vdash [\Theta_1, x : \tau_1](e_1) \Rightarrow [\Theta_1, x : \tau_1](\sigma')$	By Lemma 13
$\Theta_1, x : \tau_1 \vdash [\Theta_1](e_1) \Rightarrow [\Theta_1](\sigma')$	By definition of context application
$\Theta_1, x : \tau_1 \longrightarrow \Theta, x : \tau_1$	By Lemma 42
$\Theta_1 \longrightarrow \Theta$	By inversion
$[\Theta, x : \tau_1]([\Theta_1](e_1)) = [\Theta, x : \tau_1](e_2)$	By induction hypothesis
$[\Theta]([\Theta_1](e_1)) = [\Theta](e_2)$	By definition of context application
$[\Theta](e_1) = [\Theta](e_2)$	By Lemma 24
$[\Theta](\tau_1) = [\Theta](\tau_2)$	Similarly

– Case

$$\frac{\Gamma \vdash \tau_1 \longrightarrow \tau_3 \dashv \Theta_1 \qquad \Theta_1, x : \tau_1 \vdash [\Theta_1](\tau_2) \longrightarrow \tau_4 \dashv \Theta, x : \tau_1}{\Gamma \vdash \Pi x : \tau_1.\ \tau_2 \longrightarrow \Pi x : \tau_3.\ \tau_4 \dashv \Theta}\ \textsc{I-Pi}$$

Similar as Case I-LamAnn.

– Case

$$\frac{\Gamma \vdash e_1 \longrightarrow e_2 \dashv \Theta}{\Gamma \vdash \mathsf{cast}_{\downarrow}\ e_1 \simeq \mathsf{cast}_{\downarrow}\ e_2 \dashv \Theta}\ \textsc{I-CastDn}$$

Follows directly from induction hypothesis.

– Case

$$\frac{\Gamma \vdash e_1 \longrightarrow e_2 \dashv \Theta}{\Gamma \vdash \mathsf{cast}_{\uparrow}\ e_1 \ \simeq\ \mathsf{cast}_{\uparrow}\ e_2 \dashv \Theta}\ \textsc{I-CastUp}$$

Follows directly from induction hypothesis.

□

Lemma 44 (Type Sanitization Well Formedness). *If* $\Gamma \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta$, *and* $\Gamma \vdash \tau_1 \Rightarrow \sigma$, *then* $\Theta \vdash \tau_2 \Rightarrow [\Theta](\sigma)$.

Proof. By induction on the type sanitization.

– Case

$$\frac{}{\Gamma[\widehat{\alpha}][\widehat{\beta}] \vdash \widehat{\beta} \longrightarrow \widehat{\alpha}_1 \dashv \Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}_1]}\ \textsc{I-EVarAfter}$$

$\tau_1 = \widehat{\beta}, \sigma = \star$	By inversion
$\Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}_1] \vdash \widehat{\alpha}_1 \Rightarrow \star$	Follows directly by A-EVar.

– Case

$$\frac{}{\Gamma[\widehat{\beta}][\widehat{\alpha}] \vdash \widehat{\beta} \longrightarrow \widehat{\beta} \dashv \Gamma[\widehat{\beta}][\widehat{\alpha}]}\ \textsc{I-EVarBefore}$$

Holds trivially.

– Case

$$\frac{}{\Gamma \vdash \star \longrightarrow \star \dashv \Gamma}\ \textsc{I-Star}$$

Holds trivially.

– Case

$$\frac{\Gamma \vdash e_1 \longrightarrow e_3 \dashv \Theta_1 \qquad \Theta_1 \vdash [\Theta_1](e_2) \longrightarrow e_4 \dashv \Theta}{\Gamma \vdash e_1\ e_2 \longrightarrow e_3\ e_4 \dashv \Theta}\ \textsc{I-App}$$

$\Gamma \vdash e_1 \Rightarrow \Pi x : \sigma_1.\ \sigma_2$	By inversion
$\Gamma \vdash e_2 \Rightarrow \sigma_1$	By inversion
$\sigma = \sigma_2[x \mapsto [\Gamma](e_2)]$	By inversion
$\Theta_1 \vdash e_3 \Rightarrow [\Theta_1](\Pi x : \sigma_1.\ \sigma_2)$	By induction hypothesis
$\Gamma \longrightarrow \Theta_1$	By Lemma 42
$\Theta_1 \vdash e_2 \Rightarrow [\Theta_1](\sigma_1)$	By Lemma 30
$\Theta_1 \vdash [\Theta_1](e_2) \Rightarrow [\Theta_1](\sigma_1)$	By Lemma 13
$\Theta \vdash e_4 \Rightarrow [\Theta]([\Theta_1](\sigma_1))$	By induction hypothesis
$\Theta_1 \longrightarrow \Theta$	By Lemma 42
$\Theta \vdash e_4 \Rightarrow [\Theta](\sigma_1)$	By Lemma 24
$\Theta \vdash e_3 \Rightarrow [\Theta]([\Theta_1](\Pi x : \sigma_1.\ \sigma_2))$	By Lemma 30
$\Theta \vdash e_3 \Rightarrow [\Theta](\Pi x : \sigma_1.\ \sigma_2)$	By Lemma 24
$\Theta \vdash e_3\ e_4 \Rightarrow ([\Theta](\sigma_2))[x \mapsto [\Theta](e_4)]$	By A-App

$[\Theta]([\Theta_1](e_2)) = [\Theta](e_4)$	By Lemma 43
$[\Theta](e_2) = [\Theta](e_4)$	By Lemma 24
$\Theta \vdash e_3\ e_4 \Rightarrow ([\Theta](\sigma_2))[x \mapsto [\Theta](e_2)]$	By substituting above equality
$\Gamma \longrightarrow \Theta$	By Lemma 27
$[\Theta](e_2) = [\Theta]([\Gamma](e_2))$	By Lemma 24
$\Theta \vdash e_3\ e_4 \Rightarrow ([\Theta](\sigma_2))[x \mapsto [\Theta]([\Gamma](e_2))]$	By substituting above equality
$\Theta \vdash e_3\ e_4 \Rightarrow [\Theta](\sigma_2[x \mapsto [\Gamma](e_2)])$	By property of substitution

– Case

$$\frac{\Gamma \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta_1 \qquad \Theta_1, x : \tau_1 \vdash [\Theta_1](e_1) \longrightarrow e_2 \dashv \Theta, x : \tau_1}{\Gamma \vdash \lambda x : \tau_1.\ e_1 \longrightarrow \lambda x : \tau_2.\ e_2 \dashv \Theta}\ \textsc{I-LamAnn}$$

$\Gamma \vdash \tau_1 \Rightarrow \star$	By inversion
$\Gamma, x : \tau_1 \vdash e_1 \Rightarrow \sigma_1$	By inversion
$\sigma = \Pi x : [\Gamma](\tau_1).\ \sigma_1$	By inversion
$\Theta_1 \vdash \tau_2 \Rightarrow \star$	By induction hypothesis
$\Gamma \longrightarrow \Theta_1$	By Lemma 42
$\Gamma, x : \tau_1 \longrightarrow \Theta_1, x : \tau_1$	By CE-Var
$\Theta_1, x : \tau_1 \vdash e_1 \Rightarrow [\Theta_1, x : \tau_1](\sigma_1)$	By Lemma 30
$\Theta_1, x : \tau_1 \vdash [\Theta_1, x : \tau_1](e_1) \Rightarrow [\Theta_1, x : \tau_1](\sigma_1)$	By Lemma 13
$\Theta_1, x : \tau_1 \vdash [\Theta_1](e_1) \Rightarrow [\Theta_1](\sigma_1)$	By definition on context application
$\Theta, x : \tau_1 \vdash e_2 \Rightarrow [\Theta, x : \tau_1]([\Theta_1](\sigma_1))$	By induction hypothesis
$\Theta, x : \tau_1 \vdash e_2 \Rightarrow [\Theta]([\Theta_1](\sigma_1))$	By definition of context substitution
$\Theta_1, x : \tau_1 \longrightarrow \Theta, x : \tau_1$	By Lemma 42
$\Theta_1 \longrightarrow \Theta$	By inversion
$\Theta, x : \tau_1 \vdash e_2 \Rightarrow [\Theta](\sigma_1)$	By Lemma 24
$\Theta \vdash \tau_2 \Rightarrow \star$	By Lemma 24
$\Gamma \longrightarrow \Theta$	By Lemma 27
$\Theta \vdash \tau_1 \Rightarrow \star$	By Lemma 30
$\Theta \vdash [\Theta](\tau_1) \Rightarrow \star$	By Lemma 13
$\Theta, x : [\Theta](\tau_1) \vdash e_2 \Rightarrow [\Theta](\sigma_1)$	By Lemma 11
$[\Theta_1](\tau_1) = [\Theta_1](\tau_2)$	By Lemma 43
$[\Theta](\tau_1) = [\Theta](\tau_2)$	By Lemma 24
$\Theta, x : [\Theta](\tau_2) \vdash e_2 \Rightarrow [\Theta](\sigma_1)$	By substituting above equality
$\Theta, x : \tau_2 \vdash e_2 \Rightarrow [\Theta](\sigma_1)$	By Lemma 12
$\Theta \vdash \lambda x : \tau_2.\ e_2 \Rightarrow \Pi x : [\Theta](\tau_2).\ [\Theta](\sigma_1)$	By A-LamAnn
$\Pi x : [\Theta](\tau_2).\ [\Theta](\sigma_1)$	
$= \Pi x : [\Theta]([\Gamma](\tau_2)).\ [\Theta](\sigma_1)$	By Lemma 24
$= [\Theta](\Pi x : [\Gamma](\tau_2).\ \sigma_1)$	By property of substitution

– Case

$$\frac{\Gamma \vdash \tau_1 \longrightarrow \tau_3 \dashv \Theta_1 \qquad \Theta_1, x : \tau_1 \vdash [\Theta_1](\tau_2) \longrightarrow \tau_4 \dashv \Theta, x : \tau_1}{\Gamma \vdash \Pi x : \tau_1.\ \tau_2 \longrightarrow \Pi x : \tau_3.\ \tau_4 \dashv \Theta} \text{ I-Pi}$$

Similar as Case I-LamAnn.

– Case

$$\frac{\Gamma \vdash e_1 \longrightarrow e_2 \dashv \Theta}{\Gamma \vdash \mathsf{cast}_{\downarrow}\ e_1 \ \simeq\ \mathsf{cast}_{\downarrow}\ e_2 \dashv \Theta} \text{ I-CastDn}$$

$\Gamma \vdash e_1 \Rightarrow \sigma_1$	By inversion
$\sigma_1 \hookrightarrow \sigma$	By inversion
$\Theta \vdash e_2 \Rightarrow [\Theta](\sigma_1)$	By induction hypothesis
$[\Theta](\sigma_1) \hookrightarrow [\Theta](\sigma)$	By Lemma 9
$\Theta \vdash \mathsf{cast}_{\downarrow}\ e_2 \Rightarrow [\Theta](\sigma)$	By A-CastDn

– Case

$$\frac{\Gamma \vdash e_1 \longrightarrow e_2 \dashv \Theta}{\Gamma \vdash \mathsf{cast}_{\uparrow}\ e_1 \ \simeq\ \mathsf{cast}_{\uparrow}\ e_2 \dashv \Theta} \text{ I-CastUp}$$

Similar as Case I-CastDn

□

Lemma 45 (Type Sanitization Tail Unchanged). *If v is a binding, and $\Gamma, v \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta$, then $\Theta = \Theta', v$.*

Proof. By a straightforward induction on the type sanitization derivation. □

We use the notation $\Gamma \vdash \tau_1 = \tau_2$ to mean that $\Gamma \vdash \tau_1 \Rightarrow \sigma$, $\Gamma \vdash \tau_2 \Rightarrow \sigma$, and $\tau_1 = \tau_2$.

Lemma 46 (Type Sanitization Completeness). *Given $\Gamma \longrightarrow \Omega$, where $\Gamma = \Gamma_1, \widehat{\alpha}, \Gamma_2, \Gamma_0$, and Γ_0 only contains variables, and $[\Gamma](\sigma_1) = \sigma_1$, and $[\Omega](\tau) = \tau$, and $\widehat{\alpha} \notin FV(\sigma_1)$, and $\Gamma_1, \Gamma_0 \vDash \tau$. If $[\Omega](\Gamma) \vdash \tau = [\Omega](\sigma_1)$, then there are Δ, Ω' such that $\Omega \longrightarrow \Omega'$, and $\Delta \longrightarrow \Omega'$, and $\Gamma[\widehat{\alpha}] \vdash \sigma_1 \longrightarrow \sigma_2 \dashv \Delta$, where $\Delta = \Delta_1, \widehat{\alpha}, \Delta_2, \Gamma_0$, and $\Delta_1, \Gamma_0 \vDash \sigma_2$.*

By induction on the type τ.

We then case analyze the shape of σ_1.

– $\sigma_1 = \widehat{\beta}$

$\widehat{\alpha} \notin FV(\sigma_1)$	Given
$\widehat{\alpha} \neq \widehat{\beta}$	Follows directly
$[\Omega](\widehat{\beta}) = \tau$	Given
$\Omega \vdash \widehat{\beta} \Rightarrow \star$	By A-SolvedEVar
$\Omega \vdash \tau \Rightarrow \star$	By Lemma 13
$[\Gamma](\widehat{\beta}) = \widehat{\beta}$	Given
$\widehat{\beta} \in \mathit{unsolved}(\Gamma)$	Follows directly

According to whether $\widehat{\beta}$ is on the left of $\widehat{\alpha}$ or right, we have two cases. Moreover, $\widehat{\beta}$ cannot be in Γ_0 since it only contains variables.

- SubCase $\Gamma = \Gamma_1, \widehat{\alpha}, \Gamma_3, \widehat{\beta}, \Gamma_4, \Gamma_0$.

$\Gamma_1, \widehat{\alpha}, \Gamma_3, \widehat{\beta}, \Gamma_4, \Gamma_0 \vdash \widehat{\beta} \longrightarrow \widehat{\alpha}_1 \dashv \Gamma_1, \widehat{\alpha}_1, \widehat{\alpha}, \Gamma_3, \widehat{\beta} = \widehat{\alpha}_1, \Gamma_4, \Gamma_0$	By I-EVarAfter
$\Gamma_1, \widehat{\alpha}, \Gamma_3, \widehat{\beta}, \Gamma_4, \Gamma_0 \longrightarrow \Omega$	Given
$\Omega = \Omega_1, \widehat{\alpha} = \tau', \Omega_2, \widehat{\beta} = \tau'', \Omega_3, \Gamma_0, \Omega_4$	By Lemma 29 and Lemma 22
$\Gamma_1 \longrightarrow \Omega_1$	As above
$[\Omega](\widehat{\beta}) = [\Omega_1, \widehat{\alpha} = \tau', \Omega_2](\widehat{\beta})$	By definition of context application
$\Omega_1, \widehat{\alpha} = \tau', \Omega_2 \vdash \widehat{\beta} \Rightarrow \star$	By A-SolvedEVar
$\Omega_1, \widehat{\alpha} = \tau', \Omega_2 \vdash \tau \Rightarrow \star$	By 13
τ has no variable in Γ_0	Follows directly
$\Gamma_1, \Gamma_0 \vDash \tau$	Given
$\Gamma_1 \vDash \tau$	Follows directly
$\Omega_1 \vDash \tau$	By Lemma 31
$\Omega_1, \widehat{\alpha} = \tau', \Omega_2 \vdash \tau \Rightarrow \star$	Known
$\Omega_1 \vdash \tau \Rightarrow \star$	By Lemma 17
$\Omega' = \Omega_1, \widehat{\alpha}_1 = \tau, \widehat{\alpha} = \tau', \Omega_2, \widehat{\beta} = \tau'', \Omega_3, \Gamma_0, \Omega_4$	Choose
$\Gamma_1, \widehat{\alpha}, \Gamma_3, \widehat{\beta}, \Gamma_4, \Gamma_0 \longrightarrow \Omega_1, \widehat{\alpha} = \tau', \Omega_2, \widehat{\beta} = \tau'', \Omega_3, \Gamma_0, \Omega_4$	Known
$\Gamma_1, \widehat{\alpha}_1, \widehat{\alpha}, \Gamma_3, \widehat{\beta}, \Gamma_4, \Gamma_0 \longrightarrow \Omega'$	By Lemma 36
$[\Omega'](\widehat{\beta}) = \tau$	Known
$[\Omega'](\widehat{\alpha}_1) = \tau$	Known
$\Gamma_1, \widehat{\alpha}_1, \widehat{\alpha}, \Gamma_3, \widehat{\beta} = \widehat{\alpha}_1, \Gamma_4, \Gamma_0 \longrightarrow \Omega'$	By Lemma 37
$\Omega \longrightarrow \Omega'$	By Lemma 35

- SubCase $\Gamma = \Gamma_1, \widehat{\beta}, \Gamma_3, \widehat{\alpha}, \Gamma_2, \Gamma_0$.

$\Gamma_1, \widehat{\beta}, \Gamma_3, \widehat{\alpha}, \Gamma_2, \Gamma_0 \vdash \widehat{\beta} \longrightarrow \widehat{\beta} \dashv \Gamma_1, \widehat{\beta}, \Gamma_3, \widehat{\alpha}, \Gamma_2, \Gamma_0$	By I-EVarBefore
$\Gamma \longrightarrow \Omega$	Given
$\Omega' = \Omega$	Choose
$\Omega \longrightarrow \Omega'$	By Lemma 26

– Case $\sigma_1 = x$.

$[\Omega](x) = x$	By definition of context application
$\tau = x$	Given
$\Gamma_1, \Gamma_0 \vDash x$	Given
$\Gamma \vdash x \longrightarrow x \dashv \Gamma$	By I-Var
$\Omega' = \Omega$	Choose
$\Omega \longrightarrow \Omega'$	By Lemma 26

– Case $\sigma_1 = \star$.

$[\Omega](\star) = \star$	By definition of context application
$\tau = \star$	Given
$\Gamma_1, \Gamma_0 \vDash \star$	Given
$\Gamma \vdash \star \longrightarrow \star \dashv \Gamma$	By I-Star
$\Omega' = \Omega$	Choose
$\Omega \longrightarrow \Omega'$	By Lemma 26

– Case $\sigma_1 = e_1\ e_2$.

$[\Omega](e_1\ e_2) = [\Omega](e_1)\ [\Omega](e_2)$	By definition of context application
$\tau = \tau_1\ \tau_2$	By inversion
$[\Omega](\Gamma) \vdash \tau_1 = [\Omega](e_1)$	As above
$[\Omega](\Gamma) \vdash \tau_2 = [\Omega](e_2)$	As above
$\Gamma \vdash e_1 \longrightarrow e_3 \dashv \Theta$	By induction hypothesis
$\Theta = \Theta_1, \widehat{\alpha}, \Theta_2, \Gamma_0$	As above
$\Theta_1, \Gamma_0 \vDash e_3$	As above
$\Theta \longrightarrow \Omega_1$	As above
$\Omega \longrightarrow \Omega_1$	As above
$\Gamma \longrightarrow \Theta$	By Lemma 42
$\Gamma \longrightarrow \Omega_1$	By Lemma 27
$[\Omega](\Omega) \vdash \tau_2 = [\Omega](e_2)$	By Lemma 38
$[\Omega_1](\Omega_1) \vdash \tau_2 = [\Omega](e_2)$	By Lemma 40
$[\Omega_1](\Omega_1) \vdash \tau_2 = [\Omega_1](e_2)$	By Lemma 39
$[\Omega_1](\Theta) \vdash \tau_2 = [\Omega_1](e_2)$	By Lemma 38
$[\Omega_1](\Theta) \vdash \tau_2 = [\Omega_1]([\Theta](e_2))$	By Lemma 24
$\Theta \vdash [\Theta](e_2) \longrightarrow e_4 \dashv \Delta$	By induction hypothesis
$\Delta = \Delta_1, \widehat{\alpha}, \Delta_2, \Gamma_0$	As above
$\Delta \longrightarrow \Omega_2$	As above
$\Omega_1 \longrightarrow \Omega_2$	As above
$\Delta_1, \Gamma_0 \vDash e_4$	As above
$\Theta_1 \longrightarrow \Delta_1$	By Lemma 29
$\Theta_1, \Gamma_0 \longrightarrow \Delta_1, \Gamma_0$	By repeating CE-Var
$\Delta_1, \Gamma_0 \vDash e_3$	By Lemma 31
$\Delta_1, \Gamma_0 \vDash e_3\ e_4$	Follows directly
$\Gamma \vdash e_1\ e_2 \longrightarrow e_3\ e_4 \dashv \Delta$	By I-App
$\Omega' = \Omega_2$	Choose
$\Omega \longrightarrow \Omega_2$	By Lemma 27

– Case $\sigma_1 = \lambda x : \sigma.\ e$.

$[\Omega](\lambda x : \sigma.\ e) = \lambda x : [\Omega](\sigma).\ [\Omega](e)$	By definition of context application
$\tau = \lambda x : \tau_1.\ \tau_2$	By inversion

$[\Omega](\Gamma) \vdash \tau_1 = [\Omega](\sigma)$	As above
$[\Omega](\Gamma), x : \tau_1 \vdash \tau_2 = [\Omega](e)$	As above
$\Gamma \vdash \sigma \longrightarrow \sigma' \dashv \Theta$	By induction hypothesis
$\Theta = \Theta_1, \widehat{\alpha}, \Theta_2, \Gamma$	As above
$\Theta_1, \Gamma_0 \vDash \sigma'$	As above
$\Theta \longrightarrow \Omega_1$	As above
$\Omega \longrightarrow \Omega_1$	As above
$\Gamma \longrightarrow \Theta$	By Lemma 42
$\Gamma \longrightarrow \Omega_1$	By Lemma 27
$\Gamma, x : \tau_1 \longrightarrow \Omega, x : \tau_1$	By rulCE-Var
$\Omega, x : \tau_1 \longrightarrow \Omega_1, x : \tau_1$	By rulCE-Var
$\Theta, x : \tau_1 \longrightarrow \Omega_1, x : \tau_1$	As above
$[\Omega](\Gamma), x : \tau_1 \vdash \tau_2 = [\Omega](e)$	Known
$[\Omega, x : \tau_1](\Gamma, x : \tau_1) \vdash \tau_2 = [\Omega, x : \tau_1](e)$	By definition of context application
$[\Omega_1, x : \tau_1](\Gamma, x : \tau_1) \vdash \tau_2 = [\Omega, x : \tau_1](e)$	By Lemma 40
$[\Omega_1, x : \tau_1](\Theta, x : \tau_1) \vdash \tau_2 = [\Omega, x : \tau_1](e)$	By Lemma 41
$[\Omega_1, x : \tau_1](\Theta, x : \tau_1) \vdash \tau_2 = [\Omega_1, x : \tau_1](e)$	By Lemma 39
$[\Omega_1, x : \tau_1](\Theta, x : \tau_1) \vdash \tau_2 = [\Omega_1, x : \tau_1]([\Theta, x : \tau_1](e))$	By Lemma 24
$[\Omega_1, x : \tau_1](\Theta, x : \tau_1) \vdash \tau_2 = [\Omega_1, x : \tau_1]([\Theta](e))$	By definition of context application
$\Theta, x : \tau_1 \vdash [\Theta](e) \longrightarrow e' \dashv \Delta, x : \tau_1$	By induction hypothesis
$\Delta = \Delta_1, \widehat{\alpha}, \Delta_2, \Gamma_0$	As above
$\Delta_1, \Gamma_0, x : \tau_1 \vDash e'$	As above
$\Delta, x : \tau_1 \longrightarrow \Omega_2$	As above
$\Omega, x : \tau_1 \longrightarrow \Omega_2$	As above
$\Omega_2 = \Omega_3, x : \tau_1, \Omega_4$	By Lemma 29
$\Omega \longrightarrow \Omega_3$	As above
$\Omega' = \Omega_3$	Choose
$\Gamma \vdash \lambda x : \sigma.\ e \longrightarrow \lambda x : \sigma'.\ e' \dashv \Delta$	By I-App
$\Theta, x : \tau_1 \longrightarrow \Delta, x : \tau_1$	By Lemma 42
$\Theta \longrightarrow \Delta$	By inversion
$\Theta_1 \longrightarrow \Delta_1$	By Lemma 29
$\Theta_1, \Gamma_0 \longrightarrow \Delta_1, \Gamma_0$	By repeating CE-Var
$\Delta_1, \Gamma_0 \vDash \sigma'$	By Lemma 31
$\Delta_1, \Gamma_0 \vDash \lambda x : \sigma'.\ e'$	By WS-LamAnn

– Case $\sigma_1 = \Pi x : \sigma_2.\ \sigma_3$. Similar as last case.
– Case $\sigma_1 = \mathsf{cast}_{\downarrow}\ e_1$.

$[\Omega](\mathsf{cast}_{\downarrow}\ e_1) = \mathsf{cast}_{\downarrow}\ [\Omega](e_1)$	By definition of context application
$\tau = \mathsf{cast}_{\downarrow}\ e_2$	By inversion
$[\Omega](\Gamma) \vdash e_1 = [\Omega](e_2)$	As above
$\Gamma \vdash e_2 \longrightarrow e_1' \dashv \Delta$	By induction hypothesis

$\Gamma \vdash \mathsf{cast}_{\downarrow}\ e_2 \longrightarrow \mathsf{cast}_{\downarrow}\ e_1' \dashv \Delta$ By I-CastDn
All rest follows directly from induction hypothesis

– Case $\sigma_1 = \mathsf{cast}_{\downarrow}\ e_1$. Similar as last case.

□

Corollary 4 (Type Sanitization Completeness Pretty). *Given $\Gamma \longrightarrow \Omega$, where $\Gamma = \Gamma_1, \widehat{\alpha}, \Gamma_2$, and $[\Gamma](\sigma_1) = \sigma_1$, and $[\Omega](\tau) = \tau$, and $\widehat{\alpha} \notin FV(\sigma_1)$, and $\Gamma_1 \vDash \tau$. If $[\Omega](\Gamma) \vdash \tau = [\Omega](\sigma_1)$, then there are Δ, Ω' such that $\Omega \longrightarrow \Omega'$, and $\Delta \longrightarrow \Omega'$, and $\Gamma[\widehat{\alpha}] \vdash \sigma_1 \longrightarrow \sigma_2 \dashv \Delta$, where $\Delta = \Delta_1, \widehat{\alpha}, \Delta_2$, and $\Delta_1 \vDash \sigma_2$.*

Proof. Follows directly from Lemma 46 by choosing $\Gamma_0 = \varnothing$. □

Lemma 47 (Type Sanitization Completeness w.r.t. Unification). *Given $\Gamma \longrightarrow \Omega$, where $\Gamma = \Gamma_1, \widehat{\alpha}, \Gamma_2$, and $[\Gamma](\sigma_1) = \sigma_1$, and $\widehat{\alpha} \notin FV(\sigma_1)$.*

– *If $[\Omega](\Gamma) \vdash [\Omega](\widehat{\alpha}) = [\Omega](\sigma_1)$, then there are Δ, Ω' such that $\Omega \longrightarrow \Omega'$, and $\Delta \longrightarrow \Omega'$, and $\Gamma[\widehat{\alpha}] \vdash \widehat{\alpha} \simeq \sigma_1 \dashv \Delta$.*
– *If $[\Omega](\Gamma) \vdash [\Omega](\sigma_1) = [\Omega](\widehat{\alpha})$, then there are Δ, Ω' such that $\Omega \longrightarrow \Omega'$, and $\Delta \longrightarrow \Omega'$, and $\Gamma[\widehat{\alpha}] \vdash \sigma_1 \simeq \widehat{\alpha} \dashv \Delta$.*

Proof. **Part 1**

$[\Omega](\widehat{\alpha}) = \tau$	Assume
$[\Omega](\tau) = \tau$	Since τ contains no existential variables
$\Omega = \Omega_1, \widehat{\alpha} = \tau', \Omega_2$	By Lemma 29
$\Gamma_1 \longrightarrow \Omega_1$	As above
$[\Omega](\widehat{\alpha}) = [\Omega_1](\widehat{\alpha}) = \tau$	By definition of context application
$\Omega_1 \vDash \tau$	Follows directly
Since τ contains no existential variables	
And Γ_1 contains all type variables in Ω_1	
$\Gamma_1 \vDash \tau$	Follows directly
$[\Omega](\Gamma) \vdash \tau = [\Omega](\sigma_1)$	Given
$\Gamma \vdash \sigma_1 \longrightarrow \sigma_2 \dashv \Delta$	By Corollary 47
$\Delta = \Delta_1, \widehat{\alpha}, \Delta_2$	As above
$\Delta_1 \vDash \sigma_2$	As above
$\Delta \longrightarrow \Omega'$	As above
$\Omega \longrightarrow \Omega'$	As above
$\Gamma \vdash_{\sigma} \widehat{\alpha} \simeq \sigma_1 \dashv \Delta_1, \widehat{\alpha} = \sigma_2, \Delta_2$	By U-EVarTy
$[\Omega'](\widehat{\alpha}) = [\Omega](\widehat{\alpha}) = \tau = [\Omega](\sigma_1)$	By Lemma 39
$[\Delta](\sigma_1) = [\Delta](\sigma_2)$	By Lemma 43
$[\Omega'](\sigma_1) = [\Omega'](\sigma_2)$	By Lemma 24
$[\Omega'](\sigma_1) = [\Omega](\sigma_1)$	By Lemma 39
$[\Omega'](\widehat{\alpha}) = [\Omega'](\sigma_2)$	Follows directly
$\Delta_1, \widehat{\alpha} = \sigma_2, \Delta_2 \longrightarrow \Omega'$	By Lemma 37

Part 2 Similar as Part 1.

□

C.5 Properties of Unification

Lemma 48 (Unification Extension).

- *If* $\Gamma \vdash_e e_1 \simeq e_2 \dashv \Theta$, *and* $\Gamma \vdash e_1 \Rightarrow \sigma_1'$, *and* $\Gamma \vdash e_2 \Rightarrow \sigma_2'$, *then* $\Gamma \longrightarrow \Theta$.
- *If* $\Gamma \vdash_\sigma \tau_1 \simeq \tau_2 \dashv \Theta$, *and* $\Gamma \vdash \tau_1 \Rightarrow \star$, *and* $\Gamma \vdash \tau_2 \Rightarrow \star$, *then* $\Gamma \longrightarrow \Theta$.

Proof. By induction on the height of unification derivation. Two parts reply on each other but the size of derivation is decreasing. So the proof can terminate.

Part 1 In Part 1, we regard all $\delta = e$.

- Case

$$\frac{}{\Gamma \vdash_\delta \sigma \simeq \sigma \dashv \Gamma}\ \textsc{U-AEq}$$

  Follows trivially by Lemma 26.

- Case

$$\frac{\Gamma \vdash_e e_1 \simeq e_3 \dashv \Theta_1 \qquad \Theta_1 \vdash_e [\Theta_1](e_2) \simeq [\Theta_1](e_4) \dashv \Theta}{\Gamma \vdash_\delta e_1\ e_2 \simeq e_3\ e_4 \dashv \Theta}\ \textsc{U-App}$$

$\Gamma \vdash e_1\ e_2 \Rightarrow \sigma_1'$	Given
$\Gamma \vdash e_3\ e_4 \Rightarrow \sigma_2'$	Given
$\Gamma \vdash e_1 \Rightarrow \sigma_1''$	By inversion
$\Gamma \vdash e_3 \Rightarrow \sigma_2''$	By inversion
$\Gamma \longrightarrow \Theta_1$	By induction hypothesis
$\Gamma \vdash e_2 \Rightarrow \sigma_1'''$	By inversion
$\Gamma \vdash e_4 \Rightarrow \sigma_2'''$	By inversion
$\Theta_1 \vdash e_2 \Rightarrow [\Theta_1](\sigma_1''')$	By Lemma 30
$\Theta_1 \vdash e_4 \Rightarrow [\Theta_1](\sigma_2''')$	By Lemma 30
$\Theta_1 \vdash [\Theta_1](e_2) \Rightarrow [\Theta_1](\sigma_1''')$	By Lemma 13
$\Theta_1 \vdash [\Theta_1](e_4) \Rightarrow [\Theta_1](\sigma_2''')$	By Lemma 13
$\Theta_1 \longrightarrow \Theta$	By induction hypothesis
$\Gamma \longrightarrow \Theta$	By Lemma 27

- Case

$$\frac{\Gamma \vdash_\sigma \sigma_1 \simeq \sigma_2 \dashv \Theta_1 \qquad \Theta_1, x : \sigma_1 \vdash_e [\Theta_1](e_1) \simeq [\Theta_1](e_2) \dashv \Theta, x : \sigma_1}{\Gamma \vdash_e \lambda x : \sigma_1.\ e_1 \simeq \lambda x : \sigma_2.\ e_2 \dashv \Theta}\ \textsc{U-LamAnn}$$

$\Gamma \vdash \lambda x : \sigma_1.\ e_1 \Rightarrow \sigma_1'$	Given
$\Gamma \vdash \sigma_1 \Rightarrow \star$	By inversion

$\Gamma \vdash \lambda x : \sigma_2.\ e_2 \Rightarrow \sigma_2'$	Given
$\Gamma \vdash \sigma_2 \Rightarrow \star$	By inversion
$\Gamma \longrightarrow \Theta_1$	By Part 2
$\Gamma, x : \sigma_1 \longrightarrow \Theta_1, x : \sigma_1$	By CE-Var
$\Gamma, x : \sigma_1 \vdash e_1 \Rightarrow \sigma_1''$	By inversion
$\Theta_1, x : \sigma_1 \vdash e_1 \Rightarrow [\Theta_1, x : \sigma_1](\sigma_1'')$	By Lemma 30
$\Theta_1, x : \sigma_1 \vdash [\Theta_1, x : \sigma_1](e_1) \Rightarrow [\Theta_1, x : \sigma_1](\sigma_1'')$	By Lemma 13
$\Theta_1, x : \sigma_1 \vdash [\Theta_1](e_1) \Rightarrow [\Theta_1](\sigma_1'')$	By definition of context substitution
$\Theta_1, x : \sigma_1 \vdash [\Theta_1](e_2) \Rightarrow [\Theta_1](\sigma_2'')$	Similarly
$\Theta_1, x : \sigma_1 \longrightarrow \Theta, x : \sigma_1$	By induction hypothesis
$\Theta_1 \longrightarrow \Theta$	By inversion
$\Gamma \longrightarrow \Theta$	By Lemma 27

– Case

$$\frac{\Gamma \vdash_\sigma \sigma_1 \simeq \sigma_3 \dashv \Theta_1 \qquad \Theta_1, x : \sigma_1 \vdash_\sigma [\Theta_1](\sigma_2) \simeq [\Theta_1](\sigma_4) \dashv \Theta, x : \sigma_1}{\Gamma \vdash_\delta \Pi x : \sigma_1.\ \sigma_2 \simeq \Pi x : \sigma_3.\ \sigma_4 \dashv \Theta}\ \text{U-Pi}$$

Similar as Case U-LamAnn. The difference is that instead of using induction hypothesis, two subgoals all use Part 2.

– Case

$$\frac{\Gamma \vdash_e e_1 \simeq e_2 \dashv \Theta}{\Gamma \vdash_\delta \mathsf{cast}_\downarrow\ e_1 \simeq \mathsf{cast}_\downarrow\ e_2 \dashv \Theta}\ \text{U-CastDn}$$

$\Gamma \vdash \mathsf{cast}_\downarrow\ e_1 \Rightarrow \sigma_1'$	Given
$\Gamma \vdash e_1 \Rightarrow \sigma_1''$	By inversion
$\Gamma \vdash e_2 \Rightarrow \sigma_2''$	Similarly
$\Gamma \longrightarrow \Theta$	By induction hypothesis

– Case

$$\frac{\Gamma \vdash_e e_1 \simeq e_2 \dashv \Theta}{\Gamma \vdash_e \mathsf{cast}_\uparrow\ e_1 \simeq \mathsf{cast}_\uparrow\ e_2 \dashv \Theta}\ \text{U-CastUp}$$

Similar as Case U-CastDn.

Part 2 – Case

$$\frac{}{\Gamma \vdash_\delta \sigma \simeq \sigma \dashv \Gamma}\ \text{U-AEq}$$

Follows trivially by Lemma 26.

– Case

$$\frac{\widehat{\alpha} \notin FV(\tau_1) \qquad \Gamma[\widehat{\alpha}] \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta_1, \widehat{\alpha}, \Theta_2 \qquad \Theta_1 \vDash \tau_2}{\Gamma[\widehat{\alpha}] \vdash_\sigma \widehat{\alpha} \simeq \tau_1 \dashv \Theta_1, \widehat{\alpha} = \tau_2, \Theta_2}\ \text{U-EVarTy}$$

$\Gamma[\widehat{\alpha}] \vdash \tau_1 \Rightarrow \star$	Given

$\Gamma[\widehat{\alpha}] \longrightarrow \Theta_1, \widehat{\alpha}, \Theta_2$	By Lemma 42
$\Theta_1, \widehat{\alpha}, \Theta_2 \vdash \tau_2 \;\Rightarrow\; \star$	By Lemma 44
$\Theta_1 \vDash \tau_2$	Given
$\Theta_1 \vdash \tau_2 \;\Rightarrow\; \star$	By Lemma 17
$\Theta_1, \widehat{\alpha}, \Theta_2 \longrightarrow \Theta_1, \widehat{\alpha} = \tau_2, \Theta_2$	By Lemma 35
$\Gamma[\widehat{\alpha}] \longrightarrow \Theta_1, \widehat{\alpha} = \tau_2, \Theta_2$	By Lemma 27

– Case

$$\frac{\widehat{\alpha} \notin FV(\tau_1) \qquad \Gamma[\widehat{\alpha}] \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta_1, \widehat{\alpha}, \Theta_2 \qquad \Theta_1 \vDash \tau_2}{\Gamma[\widehat{\alpha}] \vdash_\sigma \tau_1 \;\simeq\; \widehat{\alpha} \dashv \Theta_1, \widehat{\alpha} = \tau_2, \Theta_2}\ \text{U-TyEVar}$$

Similar as Case U-EVarTy.

– Case

$$\frac{\Gamma \vdash_e e_1 \;\simeq\; e_3 \dashv \Theta_1 \qquad \Theta_1 \vdash_e [\Theta_1](e_2) \;\simeq\; [\Theta_1](e_4) \dashv \Theta}{\Gamma \vdash_\delta e_1\ e_2 \;\simeq\; e_3\ e_4 \dashv \Theta}\ \text{U-App}$$

Similar as Case U-App in Part 1.

– Case

$$\frac{\Gamma \vdash_\sigma \sigma_1 \;\simeq\; \sigma_3 \dashv \Theta_1 \qquad \Theta_1, x : \sigma_1 \vdash_\sigma [\Theta_1](\sigma_2) \;\simeq\; [\Theta_1](\sigma_4) \dashv \Theta, x : \sigma_1}{\Gamma \vdash_\delta \Pi x : \sigma_1.\ \sigma_2 \;\simeq\; \Pi x : \sigma_3.\ \sigma_4 \dashv \Theta}\ \text{U-Pi}$$

Similar as Case U-Pi in Part 1.

– Case

$$\frac{\Gamma \vdash_e e_1 \;\simeq\; e_2 \dashv \Theta}{\Gamma \vdash_\delta \mathsf{cast}_\downarrow\ e_1 \;\simeq\; \mathsf{cast}_\downarrow\ e_2 \dashv \Theta}\ \text{U-CastDn}$$

Similar as Case U-CastDn in Part 1.

□

Lemma 49 (Unification Equivalence). *If* $\Gamma \vdash_\delta \tau_1 \;\simeq\; \tau_2 \dashv \Theta$, *and* $\Gamma \vdash \tau_1 \;\Rightarrow\; \sigma_1'$, *and* $\Gamma \vdash \tau_2 \;\Rightarrow\; \sigma_2'$, *then* $[\Theta](\tau_1) = [\Theta](\tau_2)$.

Proof. By induction on unification derivation.

– Case

$$\frac{}{\Gamma \vdash_\delta \sigma \;\simeq\; \sigma \dashv \Gamma}\ \text{U-AEq}$$

The goal holds trivially.

– Case

$$\frac{\widehat{\alpha} \notin FV(\tau_1) \qquad \Gamma[\widehat{\alpha}] \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta_1, \widehat{\alpha}, \Theta_2 \qquad \Theta_1 \vDash \tau_2}{\Gamma[\widehat{\alpha}] \vdash_\sigma \widehat{\alpha} \;\simeq\; \tau_1 \dashv \Theta_1, \widehat{\alpha} = \tau_2, \Theta_2}\ \text{U-EVarTy}$$

$\Gamma[\widehat{\alpha}] \vdash \tau_1 \;\Rightarrow\; \sigma_2'$	Given

$[\Theta_1, \widehat{\alpha}, \Theta_2](\tau_1) = [\Theta_1, \widehat{\alpha}, \Theta_2](\tau_2)$	By Lemma 43
$[\Theta_1, \widehat{\alpha} = \tau_2, \Theta_2](\widehat{\alpha})$	
$= [\Theta_1](\tau_2)$	By definition of context substitution
$[\Theta_1, \widehat{\alpha} = \tau_2, \Theta_2](\tau_1)$	
$= [\Theta_1, \widehat{\alpha} = \tau_2]([\Theta_2](\tau_1))$	By definition of context application
$= [\Theta_1](([\Theta_2](\tau_1))[\widehat{\alpha} \mapsto \tau_2])$	By definition of context application
$= ([\Theta_1]([\Theta_2](\tau_1)))[\widehat{\alpha} \mapsto [\Theta_1](\tau_2)]$	By property of context application and substitution
$= ([\Theta_1, \Theta_2](\tau_1))[\widehat{\alpha} \mapsto [\Theta_1](\tau_2)]$	By definition of context application
$= ([\Theta_1, \widehat{\alpha}, \Theta_2](\tau_1))[\widehat{\alpha} \mapsto [\Theta_1](\tau_2)]$	By definition of context application
$= ([\Theta_1, \widehat{\alpha}, \Theta_2](\tau_2))[\widehat{\alpha} \mapsto [\Theta_1](\tau_2)]$	By substituting the equality
$\Theta_1 \vDash \tau_2$	Given
$\Gamma[\widehat{\alpha}]\ ctx$	By Lemma 15
$\Gamma[\widehat{\alpha}] \longrightarrow \Theta_1, \widehat{\alpha}, \Theta_2$	By Lemma 42
$\Theta_1, \widehat{\alpha}, \Theta_2\ ctx$	By Lemma 32
$\widehat{\alpha} \notin \Theta_1 \cup FV(\tau_2)$	Follows directly
τ_2 contains no existential variable in Θ_2	Follows directly
$[\Theta_1, \widehat{\alpha} = \tau_2, \Theta_2](\tau_1)$	
$= ([\Theta_1, \widehat{\alpha}, \Theta_2](\tau_2))[\widehat{\alpha} \mapsto [\Theta_1](\tau_2)]$	Known
$= ([\Theta_1](\tau_2))[\widehat{\alpha} \mapsto [\Theta_1](\tau_2)]$	Substitute fresh context
$= ([\Theta_1](\tau_2))$	Substitute fresh variable

– Case

$$\frac{\widehat{\alpha} \notin FV(\tau_1) \qquad \Gamma[\widehat{\alpha}] \vdash \tau_1 \longrightarrow \tau_2 \dashv \Theta_1, \widehat{\alpha}, \Theta_2 \qquad \Theta_1 \vDash \tau_2}{\Gamma[\widehat{\alpha}] \vdash_\sigma \tau_1 \ \simeq\ \widehat{\alpha} \dashv \Theta_1, \widehat{\alpha} = \tau_2, \Theta_2}\ \text{U-TyEVar}$$

Similar as Case U-EVarTy.

– Case

$$\frac{\Gamma \vdash_e e_1 \ \simeq\ e_3 \dashv \Theta_1 \qquad \Theta_1 \vdash_e [\Theta_1](e_2) \ \simeq\ [\Theta_1](e_4) \dashv \Theta}{\Gamma \vdash_\delta e_1\ e_2 \ \simeq\ e_3\ e_4 \dashv \Theta}\ \text{U-App}$$

$\Gamma \vdash e_1\ e_2 \ \Rightarrow\ \sigma_1'$	Given
$\Gamma \vdash e_1 \ \Rightarrow\ \sigma_1''$	By inversion
$\Gamma \vdash e_2 \ \Rightarrow\ \sigma_1'''$	By inversion
$\Gamma \vdash e_3\ e_4 \ \Rightarrow\ \sigma_2'$	Given
$\Gamma \vdash e_3 \ \Rightarrow\ \sigma_2''$	By inversion
$\Gamma \vdash e_4 \ \Rightarrow\ \sigma_2'''$	By inversion
$[\Theta_1](e_1) = [\Theta_1](e_3)$	By induction hypothesis
$\Gamma \longrightarrow \Theta_1$	By Lemma 48
$\Theta_1 \vdash e_2 \ \Rightarrow\ [\Theta_1](\sigma_1''')$	By Lemma 30
$\Theta_1 \vdash [\Theta_1](e_2) \ \Rightarrow\ [\Theta_1](\sigma_1''')$	By Lemma 13
$\Theta_1 \vdash [\Theta_1](e_4) \ \Rightarrow\ [\Theta_1](\sigma_2''')$	Similarly

$[\Theta]([\Theta_1](e_2)) = [\Theta]([\Theta_1](e_4))$ By induction hypothesis
$\Theta_1 \longrightarrow \Theta$ By Lemma 48
$[\Theta](e_2) = [\Theta](e_4)$ By Lemma 24
$[\Theta](e_1) = [\Theta](e_3)$ By Lemma 24

– Case

$$\frac{\Gamma \vdash_\sigma \sigma_1 \simeq \sigma_2 \dashv \Theta_1 \qquad \Theta_1, x : \sigma_1 \vdash_e [\Theta_1](e_1) \simeq [\Theta_1](e_2) \dashv \Theta, x : \sigma_1}{\Gamma \vdash_e \lambda x : \sigma_1.\ e_1 \simeq \lambda x : \sigma_2.\ e_2 \dashv \Theta}\ \textsc{U-LamAnn}$$

$\Gamma \vdash \lambda x : \sigma_1.\ e_1 \Rightarrow \sigma_1'$ Given
$\Gamma \vdash \sigma_1 \Rightarrow \star$ By inversion
$\Gamma, x : \sigma_1 \vdash e_1 \Rightarrow \sigma_1''$ By inversion
$\Gamma \vdash \sigma_2 \Rightarrow \star$ Similarly
$\Gamma, x : \sigma_2 \vdash e_2 \Rightarrow \sigma_2''$ Similarly
$[\Theta_1](\sigma_1) = [\Theta_1](\sigma_2)$ By induction hypothesis
$\Gamma \longrightarrow \Theta_1$ By Lemma 48
$\Gamma, x : \sigma_1 \longrightarrow \Theta_1, x : \sigma_1$ By CE-Var
$\Theta_1, x : \sigma_1 \vdash e_1 \Rightarrow [\Theta_1, x : \sigma_1](\sigma_1'')$ By Lemma 30
$\Theta_1, x : \sigma_1 \vdash [\Theta_1, x : \sigma_1](e_1) \Rightarrow [\Theta_1, x : \sigma_1](\sigma_1'')$ By Lemma 13
$\Theta_1, x : \sigma_1 \vdash [\Theta_1](e_1) \Rightarrow [\Theta_1](\sigma_1'')$ By definition of context substitution
$\Gamma, x : \sigma_2 \longrightarrow \Theta_1, x : \sigma_2$ By CE-Var
$\Theta_1, x : \sigma_2 \vdash [\Theta_1](e_2) \Rightarrow [\Theta_1](\sigma_2'')$ Similarly
$\Theta_1 \vdash \sigma_2 \Rightarrow \star$ By Lemma 30
$\Theta_1 \vdash [\Theta_1](\sigma_2) \Rightarrow \star$ By Lemma 13
$\Theta_1, x : [\Theta_1](\sigma_2) \vdash [\Theta_1](e_2) \Rightarrow [\Theta_1](\sigma_2'')$ By Lemma 11
$\Theta_1, x : [\Theta_1](\sigma_1) \vdash [\Theta_1](e_2) \Rightarrow [\Theta_1](\sigma_2'')$ By substituting the equality
$\Theta_1 \vdash \sigma_1 \Rightarrow \star$ By Lemma 30
$\Theta_1, x : \sigma_1 \vdash [\Theta_1](e_2) \Rightarrow [\Theta_1](\sigma_2'')$ By Lemma 12
$[\Theta, x : \sigma_1]([\Theta_1](e_1)) = [\Theta, x : \sigma_1]([\Theta_1](e_2))$ By induction hypothesis
$[\Theta]([\Theta_1](e_1)) = [\Theta]([\Theta_1](e_2))$ By definition of context application
$\Theta_1, x : \sigma_1 \longrightarrow \Theta, x : \sigma_1$ By Lemma 48
$\Theta_1 \longrightarrow \Theta$ By inversion
$[\Theta](e_1) = [\Theta](e_2)$ By Lemma 24
$[\Theta](\sigma_1) = [\Theta](\sigma_2)$ By Lemma 24

– Case

$$\frac{\Gamma \vdash_\sigma \sigma_1 \simeq \sigma_3 \dashv \Theta_1 \qquad \Theta_1, x : \sigma_1 \vdash_\sigma [\Theta_1](\sigma_2) \simeq [\Theta_1](\sigma_4) \dashv \Theta, x : \sigma_1}{\Gamma \vdash_\delta \Pi x : \sigma_1.\ \sigma_2 \simeq \Pi x : \sigma_3.\ \sigma_4 \dashv \Theta}\ \textsc{U-Pi}$$

Similar as Case U-LamAnn.

– Case

$$\frac{\Gamma \vdash_e e_1 \simeq e_2 \dashv \Theta}{\Gamma \vdash_\delta \mathsf{cast}_\downarrow\ e_1 \simeq \mathsf{cast}_\downarrow\ e_2 \dashv \Theta}\ \text{U-CastDn}$$

$\Gamma \vdash \mathsf{cast}_\downarrow\ e_1 \Rightarrow \sigma_1'$	Given
$\Gamma \vdash e_1 \Rightarrow \sigma_1''$	By inversion
$\Gamma \vdash e_2 \Rightarrow \sigma_2''$	Similarly
$[\Theta](e_1) = [\Theta](e_2)$	By induction hypothesis

– Case

$$\frac{\Gamma \vdash_e e_1 \simeq e_2 \dashv \Theta}{\Gamma \vdash_e \mathsf{cast}_\uparrow\ e_1 \simeq \mathsf{cast}_\uparrow\ e_2 \dashv \Theta}\ \text{U-CastUp}$$

Similar as Case U-CastDn.

□

We use the notation $\Gamma \vdash \sigma_1 = \sigma_2 \Rightarrow \tau$ to mean that $\Gamma \vdash \sigma_1 \Rightarrow \tau$, $\Gamma \vdash \sigma_2 \Rightarrow \tau$, and $\sigma_1 = \sigma_2$.

Lemma 50 (Unification Completeness). *Given $\Gamma \longrightarrow \Omega$, and $[\Gamma](\sigma_1) = \sigma_1$, and $[\Gamma](\sigma_2) = \sigma_2$. If $[\Omega](\Gamma) \vdash [\Omega](\sigma_1) = [\Omega](\sigma_2) \Rightarrow \tau$, then there are Δ, Ω' such that $\Omega \longrightarrow \Omega'$, and $\Delta \longrightarrow \Omega'$, and $\Gamma \vdash_\delta \sigma_1 \simeq \sigma_2 \dashv \Delta$ for any δ suitable.*

Proof. By induction on the typing size $[\Omega](\Gamma) \vdash [\Omega](\sigma_1) \Rightarrow \tau$. We then do case analysis on the shape of σ_1.

– Case $\sigma_1 = \widehat{\alpha}$. Depending on whether $\widehat{\alpha} \in FV(\sigma_2)$, we have two subcases.
  - SubCase $\widehat{\alpha} \in FV(\sigma_2)$. Then there is only one possible case $\sigma_2 = \widehat{\alpha}$. Otherwise, $[\Omega](\sigma_1)$ cannot be equal to $[\Omega](\sigma_2)$. Therefore, we can use rule U-AEq to get that $\Gamma \vdash \widehat{\alpha} \simeq \widehat{\alpha} \dashv \Gamma$. Choose $\Omega' = \Omega$ and we are done.
  - SubCase $\widehat{\alpha} \notin FV(\sigma_2)$. Follows directly by Lemma 47 and rule U-EVarTy.

– In all rest cases, when we do inversion on the equality, we could have case $\sigma_2 = \widehat{\alpha}$. Basically the proof for when σ_2 is an existential variable is similar as last case, with rule U-TyEVar instead of U-EVarTy. So in the rest cases, we will ignore the cases when σ_2 is an existential variable.

– Case $\sigma_1 = x$. Then by inversion we have $\sigma_2 = x$. Therefore, we can use rule U-AEq to get that $\Gamma \vdash x \simeq x \dashv \Gamma$. Choose $\Omega' = \Omega$ and we are done.

– Case $\sigma_1 = \star$. Then by inversion we have $\sigma_2 = \star$. Therefore, we can use rule U-AEq to get that $\Gamma \vdash \star \simeq \star \dashv \Gamma$. Choose $\Omega' = \Omega$ and we are done.

– Case $\sigma_1 = e_1\ e_2$.

$\sigma_2 = e_3\ e_4$	By inversion
$[\Omega](\Gamma) \vdash [\Omega](e_1) = [\Omega](e_3) \Rightarrow \tau'$	As above
$[\Omega](\Gamma) \vdash [\Omega](e_2) = [\Omega](e_4) \Rightarrow \tau''$	As above
$\Gamma \vdash_e e_1 \simeq e_3 \dashv \Theta$	By induction hypothesis
$\Theta \longrightarrow \Omega_1$	As above

$\Omega \longrightarrow \Omega_1$	As above
$[\Omega](\Omega) \vdash [\Omega](e_2) = [\Omega](e_4) \Rightarrow \tau''$	By Lemma 38
$[\Omega_1](\Omega_1) \vdash [\Omega](e_2) = [\Omega](e_4) \Rightarrow \tau''$	By Lemma 40
$[\Omega_1](\Theta) \vdash [\Omega](e_2) = [\Omega](e_4) \Rightarrow \tau''$	By Lemma 41
$[\Omega_1](\Theta) \vdash [\Omega_1](e_2) = [\Omega_1](e_4) \Rightarrow \tau''$	By Lemma 39
$[\Omega_1](\Theta) \vdash [\Omega_1]([\Theta](e_2)) = [\Omega_1]([\Theta](e_4)) \Rightarrow \tau''$	By Lemma 24
$\Theta \vdash_e [\Theta](e_2) \simeq [\Theta](e_4) \dashv \Delta$	By induction hypothesis
$\Delta \longrightarrow \Omega_2$	As above
$\Omega_1 \longrightarrow \Omega_2$	As above
$\Gamma \vdash_\delta e_1\ e_2 \simeq e_3\ e_4 \dashv \Delta$	By U-App
$\Omega' = \Omega_2$	Choose
$\Omega \longrightarrow \Omega_2$	By Lemma 27

– Case $\sigma_1 = \lambda x : \tau_1.\ e_1$

$\sigma_2 = \lambda x : \tau_2.\ e_2$	By inversion
$[\Omega](\Gamma) \vdash [\Omega](\tau_1) = [\Omega](\tau_2) \Rightarrow \star$	As above
$[\Omega](\Gamma), x : [\Omega](\tau_1) \vdash [\Omega](e_1) = [\Omega](e_2) \Rightarrow \tau'$	As above
$\Gamma \vdash_\sigma \tau_1 \simeq \tau_2 \dashv \Theta$	By induction hypothesis
$\Theta \longrightarrow \Omega_1$	As above
$\Omega \longrightarrow \Omega_1$	As above
$\Omega, x : \tau_1 \longrightarrow \Omega_1, x : \tau_1$	By CE-Var
$\Theta, x : \tau_1 \longrightarrow \Omega_1, x : \tau_1$	By CE-Var
$\Gamma, x : \tau_1 \longrightarrow \Omega, x : \tau_1$	By Lemma 48
$[\Omega, x : \tau_1](\Gamma, x : \tau_1) \vdash [\Omega, x : \tau_1](e_1) = [\Omega, x : \tau_1](e_2) \Rightarrow \tau'$	By definition of context application
$[\Omega, x : \tau_1](\Omega, x : \tau_1) \vdash [\Omega, x : \tau_1](e_1) = [\Omega, x : \tau_1](e_2) \Rightarrow \tau'$	Lemma 38
$[\Omega_1, x : \tau_1](\Omega_1, x : \tau_1) \vdash [\Omega_1, x : \tau_1](e_1) = [\Omega_1, x : \tau_1](e_2) \Rightarrow \tau'$	Lemma 40 and Lemma 39
$[\Omega_1, x : \tau_1](\Theta, x : \tau_1) \vdash [\Omega_1, x : \tau_1](e_1) = [\Omega_1, x : \tau_1](e_2) \Rightarrow \tau'$	Lemma 38
$[\Omega_1, x : \tau_1](\Theta, x : \tau_1) \vdash [\Omega_1, x : \tau_1]([\Theta](e_1))$	Lemma 24
$= [\Omega_1, x : \tau_1]([\Theta](e_2)) \Rightarrow \tau'$	and definition of context application
$\Theta, x : \tau_1 \vdash_e [\Theta](e_1) \simeq [\Theta](e_2) \dashv \Delta, x : \tau_1$	By induction hypothesis and Lemma 45
$\Delta, x : \tau_1 \longrightarrow \Omega_2$	By induction hypothesis
$\Omega_1, x : \tau_1 \longrightarrow \Omega_2$	By induction hypothesis
$\Omega_2 = \Omega_3, x : \tau_1, \Omega_4$	By Lemma 29
$\Delta \longrightarrow \Omega_3$	As above
$\Omega_1 \longrightarrow \Omega_3$	As above
$\Omega' = \Omega_3$	Choose
$\Gamma \vdash_e \lambda x : \tau_1.\ e_1 \simeq \lambda x : \tau_2.\ e_2 \dashv \Delta$	By U-LamAnn
$\Omega \longrightarrow \Omega_3$	By Lemma 27

– Case $\sigma_1 = \Pi x : \tau_1.\ \tau_2$ Similar as last case.
– Case $\sigma_1 = \mathsf{cast}_\downarrow\ \tau_1$

$\sigma_2 = \mathsf{cast}_{\downarrow}\ \tau_2$	By inversion
$[\Omega](\Gamma) \vdash [\Omega](\tau_1) = [\Omega](\tau_2) \Rightarrow \tau'$	As above
$\Gamma \vdash \tau_1 \simeq \tau_2 \dashv \Delta$	By induction hypothesis
$\Delta \longrightarrow \Omega_1$	As above
$\Omega \longrightarrow \Omega_1$	As above
$\Omega' = \Omega_1$	Choose
$\Gamma \vdash \mathsf{cast}_{\downarrow}\ \tau_1 \simeq \mathsf{cast}_{\downarrow}\ \tau_2 \dashv \Delta$	By U-CastDn

– Case $\sigma_1 = \mathsf{cast}_{\uparrow}\ \tau_1$ Similar as last cast.

□

D Implicit Polymorphic Type System

D.1 Referred Lemmas

Lemma 51 (Unsolved Variable Addition For Extension). $\Gamma_L, \Gamma_R \longrightarrow \Gamma_L, \widehat{\alpha}, \Gamma_R$.

Lemma 52 (Solution Admissibility for Extension). *If* $\Gamma \vdash \tau$, *then* $\Gamma_L, \widehat{\alpha}, \Gamma_R \longrightarrow \Gamma_L, \widehat{\alpha} = \tau, \Gamma_R$.

Lemma 53 (Transitivity). *If* $\Gamma \longrightarrow \Delta$, *and* $\Delta \longrightarrow \Theta$, *then* $\Gamma \longrightarrow \Theta$.

Lemma 54 (Reflexivity). *If* Γ *is well-formed, then* $\Gamma \longrightarrow \Gamma$.

Lemma 55 (Confluence of Completeness). *If* $\Delta_1 \longrightarrow \Omega$, *and* $\Delta_2 \longrightarrow \Omega$, *then* $[\Omega](\Delta_1) = [\Omega](\Delta_2)$.

Lemma 56 (Substitution Extension Invariance). *If* $\Theta \vdash A$, *and* $\Theta \longrightarrow \Gamma$, *then* $[\Gamma](A) = [\Gamma]([\Theta](A))$, *and* $[\Gamma](A) = [\Theta]([\Gamma](A))$.

Lemma 57 (Reflexivity of Declarative Subtyping). *If* $\Psi \vdash A$, *then* $\Psi \vdash A \leq A$.

Lemma 58 (Monotype Equality). *If* $\Psi \vdash \sigma \leq \tau$, *then* $\sigma = \tau$.

Lemma 59 (Parallel Admissibility). *If* $\Gamma_L \longrightarrow \Delta_L$, *and* $\Gamma_L, \Gamma_R \longrightarrow \Delta_L, \Delta_R$, *then:*

– $\Gamma_L, \widehat{\alpha}, \Gamma_R \longrightarrow \Delta_L, \widehat{\alpha}, \Delta_R$
– *If* $\Delta_L \vdash \tau'$, *then* $\Gamma_L, \widehat{\alpha}, \Gamma_R \longrightarrow \Delta_L, \widehat{\alpha} = \tau', \Delta_R$
– *If* $\Gamma_L \vdash \tau$, *and* $\Delta_L \vdash \tau'$, *and* $[\Delta_L](\tau) = [\Delta_L](\tau')$, *then* $\Gamma_L, \widehat{\alpha} = \tau, \Gamma_R \longrightarrow \Delta_L, \widehat{\alpha} = \tau', \Delta_R$.

Lemma 60 (Parallel Extension Solution). *If* $\Gamma_L, \widehat{\alpha}, \Gamma_R \longrightarrow \Delta_L, \widehat{\alpha} = \tau', \Delta_R$, *and* $\Gamma_L \vdash \tau$, *and* $[\Delta_L](\tau) = [\Delta_L](\tau')$, *then* $\Gamma_L, \widehat{\alpha} = \tau, \Gamma_R \longrightarrow \Delta_L, \widehat{\alpha} = \tau', \Delta_R$.

Lemma 61 (Solved Variable Addition for Extension). *If* $\Gamma_L \vdash \tau$, *and* $\Gamma_L, \Gamma_R \longrightarrow \Gamma_L, \widehat{\alpha} = \tau, \Gamma_R$.

Lemma 62 (Extension Order).

- $\Gamma_L, \alpha, \Gamma_R \longrightarrow \Delta$, *then* $\Delta = (\Delta_L, \alpha, \Delta_R)$ *where* $\Gamma_L \longrightarrow \Delta_L$. *Moreover, if* Γ_R *is soft then* Δ_R *is soft.*
- $\Gamma_L, \blacktriangleright_{\widehat{\alpha}}, \Gamma_R \longrightarrow \Delta$, *then* $\Delta = (\Delta_L, \blacktriangleright_{\widehat{\alpha}}, \Delta_R)$ *where* $\Gamma_L \longrightarrow \Delta_L$. *Moreover, if* Γ_R *is soft then* Δ_R *is soft.*
- $\Gamma_L, \widehat{\alpha}, \Gamma_R \longrightarrow \Delta$, *then* $\Delta = (\Delta_L, \Theta, \Delta_R)$ *where* $\Gamma_L \longrightarrow \Delta_L$, *and* Θ *is either* $\widehat{\alpha}$ *or* $\widehat{\alpha} = \tau$ *for some* τ.
- $\Gamma_L, \widehat{\alpha} = \tau, \Gamma_R \longrightarrow \Delta$, *then* $\Delta = (\Delta_L, \widehat{\alpha} = \tau', \Delta_R)$ *where* $\Gamma_L \longrightarrow \Delta_L$, *and* $[\Delta_L](\tau) = [\Delta_L](\tau')$.
- $\Gamma_L, x : A, \Gamma_R \longrightarrow \Delta$, *then* $\Delta = (\Delta_L, x : A', \Delta_R)$ *where* $\Gamma_L \longrightarrow \Delta_L$, *and* $[\Delta_L](A) = [\Delta_L](A')$. *Moreover, if* Γ_R *is soft then* Δ_R *is soft.*

Lemma 63 (Reverse Declaration Order Preservation). *If* $\Gamma \longrightarrow \Delta$ *and* u *and* v *are both declared in* Γ u *is declared to the left of* v *in* Δ, *then* u *is declared to the left of* v *in* Γ.

Lemma 64 (Stability of Complete Contexts). *If* $\Gamma \longrightarrow \Omega$, *then* $[\Omega](\Gamma) = [\Omega](\Omega)$.

Lemma 65 (Finishing Types). *If* $\Omega \vdash A$, *and* $\Omega \longrightarrow \Omega'$, *then* $[\Omega](A) = [\Omega'](A)$.

Lemma 66 (Finishing Completions). *If* $\Omega \longrightarrow \Omega'$, *then* $[\Omega](\Omega) = [\Omega'](\Omega')$.

Lemma 67 (Extension Weakening). *If* $\Gamma \vdash A$, *and* $\Gamma \longrightarrow \Delta$, *then* $\Delta \vdash A$.

Lemma 68 (Substitution Typing). *If* $\Gamma \vdash A$, *then* $\Gamma \vdash [\Gamma](A)$.

Proposition 1 (Weakening). *If* $\Psi \vdash A$, *then* $\Psi, \Psi' \vdash A$ *by a derivation of the same size.*

D.2 Properties of Polymorphic Type Sanitization

Lemma 69 (Polymorphic Type Sanitization Extension). *If* $\Gamma \vdash^s A \longrightarrow \sigma \dashv \Theta$, *then* $\Gamma \longrightarrow \Theta$.

Proof. By induction on polymorphic type sanitization derivation.

- Case

$$\frac{\Gamma[\widehat{\beta}, \widehat{\alpha}] \vdash^+ A[\alpha \mapsto \widehat{\beta}] \longrightarrow \sigma \dashv \Theta}{\Gamma[\widehat{\alpha}] \vdash^+ \forall \alpha. A \longrightarrow \sigma \dashv \Theta} \text{ I-All-Plus}$$

$\Gamma[\widehat{\beta}, \widehat{\alpha}] \longrightarrow \Theta$	By induction hypothesis
$\Gamma[\widehat{\alpha}] \longrightarrow \Gamma[\widehat{\beta}, \widehat{\alpha}]$	By Lemma 51
$\Gamma[\widehat{\alpha}] \longrightarrow \Theta$	By Lemma 53

– Case

$$\frac{\Gamma, \alpha \vdash^{-} A \longrightarrow \sigma \dashv \Theta, \alpha}{\Gamma \vdash^{-} \forall \alpha.A \longrightarrow \sigma \dashv \Theta} \text{ I-All-Minus}$$

$\Gamma, \alpha \longrightarrow \Theta, \alpha$ By induction hypothesis
$\Gamma \longrightarrow \Theta$ By inversion

– Case

$$\frac{\Gamma \vdash^{-s} A_1 \longrightarrow \sigma_1 \dashv \Theta_1 \qquad \Theta_1 \vdash^{s} [\Theta_1](A_2) \longrightarrow \sigma_2 \dashv \Theta}{\Gamma \vdash^{s} A_1 \to A_2 \longrightarrow \sigma_1 \to \sigma_2 \dashv \Theta} \text{ I-Pi-Poly}$$

$\Gamma \longrightarrow \Theta_1$ By induction hypothesis
$\Theta_1 \longrightarrow \Theta$ By induction hypothesis
$\Gamma \longrightarrow \Theta$ By Lemma 53

– Case

$$\frac{}{\Gamma \vdash^{s} 1 \longrightarrow 1 \dashv \Gamma} \text{ I-Unit}$$

Γ is well-formed By implicit assumption
$\Gamma \longrightarrow \Gamma$ By Lemma 54

– Case

$$\frac{}{\Gamma[\alpha] \vdash^{s} \alpha \longrightarrow \alpha \dashv \Gamma[\alpha]} \text{ I-TVar}$$

$\Gamma[\alpha]$ is well-formed By implicit assumption
$\Gamma[\alpha] \longrightarrow \Gamma[\alpha]$ By Lemma 54

– Case

$$\frac{}{\Gamma[\widehat{\alpha}][\widehat{\beta}] \vdash^{s} \widehat{\beta} \longrightarrow \widehat{\alpha}_1 \dashv \Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}_1]} \text{ I-EVarAfter-Poly}$$

$\Gamma[\widehat{\alpha}][\widehat{\beta}] \longrightarrow \Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta}]$ By Lemma 51
$\Gamma[\widehat{\alpha}_1, \widehat{\alpha}] \vdash \widehat{\alpha}_1$ By EVarWF
$\Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta}] \longrightarrow \Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}_1]$ By Lemma 52
$\Gamma[\widehat{\alpha}][\widehat{\beta}] \longrightarrow \Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}_1]$ By Lemma 53

– Case

$$\frac{}{\Gamma[\widehat{\beta}][\widehat{\alpha}] \vdash^{s} \widehat{\beta} \longrightarrow \widehat{\beta} \dashv \Gamma[\widehat{\beta}][\widehat{\alpha}]} \text{ I-EVarBefore-Poly}$$

$\Gamma[\widehat{\beta}][\widehat{\alpha}]$ is well-formed	By implicit assumption
$\Gamma[\widehat{\beta}][\widehat{\alpha}] \longrightarrow \Gamma[\widehat{\beta}][\widehat{\alpha}]$	By Lemma 54

□

Lemma 70 (Polymorphic Type Sanitization Soundness). *Given* $\Theta \longrightarrow \Omega$, *and* $[\Gamma](A) = A$*:*

- *If* $\Gamma[\widehat{\alpha}] \vdash^{+} A \longrightarrow \sigma \dashv \Theta$, *then* $[\Omega](\Theta) \vdash [\Omega](A) \leq [\Omega](\sigma)$.
- *If* $\Gamma[\widehat{\alpha}] \vdash^{-} A \longrightarrow \sigma \dashv \Theta$, *then* $[\Omega](\Theta) \vdash [\Omega](\sigma) \leq [\Omega](A)$.

Proof. By induction on the polymorphic type sanitization derivation. These two parts rely on each other, but the derivation get smaller so the proof terminates.

Part 1 In this part, we regard $s = +$.

–

$$\frac{\Gamma[\widehat{\beta}, \widehat{\alpha}] \vdash^{+} A[\alpha \mapsto \widehat{\beta}] \longrightarrow \sigma \dashv \Theta}{\Gamma[\widehat{\alpha}] \vdash^{+} \forall \alpha.A \longrightarrow \sigma \dashv \Theta} \text{ I-All-Plus}$$

$\Theta \longrightarrow \Omega$	Given
$[\Gamma[\widehat{\alpha}]](\forall \alpha.A) = \forall \alpha.A$	Given
$[\Gamma[\widehat{\beta}, \widehat{\alpha}]](A[\alpha \mapsto \widehat{\beta}]) = A[\alpha \mapsto \widehat{\beta}]$	Follows directly
$[\Omega](\Theta) \vdash [\Omega](A[\alpha \mapsto \widehat{\beta}]) \leq [\Omega](\sigma)$	By induction hypothesis
$[\Omega](\Theta) \vdash [\Omega](A)[\alpha \mapsto [\Omega](\widehat{\beta})] \leq [\Omega](\sigma)$	By property of substitution
$[\Omega](\Theta) \vdash [\Omega](\forall \alpha.A) \leq [\Omega](\sigma)$	By $\leq$ $\forall$ L with $\tau = [\Omega](\widehat{\beta})$

–

$$\frac{\Gamma \vdash^{-s} A_1 \longrightarrow \sigma_1 \dashv \Theta_1 \qquad \Theta_1 \vdash^{s} [\Theta_1](A_2) \longrightarrow \sigma_2 \dashv \Theta}{\Gamma \vdash^{s} A_1 \to A_2 \longrightarrow \sigma_1 \to \sigma_2 \dashv \Theta} \text{ I-Pi-Poly}$$

$[\Gamma](A_1 \to A_2) = A_1 \to A_2$	Given
$[\Gamma](A_1) = A_1$	Follows directly
$\Theta_1 \longrightarrow \Theta$	By Lemma 69
$\Theta \longrightarrow \Omega$	Given
$\Theta_1 \longrightarrow \Omega$	By Lemma 53
$[\Omega](\Theta_1) \vdash [\Omega](\sigma_1) \leq [\Omega](A_1)$	By Part 2
$[\Omega](\Theta) \vdash [\Omega](\sigma_1) \leq [\Omega](A_1)$	By Lemma 55
$[\Theta_1]([\Theta_1](A_2)) = [\Theta_1](A_2)$	
$[\Omega](\Theta) \vdash [\Omega]([\Theta_1](A_2)) \leq [\Omega](\sigma_2)$	By induction hypothesis
$[\Omega](\Theta) \vdash [\Omega](A_2) \leq [\Omega](\sigma_2)$	By Lemma 56
$[\Omega](\Theta) \vdash [\Omega](A_1 \to A_2) \leq [\Omega](\sigma_1 \to \sigma_2)$	By $\leq$ $\to$

–

$$\frac{}{\Gamma \vdash^{s} 1 \longrightarrow 1 \dashv \Gamma} \text{ I-Unit}$$

$[\Omega](\Gamma) \vdash 1 \ \leq \ 1$ By $\leq$ UNIT

–

$$\frac{}{\Gamma[\alpha] \vdash^s \alpha \longrightarrow \alpha \dashv \Gamma[\alpha]} \text{ I-TVAR}$$

$[\Omega](\Gamma[\widehat{\alpha}]) \vdash [\Omega](\widehat{\alpha}) \ \leq \ [\Omega](\widehat{\alpha})$ By Lemma 57

–

$$\frac{}{\Gamma[\widehat{\alpha}][\widehat{\beta}] \vdash^s \widehat{\beta} \longrightarrow \widehat{\alpha}_1 \dashv \Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}_1]} \text{ I-EVARAFTER-POLY}$$

$[\Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}_1]](\widehat{\alpha}_1)$
$= [\Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}_1]](\widehat{\beta})$
$= \widehat{\alpha}_1$ By definition of context application
$\Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}_1] \longrightarrow \Omega$ Given
$[\Omega](\widehat{\beta}) = [\Omega](\widehat{\alpha}_1)$ By Lemma 56
$[\Omega](\Gamma) \vdash [\Omega](\widehat{\beta}) \ \leq \ [\Omega](\widehat{\alpha}_1)$ By Lemma 57

–

$$\frac{}{\Gamma[\widehat{\beta}][\widehat{\alpha}] \vdash^s \widehat{\beta} \longrightarrow \widehat{\beta} \dashv \Gamma[\widehat{\beta}][\widehat{\alpha}]} \text{ I-EVARBEFORE-POLY}$$

$[\Omega](\Gamma[\widehat{\alpha}]) \vdash [\Omega](\widehat{\beta}) \ \leq \ [\Omega](\widehat{\beta})$ By Lemma 57

Part 2 In this part, we regard $s = -$.

–

$$\frac{\Gamma, \alpha \vdash^- A \longrightarrow \sigma \dashv \Theta, \alpha}{\Gamma \vdash^- \forall \alpha.A \longrightarrow \sigma \dashv \Theta} \text{ I-ALL-MINUS}$$

$\Theta \longrightarrow \Omega$ Given
$\Theta, \alpha \longrightarrow \Omega, \alpha$ By $\longrightarrow$ UVAR
$[\Gamma](\forall \alpha.A) = \forall \alpha.A$ Given
$[\Gamma, \alpha](A) = A$ Follows directly
$[\Omega, \alpha](\Gamma, \alpha) \vdash [\Omega, \alpha](A) \ \leq \ [\Omega, \alpha](\sigma)$ By induction hypothesis
$[\Omega](\Gamma), \alpha \vdash [\Omega](A) \ \leq \ [\Omega](\sigma)$ By definition of context application
$[\Omega](\Gamma) \vdash [\Omega](\forall \alpha.A) \ \leq \ [\Omega](\sigma)$ By $\leq \ \forall$ L

–

$$\frac{\Gamma \vdash^{-s} A_1 \longrightarrow \sigma_1 \dashv \Theta_1 \qquad \Theta_1 \vdash^s [\Theta_1](A_2) \longrightarrow \sigma_2 \dashv \Theta}{\Gamma \vdash^s A_1 \to A_2 \longrightarrow \sigma_1 \to \sigma_2 \dashv \Theta} \text{ I-PI-POLY}$$

Similar as in Part 1.

–

$$\frac{}{\Gamma \vdash^s 1 \longrightarrow 1 \dashv \Gamma}\ \textsc{I-Unit}$$

Similar as in Part 1.

–

$$\frac{}{\Gamma[\alpha] \vdash^s \alpha \longrightarrow \alpha \dashv \Gamma[\alpha]}\ \textsc{I-TVar}$$

Similar as in Part 1.

–

$$\frac{}{\Gamma[\widehat{\alpha}][\widehat{\beta}] \vdash^s \widehat{\beta} \longrightarrow \widehat{\alpha}_1 \dashv \Gamma[\widehat{\alpha}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\alpha}_1]}\ \textsc{I-EVarAfter-Poly}$$

Similar as in Part 1.

–

$$\frac{}{\Gamma[\widehat{\beta}][\widehat{\alpha}] \vdash^s \widehat{\beta} \longrightarrow \widehat{\beta} \dashv \Gamma[\widehat{\beta}][\widehat{\alpha}]}\ \textsc{I-EVarBefore-Poly}$$

Similar as in Part 1.

□

Lemma 71 (Polymorphic Type Sanitization Completeness). *Given $\Gamma \longrightarrow \Omega$, where $\Gamma = \Gamma_1, \widehat{\alpha}, \Gamma_2$, and $[\Gamma](A) = A$, and $[\Omega](\tau) = \tau$, and $\widehat{\alpha} \notin FV(A)$, and $\Gamma_1 \vdash \tau$:*

- *If $[\Omega](\Gamma) \vdash \tau \leq [\Omega](A)$, then there are Δ, Ω' such that $\Omega \longrightarrow \Omega'$, and $\Delta \longrightarrow \Omega'$, and $\Gamma[\widehat{\alpha}] \vdash^- A \longrightarrow \sigma \dashv \Delta$, where $\Delta = \Delta_1, \widehat{\alpha}, \Delta_2$, and $\Delta_1 \vdash \sigma$, and $[\Omega'](\sigma) = \tau$.*
- *If $[\Omega](\Gamma) \vdash [\Omega](A) \leq \tau$, then there are Δ, Ω' such that $\Omega \longrightarrow \Omega'$, and $\Delta \longrightarrow \Omega'$, and $\Gamma[\widehat{\alpha}] \vdash^+ A \longrightarrow \sigma \dashv \Delta$, where $\Delta = \Delta_1, \widehat{\alpha}, \Delta_2$, and $\Delta_1 \vdash \sigma$, and $[\Omega'](\sigma) = \tau$.*

Proof. By induction on the height of subtyping derivation.

Part 1 We have $[\Omega](\Gamma) \vdash \tau \leq [\Omega](A)$. We now case analyze the shape of A.

– Case $A = \widehat{\beta}$.

$\widehat{\alpha} \notin FV(A)$	Given
$\widehat{\alpha} \neq \widehat{\beta}$	Follows directly
$[\Omega](\widehat{\beta}) = \tau_2$	Assume
$[\Omega](\Gamma) \vdash \tau \leq \tau_2$	Given
$\tau = \tau_2$	By Lemma 58
$[\Gamma](\widehat{\beta}) = \widehat{\beta}$	Given
$\widehat{\beta} \in unsolved(\Gamma)$	Follows directly

According to whether $\widehat{\beta}$ is on the left of $\widehat{\alpha}$ or right, we have two cases.

- SubCase $\Gamma = \Gamma_1, \widehat{\alpha}, \Gamma_3, \widehat{\beta}, \Gamma_4$.

$\Gamma_1, \widehat{\alpha}, \Gamma_3, \widehat{\beta}, \Gamma_4 \vdash^- \widehat{\beta} \longrightarrow \widehat{\alpha}_1 \dashv \Gamma_1, \widehat{\alpha}_1, \widehat{\alpha}, \Gamma_3, \widehat{\beta} = \widehat{\alpha}_1, \Gamma_4$ By I-EVarAfterPoly

$\Gamma_1, \widehat{\alpha}, \Gamma_3, \widehat{\beta}, \Gamma_4 \longrightarrow \Omega$	Given
$\Omega = \Omega[\widehat{\alpha} = \tau']$	Assume
$\Gamma_1, \widehat{\alpha}_1 = \tau, \widehat{\alpha}, \Gamma_3, \widehat{\beta}, \Gamma_4 \longrightarrow \Omega[\widehat{\alpha}_1 = \tau, \widehat{\alpha} = \tau']$	By Lemma 59
$[\Omega[\widehat{\alpha}_1 = \tau, \widehat{\alpha} = \tau']](\widehat{\beta}) = \tau$	Known
$\Gamma_1, \widehat{\alpha}_1 = \tau, \widehat{\alpha}, \Gamma_3, \widehat{\beta} = \widehat{\alpha}_1, \Gamma_4 \longrightarrow \Omega[\widehat{\alpha}_1 = \tau, \widehat{\alpha} = \tau']$	By Lemma 60
$\Gamma_1, \widehat{\alpha}_1, \widehat{\alpha}, \Gamma_3, \widehat{\beta} = \widehat{\alpha}_1, \Gamma_4 \longrightarrow \Gamma_1, \widehat{\alpha}_1 = \tau, \widehat{\alpha}, \Gamma_3, \widehat{\beta} = \widehat{\alpha}_1, \Gamma_4$	By Lemma 52
$\Gamma_1, \widehat{\alpha}_1, \widehat{\alpha}, \Gamma_3, \widehat{\beta} = \widehat{\alpha}_1, \Gamma_4 \longrightarrow \Omega[\widehat{\alpha}_1 = \tau, \widehat{\alpha} = \tau']$	By Lemma 53
$\Omega' = \Omega[\widehat{\alpha}_1 = \tau, \widehat{\alpha} = \tau']$	Choose
$\Omega[\widehat{\alpha} = \tau'] \longrightarrow \Omega[\widehat{\alpha}_1 = \tau, \widehat{\alpha} = \tau']$	By Lemma 61
$[\Omega'](\widehat{\alpha}_1) = \tau$	Follows directly

- SubCase $\Gamma = \Gamma_1, \widehat{\beta}, \Gamma_3, \widehat{\alpha}, \Gamma_4$.

$\Gamma_1, \widehat{\beta}, \Gamma_3, \widehat{\alpha}, \Gamma_4 \vdash^- \widehat{\beta} \longrightarrow \widehat{\beta} \dashv \Gamma_1, \widehat{\beta}, \Gamma_3, \widehat{\alpha}, \Gamma_4$	By I-EVarBeforePoly
$\Gamma_1, \widehat{\beta}, \Gamma_3, \widehat{\alpha}, \Gamma_4 \longrightarrow \Omega$	Given
$\Omega' = \Omega$	Choose
$\Omega \longrightarrow \Omega'$	By Lemma 54
$[\Omega'](\widehat{\beta}) = \tau$	Known

– Case $A = \alpha$.

$[\Omega](\alpha) = \alpha$	By definition of context application
$[\Omega](\Gamma) \vdash \tau \ \leq \ \alpha$	Given
$\tau = \alpha$	By inversion
$\Gamma_1 \vdash \alpha$	Given
α is declared to the left of $\widehat{\alpha}$ in Γ	
$\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash^- \alpha \longrightarrow \alpha \dashv \Gamma_1, \widehat{\alpha}, \Gamma_2$	By I-TVar
$\Omega' = \Omega$	Choose
$\Omega \longrightarrow \Omega'$	By Lemma 54
$[\Omega'](\alpha) = \alpha$	By definition of context application

– Case $A = 1$.

$[\Omega](1) = 1$	By definition of context application
$[\Omega](\Gamma) \vdash \tau \ \leq \ 1$	Given
$\tau = 1$	By inversion
$\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash^- 1 \longrightarrow 1 \dashv \Gamma_1, \widehat{\alpha}, \Gamma_2$	By I-Unit
$\Omega' = \Omega$	Choose
$\Omega \longrightarrow \Omega'$	By Lemma 54
$[\Omega'](\alpha) = \alpha$	By definition of context application

– Case $A = A_1 \to A_2$

$[\Omega](A_1 \to A_2) = [\Omega](A_1) \to [\Omega](A_2)$	By definition of context application
$[\Omega](\Gamma) \vdash \tau \ \leq \ [\Omega](A_1) \to [\Omega](A_2)$	Given
$\tau = \tau_1 \to \tau_2$	By inversion
$[\Omega](\Gamma) \vdash [\Omega](A_1) \ \leq \ \tau_1$	As above
$[\Omega](\Gamma) \vdash \tau_2 \ \leq \ [\Omega](A_2)$	As above
$\Gamma \vdash^+ A_1 \longrightarrow \sigma_1 \dashv \Theta$	By Part 2
$\Theta = \Theta_1, \widehat{\alpha}, \Theta_2$	As above
$\Theta_1 \vdash \sigma_1$	As above
$\Theta \longrightarrow \Omega_1$	As above
$\Omega \longrightarrow \Omega_1$	As above
$[\Omega_1](\sigma_1) = \tau_1$	As above
$\Gamma \longrightarrow \Theta$	By Lemma 69
$\Gamma_1 \longrightarrow \Theta_1$	By Lemma 62
$\Gamma \longrightarrow \Omega_1$	By Lemma 53
$[\Omega](\Omega) \vdash \tau_2 \ \leq \ [\Omega](A_2)$	By Lemma 64
$[\Omega_1](\Omega_1) \vdash \tau_2 \ \leq \ [\Omega](A_2)$	By Lemma 66
$[\Omega_1](\Gamma) \vdash \tau_2 \ \leq \ [\Omega](A_2)$	By Lemma 64
$[\Omega_1](\Theta) \vdash \tau_2 \ \leq \ [\Omega](A_2)$	By Lemma 55
$[\Omega_1](\Theta) \vdash \tau_2 \ \leq \ [\Omega_1](A_2)$	By Lemma 65
$[\Omega_1](\Theta) \vdash \tau_2 \ \leq \ [\Omega_1]([\Theta](A_2))$	By Lemma 56
$\Theta \vdash^- [\Theta](A_2) \longrightarrow \sigma_2 \dashv \Delta$	By induction hypothesis
$\Delta = \Delta_1, \widehat{\alpha}, \Delta_2$	As above
$\Delta \longrightarrow \Omega_2$	As above
$\Omega_1 \longrightarrow \Omega_2$	As above
$\Delta_1 \vdash \sigma_2$	As above
$[\Omega_2](\sigma_2) = \tau_2$	As above
$[\Omega_2](\sigma_1) = \tau_1$	By Lemma 65
$\Theta_1 \longrightarrow \Delta_1$	By Lemma 62
$\Delta_1 \vdash \sigma_1$	By Lemma 67
$\Delta_1 \vdash \sigma_1 \to \sigma_2$	Follows directly
$\Gamma \vdash^+ A_1 \longrightarrow \sigma_1 \dashv \Theta$	Known
$\Theta \vdash^- [\Theta](A_2) \longrightarrow \sigma_2 \dashv \Delta$	Known
$\Gamma \vdash^- A_1 \to A_2 \longrightarrow \sigma_1 \to \sigma_2 \dashv \Delta$	By I-Pi-Poly
$\Omega \longrightarrow \Omega_2$	By Lemma 53
$\Omega' = \Omega_2$	Choose
$[\Omega'](\sigma_1 \to \sigma_2) = \tau_1 \to \tau_2$	By definition of context application

– Case $A = \forall \alpha . A_1$

$[\Omega](\forall \alpha . A_1) = \forall \alpha . [\Omega](A_1)$	By definition of context application
$[\Omega](\Gamma) \vdash \tau \ \leq \ \forall \alpha . [\Omega](A_1)$	Given
$[\Omega](\Gamma), \alpha \vdash \tau \ \leq \ [\Omega](A_1)$	By inversion

$[\Omega, \alpha](\Gamma, \alpha) \vdash \tau \ \leq \ [\Omega, \alpha](A_1)$	Rewrite the judgment
$\Gamma, \alpha \vdash^- A_1 \longrightarrow \sigma \dashv \Theta$	By induction hypothesis
$\longrightarrow$ will not add new variables in the end	Follows from definition
$\Theta = \Theta_1, \widehat{\alpha}, \Theta_2, \alpha$	By induction hypothesis and Lemma 62 and above
$\Theta_1 \vdash \sigma$	By induction hypothesis
$\Theta \longrightarrow \Omega_1$	As above
$\Omega, \alpha \longrightarrow \Omega_1$	As above
$[\Omega_1](\sigma) = \tau$	As above
$\Omega_1 = \Omega_2, \alpha, \Omega_3$	By Lemma 62
$\Theta_1, \widehat{\alpha}, \Theta_2 \longrightarrow \Omega_2$	As above
$\Omega \longrightarrow \Omega_2$	As above
$\Theta_1, \widehat{\alpha}, \Theta_2 \vdash \sigma$	By Lemma 1
$\Omega_2 \vdash \sigma$	By Lemma 67
$[\Omega_1](\sigma) = [\Omega_2](\sigma) = \tau$	Follows directly
$\Gamma \vdash^- \forall \alpha. A_1 \longrightarrow \sigma \dashv \Theta_1, \widehat{\alpha}, \Theta_2$	By I-All-Minus
$\Omega' = \Omega_2$	Choose

Part 2 We have $[\Omega](\Gamma) \vdash [\Omega](A) \ \leq \ \tau$. We now case analyze the shape of A. These cases are mostly symmetric as in Part 1. The only exception is when A is a polymorphic type.

– Case $A = \forall \alpha. A_1$

$[\Omega](\forall \alpha. A_1) = \forall \alpha. [\Omega](A_1)$	By definition of context application
$[\Omega](\Gamma) \vdash \forall \alpha. [\Omega](A_1) \ \leq \ \tau$	Given
$[\Omega](\Gamma) \vdash ([\Omega](A_1))[\alpha \mapsto \tau_1] \ \leq \ \tau$	By inversion
$[\Omega](\Gamma) \vdash \tau_1$	Same as above
τ_1 contains no existential variable	Follows directly
$[\Omega](\tau_1) = \tau_1$	Follows directly
$[\Omega](\Gamma) \vdash [\Omega](A_1[\alpha \mapsto \tau_1]) \ \leq \ \tau$	Rewrite the judgment
$[\Omega](\Gamma) \vdash [\Omega](A_1[\alpha \mapsto \widehat{\beta}][\widehat{\beta} \mapsto \tau_1]) \ \leq \ \tau$	Rewrite the judgment
$\Gamma = \Gamma_1, \widehat{\alpha}, \Gamma_2$	Given
$\Gamma \longrightarrow \Omega$	Given
$\Omega = \Omega[\widehat{\alpha} = \tau']$	By Lemma 62
$[\Omega[\widehat{\beta} = \tau_1, \widehat{\alpha} = \tau']](\Gamma_1, \widehat{\beta}, \widehat{\alpha}, \Gamma_2) \vdash [\Omega[\widehat{\beta} = \tau_1, \widehat{\alpha} = \tau']](A_1[\alpha \mapsto \widehat{\beta}]) \ \leq \ \tau$	Rewrite the judgment
$\Gamma_1, \widehat{\beta}, \widehat{\alpha}, \Gamma_2 \vdash^+ A_1[\alpha \mapsto \widehat{\beta}] \longrightarrow \sigma \dashv \Delta$	By induction hypothesis
$\Delta = \Delta_1, \widehat{\alpha}, \Delta_2$	As above
$\Delta_1 \vdash \sigma$	As above
$\Delta \longrightarrow \Omega_1$	As above
$\Omega[\widehat{\beta} = \tau_1, \widehat{\alpha} = \tau'] \longrightarrow \Omega_1$	As above
$[\Omega_1](\sigma) = \tau$	As above
$\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash^+ \forall \alpha. A_1 \longrightarrow \sigma \dashv \Delta$	By I-All-Plus
$\Omega' = \Omega_1$	Choose

$\Omega[\widehat{\alpha} = \tau'] \longrightarrow \Omega[\widehat{\beta} = \tau_1, \widehat{\alpha} = \tau']$	By Lemma 61
$\Omega \longrightarrow \Omega_1$	By Lemma 53

□

Corollary 5 (Polymorphic Type Sanitization Completeness w.r.t Subtyping). *Given* $\Gamma \longrightarrow \Omega$, *and* $[\Gamma](A) = A$, *and* $\widehat{\alpha} \notin FV(A)$, *and* $\widehat{\alpha} \in unsolved(\Gamma)$ *:*

- *If* $[\Omega](\Gamma) \vdash [\Omega](\widehat{\alpha}) \ \leq \ [\Omega](A)$, *then there are* Δ, Ω' *such that* $\Omega \longrightarrow \Omega'$, *and* $\Delta \longrightarrow \Omega'$, *and* $\Gamma[\widehat{\alpha}] \vdash \widehat{\alpha} \ <: \ A \dashv \Delta$.
- *If* $[\Omega](\Gamma) \vdash [\Omega](A) \ \leq \ [\Omega](\widehat{\alpha})$, *then there are* Δ, Ω' *such that* $\Omega \longrightarrow \Omega'$, *and* $\Delta \longrightarrow \Omega'$, *and* $\Gamma[\widehat{\alpha}] \vdash A \ <: \ \widehat{\alpha} \dashv \Delta$.

Part 1

$[\Omega](\widehat{\alpha}) = \tau$	Assume
$[\Omega](\tau) = \tau$	Since τ contains no existential variables
$\widehat{\alpha} \in unsolved(\Gamma)$	Given
$\Gamma = \Gamma_1, \widehat{\alpha}, \Gamma_2$	Assume
$\Omega = \Omega_1, \widehat{\alpha} = \tau', \Omega_2$	By Lemma 62
$\Gamma_1 \longrightarrow \Omega_1$	Same as above
$[\Omega](\widehat{\alpha}) = [\Omega_1](\widehat{\alpha}) = \tau$	By definition of context application
$\Omega_1 \vdash \tau$	By Lemma 68
Since τ contains no existential variables	
And Γ_1 contains all type variables in Ω_1	
$\Gamma_1 \vdash \tau$	Follows directly
$[\Omega](\Gamma) \vdash \tau \ \leq \ [\Omega](A)$	Given
$\Gamma[\widehat{\alpha}] \vdash^{-} A \longrightarrow \sigma \dashv \Delta$	By Lemma 71
$\Delta = \Delta_1, \widehat{\alpha}, \Delta_2$	As above
$\Delta_1 \vdash \sigma$	As above
$\Delta \longrightarrow \Omega'$	As above
$\Omega \longrightarrow \Omega'$	As above
$[\Omega'](\sigma) = \tau$	As above
$[\Omega'](\widehat{\alpha}) = \tau$	By Lemma 65
$\Gamma[\widehat{\alpha}] \vdash \widehat{\alpha} \ <: \ A \dashv \Delta_1, \widehat{\alpha} = \sigma, \Delta_2$	By $<:$ InstL
$\Delta_1, \widehat{\alpha} = \sigma, \Delta_2 \longrightarrow \Omega'$	By Lemma 60

Part 2 Similar as Part 1.

□