2013

A Type- and Control-Flow Analysis for System F: Technical Report

Matthew Fluet

A Type- and Control-Flow Analysis for System F
Technical Report

Matthew Fluet

Computer Science Department
Rochester Institute of Technology, Rochester NY 14623
mtf@cs.rit.edu

Abstract. We present a monovariant flow analysis for System F (with recursion). The flow analysis yields both *control-flow* information, approximating the λ- and Λ-expressions that may be bound to variables, and *type-flow* information, approximating the type expressions that may instantiate type variables. Moreover, the two flows are mutually beneficial: the control flow determines which Λ-expressions may be applied to which type expressions (and, hence, which type expressions may instantiate which type variables), while the type flow filters the λ- and Λ-expressions that may be bound to variables (by rejecting expressions with static types that are incompatible with the static type of the variable under the type flow). As is typical for a monovariant control-flow analysis, control-flow information is expressed as an abstract environment mapping variables to sets of (syntactic) λ- and Λ-expressions that occur in the program under analysis. Similarly, type-flow information is expressed as an abstract environment mapping type variables to sets of (syntactic) types that occur in the program under analysis. Compatibility of static types (with free type variables) under a type flow is decided by interpreting the abstract environment as productions for a *regular-tree grammar* and querying if the languages generated by taking the types in question as starting terms have a non-empty intersection.
This is a companion technical report, providing additional commentary and proof details, to a paper [11] appearing in *Implementation and Application of Functional Languages: 24th International Symposium (IFL'12)*.

1 Introduction

Control-flow analysis is an important enabling technology for the compilation and optimization of functional languages. Because functional languages have first-class functions, the control flow of a functional program is not syntactically apparent: in an application expression, the function can itself be the result of a computation and may not be available until run time. Indeed, during the execution of a program, many different functions may be applied at the same (source-program) application expression. A control-flow analysis [21,48,47,34,27] approximates, at compile time, the flow of first-class functions in a program: which first-class functions might be bound to a given variable or returned by a

given expression at run time. This approximate control-flow information can be used to enable optimizations of a functional language.

Control-flow analyses have typically been formulated for *dynamically-* or *simply-*typed functional languages.[1] However, most statically-typed functional languages have rich type systems that include polymorphic types. Indeed, System F [13,41], the polymorphic lambda calculus, and extensions thereof are commonly used as typed intermediate languages in compilers for functional languages [52,36,50]. Typed intermediate languages provide a number of benefits. First, explicit type information can support type-dependent optimizations, such as using a specialized representation for known types rather than a universal representation. Second, explicit type information can support validation of optimizations, by detecting when an optimization transforms a well-typed program to an ill-typed program. Since optimizations are performed on a typed intermediate language and optimizations are enabled by control-flow analyses, it is natural to seek a control-flow analysis that is formulated for System F.

While one could naïvely adopt an existing control-flow analysis that is formulated for a dynamically- or simply-typed functional language and ignore the System F features of type abstraction and type application, such an approach fails to take advantage of the static information provided by a well-typed program. Intuitively, a control-flow analysis for System F should exploit the well-typedness of the program under analysis in order to obtain more precise control-flow information. For instance, if a control-flow analysis asserts that a variable x might be bound to a function of type $\mathsf{int} \to \mathsf{int}$, a function of type $\mathsf{bool} \to \mathsf{bool}$, and a function of type $\mathsf{string} \to \mathsf{string}$ (and no other functions), but the static type of x is $\mathsf{int} \to \mathsf{int}$, then the type soundness of the language guarantees that x will only be bound to functions of type $\mathsf{int} \to \mathsf{int}$ at run time and the control-flow result may be soundly refined to assert that x might only be bound to the function of type $\mathsf{int} \to \mathsf{int}$. However, if the static type of x is $\alpha \to \alpha$ (where the type variable α is bound by a type abstraction in the program under analysis), then it is unclear whether or not the control-flow result may be soundly refined, because the type variable α may be soundly instantiated at any type.

Given additional information that asserts that the type variable α might be instantiated at the type int and the type bool (and no other types), then the control-flow result may be soundly refined to assert that x might only be bound to the function of type $\mathsf{int} \to \mathsf{int}$ and the function of type $\mathsf{bool} \to \mathsf{bool}$. This additional information may be obtained by a *type-flow analysis* that approximates, at compile time, the flow of types in a program: which types might instantiate a given type variable at run time. As demonstrated by the example above, this approximate type-flow information can be used to increase the precision of a control-flow analysis. Furthermore, this approximate type-flow information can be used to enable type-dependent optimizations, such as guiding the specializa-

[1] Although there are many *type-based* [35] control-flow analyses, where the analyses are expressed as a sophisticated type systems (e.g., type-and-effect systems [9,33,19], type systems with polymorphic types [39], type systems with union/intersection types [54,31]), the language under analysis is typically a simply-typed language.

tion of a polymorphic function that is used at a small number of distinct types or eliminating type operations in a language with intensional polymorphism [14]. Just as a control-flow analysis yields useful information because, for a given program, it is unlikely that a given variable is bound to *every* function during execution, a type-flow analysis yields useful information because it is unlikely that a given type variable is instantiated at *every* type during execution.

Although type-flow information and control-flow information might be obtained by independent analyses, the two kinds of information can be mutually beneficial, particularly for the higher-rank impredicative polymorphism of System F. Control-flow analysis supports type-flow analysis by yielding information about the type abstractions that may be applied at type applications and, hence, about the types at which type variables may be instantiated. The type-flow information soundly refines the control-flow information by rejecting flows that are incompatible with the static typing of the program under analysis; because the static typing may be expressed in terms of syntactic types with free type variables, the type-flow information is used to determine the compatibility of types. When the type-flow information refines the control-flow information by rejecting the flow of a type abstraction, the type-flow information itself may be refined because the type abstraction may be applied at fewer type applications, and, hence, there may be fewer types at which the type variable may be instantiated.[2]

In a combined type- and control-flow analysis, the type-flow information soundly refines the control-flow information by determining when types are incompatible. In the presence of recursion and higher-rank impredicative polymorphism, the type-flow information must approximate complex relationships between type variables and types and the compatibility or incompatibility of types under the type-flow information may not be obvious. Indeed, during the execution of a program that is well-typed in System F with recursion, a type variable may be instantiated at an infinite number of types. In order to obtain a computable analysis, the type-flow information must use a finite representation that approximates the (potentially infinite) set of closed types that may instantiate a type variable and the compatibility of types under the type-flow information must be a decidable property.

Most control-flow analyses approximate the (potentially infinite) set of first-class functions that might be bound to a variable at run time by a (necessarily finite) set of function expressions (possibly with free variables) that occur in the program under analysis. Similarly, a type-flow analysis may approximate the (potentially infinite) set of closed types that may instantiate a type variable at run time by a (necessarily finite) set of type expressions (possibly with free type variables) that occur in the program under analysis. For instance, if a type-flow analysis asserts that a type variable α might be instantiated at the type expression $\mathsf{int} \to \mathsf{int}$ and the type expression $\mathsf{int} \to \alpha$ (and no other type

[2] In practice, though, a flow analysis is computed by adding information that is consistent with existing information (i.e., ascending a lattice) rather than removing information that is inconsistent with existing information (i.e., descending a lattice).

expressions), then, by interpreting the type-flow information as productions for a *regular-tree grammar* [12,2,7], the type-flow analysis may be seen to be asserting that the type variable α might be instantiated at the infinite set of closed types $\{\mathsf{int} \to \mathsf{int}, \mathsf{int} \to \mathsf{int} \to \mathsf{int}, \mathsf{int} \to \mathsf{int} \to \mathsf{int} \to \mathsf{int}, \ldots\}$. Furthermore, if the type-flow analysis asserts that a type variable β might be instantiated at the type expression $\mathsf{int} \to \mathsf{bool}$ and the type expression $\mathsf{int} \to \beta$ (and no other type expressions) and a control-flow analysis asserts that a variable x might be bound to a function of type $\mathsf{int} \to \mathsf{int}$, a function of type $\mathsf{bool} \to \mathsf{int}$, a function of type $\mathsf{string} \to \mathsf{int}$, a function of type $\mathsf{int} \to \alpha$, and a function of type $\mathsf{int} \to \beta$ (and no other functions), but the static type of x is α, then the control-flow result may be soundly refined to assert that x might only be bound to the function of type $\mathsf{int} \to \mathsf{int}$ and the function of type $\mathsf{int} \to \alpha$, because the types of these two functions are compatible with the type α (under the type-flow information), while the types of the other three functions are incompatible with the type α.

Two types are compatible (under the type-flow information) if there exists a closed type that is a member of the sets of closed types at which the types might be instantiated; conversely, two types are incompatible if there does not exist a closed type that is a member of the sets of closed types at which the types might be instantiated. The type soundness of the language guarantees that a variable will only be bound to a well-typed closed function of a closed type at run time; hence, if there is no closed type at which both the static type of a variable and the static type of a function might be instantiated, then that variable will never be bound to that function at run time. For example, the types $\mathsf{int} \to \alpha$ and α are compatible because the type $\mathsf{int} \to \alpha$, interpreted as a starting term for the regular-tree grammar corresponding to the type-flow information, represents the infinite set of closed types $\{\mathsf{int} \to \mathsf{int} \to \mathsf{int}, \mathsf{int} \to \mathsf{int} \to \mathsf{int} \to \mathsf{int}, \ldots\}$, which has a non-empty intersection with the infinite set of closed types that might instantiate the type variable α (given above). Similarly, the types $\mathsf{int} \to \beta$ and α are incompatible because the type $\mathsf{int} \to \beta$ represents the infinite set of closed types $\{\mathsf{int} \to \mathsf{int} \to \mathsf{bool}, \mathsf{int} \to \mathsf{int} \to \mathsf{int} \to \mathsf{bool}, \ldots\}$, which has an empty intersection with the infinite set of closed types that might instantiate the type variable α. Since regular-tree grammars are closed under intersection and the emptiness of a regular-tree grammar is decidable, the compatibility of types under the type-flow information is a decidable property.

Overview We present a monovariant[3] type- and control-flow analysis for System F extended with recursive functions. Our flow analysis is a variation on 0CFA, the classic monovariant control-flow analysis [34]. For a given program, the flow analysis computes an abstract environment that maps variables to (finite) sets of λ- and Λ-expressions that occur in the program and maps type variables to (finite) sets of type expressions that occur in the program.

Our formulation of the type- and control-flow analysis as a refinement of the syntax-directed constraint-based formulation of 0CFA establishes that the

[3] i.e., context insensitive

$$
\begin{array}{lrl}
\textit{Types} & \textit{Type} \ni \tau & ::= \tau_a \to \tau_b \mid \alpha \mid \forall\alpha.\ \tau_b \\
\textit{Type variables} & \textit{TyVar} \ni \alpha, \beta, \ldots & \\
 & & \\
\textit{Expressions} & \textit{Exp} \ni e & ::= x \mid \texttt{let}\ x{:}\tau_x\ \texttt{=}\ b\ \texttt{in}\ e \\
\textit{Binds} & \textit{Bnd} \ni b & ::= \mu f{:}\tau_f\,.\,\lambda z{:}\tau_z\,.\,e_b \mid x_f\ x_a \mid \\
 & & \quad\ \mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b \mid x_f\ [\tau_a] \\
\textit{Variables} & \textit{Var} \ni x, y, z, f, g, \ldots & \\
 & & \\
 & & \lfloor\cdot\rfloor :: \textit{Exp} \to \textit{Var} \\
 & & \lfloor x \rfloor = x \\
 & \lfloor \texttt{let}\ x{:}\tau_x\ \texttt{=}\ b\ \texttt{in}\ e \rfloor & = \lfloor e \rfloor
\end{array}
$$

Fig. 1. Syntax of ANF System F

combined type- and control-flow analysis can be more precise than 0CFA. Although not as precise as a type-directed polyvariant[4] control-flow analysis [20], our monovariant type- and control-flow analysis nonetheless rejects some similar classes of spurious flows and, furthermore, has the benefits of handling full (i.e., impredicative) System F and terminating for all well-typed programs.

Soundness of the analysis is proven with respect to an operational semantics for System F given in the style of the administrative-normal-form (ANF) environment- and continuation-based C_aEK abstract machine [10], where the (concrete) environment component of the abstract machine maps variables to closures (pairs of λ- or Λ-expressions and an environment, which captures the free variables and type variables of the λ- or Λ-expression) and maps type variables to *type closures* (pairs of type expressions and an environment, which captures the free type variables of the type expression). A sound flow analysis computes an abstract environment that approximates every concrete environment that arises during evaluation. We present the analysis-time type-compatibility predicate as a judgment; this yields a declarative specification of type compatibility, for which the regular-tree-grammar interpretation gives an algorithmic implementation.

2 Language and Semantics

Our source language is a variant of System F, extended with recursive functions and presented in (a restriction of) administrative normal form (ANF). The operational semantics of the language is presented as an abstract machine. The static semantics of the language is entirely standard, but given for completeness.

2.1 Syntax

The syntax of our ANF variant of System F is given in Figure 1.

[4] i.e., context sensitive

Types include function types, type variables, and universal types; in the universal type $\forall\alpha.\ \tau_b$, the type variable α is bound in the type τ_b. Type equality is syntactic identity (up to α-conversion of bound type variables).

Expressions include variables, `let`-bindings of recursive functions, `let`-bindings of non-tail function applications, `let`-bindings of recursive type abstractions, and `let`-bindings of non-tail type applications. In the `let`-binding expression `let` $x{:}\tau_x$ `=` b `in` e, the variable x is bound in e; in the recursive function $\mu f{:}\tau_f\,.\,\lambda z{:}\tau_z\,.\,e_b$, the variables f and z are bound in the expression e_b and in the recursive type abstraction $\mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b$, the variable f and the type variable β are bound in the expression e_b. Programs are (closed, well-typed) expressions. Finally, we define a function $\lfloor\cdot\rfloor$ on expressions, which extracts the variable that yields the expression's value.

The language is Church-style, in which every bound variable is annotated with its type. In contrast to some presentations of ANF-like languages [42,8,10] but in concert with some others [51,30,53,4], we restrict the constituents of function applications and type applications to variables, rather than allowing a larger class of "trivial" expressions, and we restrict function applications and type applications to non-tail calls, rather than allowing tail calls. Neither restriction is essential for the forthcoming type- and control-flow analysis; we adopt them simply to reduce the number of inference rules and helper functionsn the operational semantics, static semantics, and flow analysis.

We do not assume that all `let`-, μ-, and λ-bound variables and all Λ-bound type variables in a program are distinct, although the static semantics will prohibit shadowing of variables and type variables within the same scope.

2.2 Operational Semantics

The operational semantics for our ANF-variant of System F is presented as an adaptation of the environment- and continuation-based C_aEK machine [10] (and is similar in some ways to the $\lambda_{\mathrm{gc}}^{\rightarrow\forall}$ abstract machine [30]) and is given in Figure 2.

A machine state ς has four components: a control expression, a run-time type environment, a run-time value environment, and a continuation.

A run-time type environment ϕ is a map from type variables to run-time types and a run-time value environment ρ is a map from variables to run-time values. A run-time type π is a "type closure": a pair of a (possibly open) type and a run-time type environment; the run-time type environment captures the free type variables of the type. A run-time value w is a "function closure" or a "type-abstraction closure": a triple of a (possibly open) value (a recursive function or a recursive type abstraction), a run-time type environment (that captures the free type variables of the value), and a run-time value environment (that captures the free variables of the value).

A continuation κ is a stack of frames, each of the form $\langle x; \tau_x; \phi; \rho; e\rangle$, where x is the variable receiving the result w of a non-tail function application or non-tail type application, τ_x is the (static, syntactic) type of x, and e is the expression to

$$
\begin{array}{lrl}
\textit{Run-time types} & \textit{RType} \ni \pi & ::= \langle \tau ; \phi \rangle \\
\textit{Run-time type environments} & \textit{RTEnv} \ni \phi & ::= \bullet \mid \phi, \alpha \mapsto \pi \\
\\
\textit{Run-time values} & \textit{RValue} \ni w & ::= \langle \mu f\!:\!\tau_f \,.\, \lambda z\!:\!\tau_z \,.\, e_b ; \phi ; \rho \rangle \mid \\
 & & \quad \langle \mu f\!:\!\tau_f \,.\, \Lambda \beta \,.\, e_b ; \phi ; \rho \rangle \\
\textit{Run-time value environments} & \textit{RVEnv} \ni \rho & ::= \bullet \mid \rho, x \mapsto w \\
\\
\textit{Continuations} & \textit{Kont} \ni \kappa & ::= \circ \mid \langle x ; \tau_x ; e ; \phi ; \rho \rangle :: \kappa \\
\textit{States} & \textit{State} \ni \varsigma & ::= \langle e ; \phi ; \rho ; \kappa \rangle
\end{array}
$$

$\boxed{\varsigma \longrightarrow \varsigma}$

$$\frac{\rho_r(x_r) = w_r}{\langle x_r ; \phi_r ; \rho_r ; \langle z ; \tau_z ; e ; \phi ; \rho \rangle :: \kappa \rangle \longrightarrow \langle e ; \phi ; \rho, z \mapsto w_r ; \kappa \rangle}$$

$$\frac{w_r = \langle \mu f\!:\!\tau_f \,.\, \lambda z\!:\!\tau_z \,.\, e_b ; \phi ; \rho \rangle}{\langle \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f \,.\, \lambda z\!:\!\tau_z \,.\, e_b\ \texttt{in}\ e ; \phi ; \rho ; \kappa \rangle \longrightarrow \langle e ; \phi ; \rho, x \mapsto w_r ; \kappa \rangle}$$

$$\frac{w_r = \langle \mu f\!:\!\tau_f \,.\, \Lambda \beta \,.\, e_b ; \phi ; \rho \rangle}{\langle \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f \,.\, \Lambda \beta \,.\, e_b\ \texttt{in}\ e ; \phi ; \rho ; \kappa \rangle \longrightarrow \langle e ; \phi ; \rho, x \mapsto w_r ; \kappa \rangle}$$

$$\frac{\rho(x_f) = w_f \qquad w_f = \langle \mu f\!:\!\tau_f \,.\, \lambda z\!:\!\tau_z \,.\, e_b ; \phi_f ; \rho_f \rangle \qquad \rho(x_a) = w_a}{\langle \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e ; \phi ; \rho ; \kappa \rangle \longrightarrow \langle e_b ; \phi_f ; \rho_f, f \mapsto w_f, z \mapsto w_a ; \langle x ; \tau_x ; e ; \phi ; \rho \rangle :: \kappa \rangle}$$

$$\frac{\rho(x_f) = w_f \qquad w_f = \langle \mu f\!:\!\tau_f \,.\, \Lambda \beta \,.\, e_b ; \phi_f ; \rho_f \rangle \qquad \pi_a = \langle \tau_a ; \phi \rangle}{\langle \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ x_f\ \texttt{[}\tau_a\texttt{]}\ \texttt{in}\ e ; \phi ; \rho ; \kappa \rangle \longrightarrow \langle e_b ; \phi_f, \beta \mapsto \pi_a ; \rho_f, f \mapsto w_f ; \langle x ; \tau_x ; e ; \phi ; \rho \rangle :: \kappa \rangle}$$

Fig. 2. Operational Semantics of ANF System F

be evaluated in the environments ϕ and ρ extended with x bound to w to yield the result of the frame.

The machine transition rules are straightforward. The first rule returns a result to the top-most frame of the continuation when the control expression has been reduced to a variable. The second and third rules create function closures and type-abstraction closures. The fourth and fifth rules extract the expression body, run-time type environment, and run-time value environment from an applied function closure or type-abstraction closure, extend the closure's run-time value environment with f bound to the closure (making the recursive function or recursive type-abstraction available to the expression body), extend the closure's run-time value environment with the run-time value argument (in the case of a function application) or extend the closure's run-time type environment with the run-time type argument (in the case of a type application), and push a frame onto the continuation to receive the result of the function application or type application. Note that the machine transitions are syntax directed and deterministic.

$$\begin{array}{ll} \textit{Type-variable contexts} & \textit{TCtxt} \ni \Delta ::= \bullet \mid \Delta, \alpha{:}\star \\ \textit{Variable contexts} & \textit{VCtxt} \ni \Gamma ::= \bullet \mid \Gamma, x{:}\tau \end{array}$$

$\boxed{\vdash \Delta}$

$$\frac{}{\vdash \bullet} \qquad \frac{\vdash \Delta \qquad \alpha \notin \mathsf{dom}(\Delta)}{\vdash \Delta, \alpha{:}\star}$$

$\boxed{\Delta \vdash \tau}$

$$\frac{\Delta \vdash \tau_a \qquad \Delta \vdash \tau_b}{\Delta \vdash \tau_a \to \tau_b} \qquad \frac{\vdash \Delta \qquad \Delta(\alpha) = \star}{\Delta \vdash \alpha} \qquad \frac{\Delta, \alpha{:}\star \vdash \tau_b}{\Delta \vdash \forall \alpha.\ \tau_b}$$

$\boxed{\Delta \vdash \Gamma}$

$$\frac{}{\Delta \vdash \bullet} \qquad \frac{\Delta \vdash \Gamma \qquad \Delta \vdash \tau \qquad x \notin \mathsf{dom}(\Gamma)}{\Delta \vdash \Gamma, x{:}\tau}$$

$\boxed{\Delta; \Gamma \vdash e : \tau}$

$$\frac{\Delta \vdash \Gamma \qquad \Gamma(x) = \tau}{\Delta; \Gamma \vdash x : \tau} \qquad \frac{\Delta \vdash \tau_x \qquad \Delta; \Gamma \vdash b : \tau_r \qquad \tau_x = \tau_r \qquad \Delta; \Gamma, x{:}\tau_x \vdash e : \tau}{\Delta; \Gamma \vdash \mathtt{let}\ x{:}\tau_x\ \mathtt{=}\ b\ \mathtt{in}\ e : \tau}$$

$\boxed{\Delta; \Gamma \vdash b : \tau}$

$$\frac{\begin{array}{c}\Delta \vdash \Gamma \qquad \Delta \vdash \tau_f \qquad \Delta \vdash \tau_z \qquad \Delta; \Gamma, f{:}\tau_f, z{:}\tau_z \vdash e_b : \tau_b \\ \tau_f = \tau_a \to \tau_b \qquad \tau_z = \tau_a\end{array}}{\Delta; \Gamma \vdash \mu f{:}\tau_f\,.\,\lambda z{:}\tau_z\,.\,e_b : \tau_a \to \tau_b}$$

$$\frac{\begin{array}{c}\Delta \vdash \Gamma \qquad \Delta \vdash \tau_f \qquad \Delta, \beta{:}\star; \Gamma, f{:}\tau_f \vdash e_b : \tau_b \\ \tau_f = \forall \beta.\ \tau_b\end{array}}{\Delta; \Gamma \vdash \mu f{:}\tau_f\,.\,\Lambda \beta\,.\,e_b : \forall \beta.\ \tau_b}$$

$$\frac{\Delta \vdash \Gamma \qquad \Gamma(x_f) = \tau_a \to \tau_b \qquad \Gamma(x_a) = \tau_a}{\Delta; \Gamma \vdash x_f\ x_a : \tau_b} \qquad \frac{\Delta \vdash \Gamma \qquad \Gamma(x_f) = \forall \alpha.\ \tau_b \qquad \Delta \vdash \tau_a}{\Delta; \Gamma \vdash x_f\ [\tau_a] : [\alpha \rightarrowtail \tau_a]\tau_b}$$

Fig. 3. Static Semantics of ANF System F

2.3 Static Semantics

The standard static semantics for System F, adapted to our ANF variant, is given in Figure 3. A type-variable context Δ records free type variables, the judgment $\vdash \Delta$ asserts that the type-variable context Δ is well-formed (all type variables are distinct), and the judgment $\Delta \vdash \tau$ asserts that the type τ is well-

formed in Δ. A variable context Γ records free variables and their types, the judgment $\Delta \vdash \Gamma$ asserts that the variable context Γ is well-formed in Δ (all variables are distinct and all types are well-formed in Δ), and the judgments $\Delta; \Gamma \vdash e : \tau$ and $\Delta; \Gamma \vdash b : \tau$ assert that the expression e and bind b have the type τ in Δ and Γ; in the rule for type applications, we write $[\alpha \rightarrowtail \tau_a]\tau_b$ for the capture-avoiding substitution of τ_a for free occurrences of α in τ_b. All judgments are designed to require, directly or indirectly, that their type-variable contexts and variable contexts are well-formed.

2.4 Type Soundness

A syntactic proof [55] of type soundness, using entirely standard Progress and Preservation theorems, is given in Appendix A.

Theorem 1 (Type Soundness).
If $\bullet; \bullet \vdash e : \tau$ *and* $\langle e; \bullet; \bullet; \circ \rangle \longrightarrow^* \varsigma'$,
then either there exists x', ϕ', *and* ρ' *such that* $\varsigma' = \langle x'; \phi'; \rho'; \circ \rangle$
or there exists ς' *such that* $\varsigma \longrightarrow \varsigma'$.

In addition to the judgments of Figure 3, we introduce judgments that assert the "well-typedness" of run-time types ($\vdash \pi \Rrightarrow \tau$), run-time type environments ($\vdash \phi : \theta$), run-time values ($\vdash w : \tau$), run-time value environments ($\vdash \rho : \Gamma$), continuations ($\vdash \tau \rightsquigarrow \kappa : \tau$), and states ($\vdash \varsigma : \tau$); see Figure 6 in Appendix A. Of note is the judgment $\vdash \phi : \theta$ that asserts that the run-time type environment ϕ corresponds to a substitution θ; the domains of ϕ and θ are equal, but whereas ϕ maps a type variable to a run-time type (a pair of a (possibly open) type and a (closing) run-time type environment), θ maps a type variable to a closed type obtained by (recursively) expanding the (possibly open) type by its (closing) run-time type environment.

3 Type- And Control-Flow Analysis

Our type- and control-flow analysis is presented as an adaptation of the syntax-directed 0CFA, the classic monovariant control-flow analysis [34, Section 3.3], and is given in Figure 4.

The result of our type- and control-flow analysis is a pair of abstract environments. An abstract type environment $\hat{\phi}$ is a map from type variables to sets of abstract types, where an abstract type is simply a (possibly open) type.[5] An abstract value environment $\hat{\rho}$ is a map from variables to sets of abstract values, where an abstract value is simply a (possibly open) recursive function or

[5] We introduce abstract types in preparation for future extensions of the analysis; for example, we may wish to introduce a $\top$ abstract type to represent an unknown type coming from outside the scope of the analysis.

$$
\begin{array}{lrl}
\textit{Abstract types} & \textit{AType} \ni \hat{\pi} & ::= \tau \\
\textit{Sets of abstract types} & \mathcal{P}(\textit{AType}) \ni \hat{\Pi} & \\
\textit{Abstract type environments} & \textit{ATEnv} \ni \hat{\phi} & \in \textit{TyVar} \to \mathcal{P}(\textit{AType}) \\
\\
\textit{Abstract values} & \textit{AValue} \ni \hat{w} & ::= \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b \mid \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b \\
\textit{Sets of abstract values} & \mathcal{P}(\textit{AValue}) \ni \hat{W} & \\
\textit{Abstract value environments} & \textit{AVEnv} \ni \hat{\rho} & \in \textit{Var} \to \mathcal{P}(\textit{AValue})
\end{array}
$$

$$
\begin{array}{ll}
\hat{\phi}_1 \sqsubseteq \hat{\phi}_2 \stackrel{\text{def}}{=} \forall \alpha \in \textit{TyVar}.\hat{\phi}_1(\alpha) \subseteq \hat{\phi}_2(\alpha) & \hat{\rho}_1 \sqsubseteq \hat{\rho}_2 \stackrel{\text{def}}{=} \forall x \in \textit{Var}.\hat{\rho}_1(x) \subseteq \hat{\rho}_2(x) \\
\left(\bigsqcup_{i\in I} \hat{\phi}_i\right)(\alpha) \stackrel{\text{def}}{=} \bigcup_{i\in I} \hat{\phi}_i(\alpha) & \left(\bigsqcup_{i\in I} \hat{\rho}_i\right)(x) \stackrel{\text{def}}{=} \bigcup_{i\in I} \hat{\rho}_i(x) \\
\left(\bigsqcap_{i\in I} \hat{\phi}_i\right)(\alpha) \stackrel{\text{def}}{=} \bigcap_{i\in I} \hat{\phi}_i(\alpha) & \left(\bigsqcap_{i\in I} \hat{\rho}_i\right)(x) \stackrel{\text{def}}{=} \bigcap_{i\in I} \hat{\rho}_i(x) \\
\hat{\phi}_\bot(\alpha) \stackrel{\text{def}}{=} \{\} & \hat{\rho}_\bot(x) \stackrel{\text{def}}{=} \{\} \\
\hat{\phi}_\top(\alpha) \stackrel{\text{def}}{=} \textit{AType} & \hat{\rho}_\top(x) \stackrel{\text{def}}{=} \textit{AValue}
\end{array}
$$

$$
\langle \hat{\phi}_1, \hat{\rho}_1 \rangle \sqsubseteq \langle \hat{\phi}_2, \hat{\rho}_2 \rangle \stackrel{\text{def}}{=} \hat{\phi}_1 \sqsubseteq \hat{\phi}_2 \wedge \hat{\rho}_1 \sqsubseteq \hat{\rho}_2
$$

$\boxed{\hat{\phi}; \hat{\rho} \vDash e}$

$$
\frac{}{\hat{\phi}; \hat{\rho} \vDash x}
\qquad
\frac{\hat{\phi}; \hat{\rho} \vDash b \rightsquigarrow \hat{W}_r \qquad \{\hat{w}_r \in \hat{W}_r \mid \hat{\phi} \vdash \hat{w}_r :\approx\!\!\gg \tau_x\} \subseteq \hat{\rho}(x) \qquad \hat{\phi}; \hat{\rho} \vDash e}{\hat{\phi}; \hat{\rho} \vDash \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ b\ \texttt{in}\ e}
$$

$\boxed{\hat{\phi}; \hat{\rho} \vDash b \rightsquigarrow \hat{W}}$

$$
\frac{\{\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\} \subseteq \hat{\rho}(f) \qquad \hat{\phi}; \hat{\rho} \vDash e_b \qquad \{\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\} \subseteq \hat{W}}{\hat{\phi}; \hat{\rho} \vDash \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b \rightsquigarrow \hat{W}}
\qquad
\frac{\{\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\} \subseteq \hat{\rho}(f) \qquad \hat{\phi}; \hat{\rho} \vDash e_b \qquad \{\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\} \subseteq \hat{W}}{\hat{\phi}; \hat{\rho} \vDash \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b \rightsquigarrow \hat{W}}
$$

$$
\frac{\bigwedge_{\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b \in \hat{\rho}(x_f)} \left(\{\hat{w}_a \in \hat{\rho}(x_a) \mid \hat{\phi} \vdash \hat{w}_a :\approx\!\!\gg \tau_z\} \subseteq \hat{\rho}(z) \wedge \hat{\rho}(\lfloor e_b \rfloor) \subseteq \hat{W}\right)}{\hat{\phi}; \hat{\rho} \vDash x_f\ x_a \rightsquigarrow \hat{W}}
$$

$$
\frac{\bigwedge_{\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b \in \hat{\rho}(x_f)} \left(\{\tau_a\} \subseteq \hat{\phi}(\beta) \wedge \hat{\rho}(\lfloor e_b \rfloor) \subseteq \hat{W}\right)}{\hat{\phi}; \hat{\rho} \vDash x_f\ \texttt{[}\tau_a\texttt{]} \rightsquigarrow \hat{W}}
$$

Fig. 4. Type- and Control-Flow Analysis of ANF System F

recursive type abstraction.[6] Pairs of abstract type and value environments form complete lattices with the usual partial orders for pairs, functions, and power sets.

The judgments $\hat{\phi}; \hat{\rho} \vDash e$ and $\hat{\phi}; \hat{\rho} \vDash b \rightsquigarrow \hat{W}$ assert that a pair of abstract environments $\hat{\phi}$ and $\hat{\rho}$ is an acceptable type- and control-flow analysis of the expression e and bind b, respectively. An acceptable type- and control-flow analysis is one that soundly approximates the run-time behavior of the program, in a sense made precise by Theorem 2; intuitively, acceptable abstract type and value environments must describe every run-time type and value environment that arises during the evaluation of the program. The judgment $\hat{\phi}; \hat{\rho} \vDash b \rightsquigarrow \hat{W}$ additionally asserts that the bind b is approximated by the set of abstract values $\hat{W}$.

Ignoring the shaded terms, the constraints asserted by the rules are standard for a monovariant control-flow analysis. The rule for a `let`-binding expression `let` $x{:}\tau_x$ `=` b `in` e asserts that the abstract environments are acceptable for the bind b and the expression e and that the set of abstract values that approximate the bind b are included in the set of abstract values mapped from the variable x. The rules for values (recursive functions and recursive type abstractions) assert that the value itself is included in both the set of abstract values approximating the bind and the set of abstract values mapped from the μ-bound variable f (corresponding to the $f \mapsto w_f$ binding in the operational semantics making the recursive function or recursive type-application available to the expression body) and that the abstract environments are acceptable for the body expression. The rule for a non-tail function application asserts that, for all functions in the set of abstract values mapped from the variable x_f, the abstract values mapped from the actual argument x_a flow to the formal argument z and the abstract values from the function result $\lfloor e_b \rfloor$ flow to the set of abstract values approximating the function application. Similarly, the rule for a non-tail type application asserts that, for all type abstractions in the set of abstract values mapped from the variable x_f, the actual type argument τ_a flows to the formal type argument β and the abstract values from the function result $\lfloor e_b \rfloor$ flow to the set of abstract values approximating the type application.

Now consider the shaded terms in the rules for a `let`-binding expression and a non-tail function application and the judgments and rules in Figure 5. In essence, the shaded terms perform a kind of analysis-time type checking at the point where there is a non-local flow of abstract values. The judgment $\hat{\phi} \vdash \hat{w} :\approx \pi$ asserts that (the abstract type of) the abstract value $\hat{w}$ is compatible with the abstract type π under $\hat{\phi}$. Thus, in the rule for a `let`-binding expression, each abstract result $\hat{w}_r \in \hat{W}_r$ that flows from the bind to the receiving variable x must have an abstract type that is compatible with τ_x, the static type of the receiving variable. Similarly, in the rule for non-tail function applications, each

[6] Again, we introduce abstract values in preparation for future extensions of the analysis; for example, we may wish to introduce a $\top$ abstract value to represent an unknown value coming from outside the scope of the analysis [49,28] or we may wish to introduce $[m, n]$ abstract values to incorporate an interval/range data-flow analysis [5,15].

$\boxed{\hat{\phi} \vdash \hat{w} :\approx\!\!\gg \hat{\pi}}$

$$\frac{\hat{\phi} \vdash \tau_f \approx\!\!\gg \hat{\pi}}{\hat{\phi} \vdash \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b :\approx\!\!\gg \hat{\pi}} \qquad \frac{\hat{\phi} \vdash \tau_f \approx\!\!\gg \hat{\pi}}{\hat{\phi} \vdash \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b :\approx\!\!\gg \hat{\pi}}$$

$\boxed{\hat{\phi} \vdash \hat{\pi} \approx\!\!\gg \hat{\pi}}$

$$\frac{\bullet; \hat{\phi} \vdash \hat{\pi}_1 \gg\!\!= \tau_1 \qquad \bullet; \hat{\phi} \vdash \hat{\pi}_2 \gg\!\!= \tau_2 \qquad \tau_1 = \tau_2}{\hat{\phi} \vdash \hat{\pi}_1 \approx\!\!\gg \hat{\pi}_2}$$

$\boxed{\Delta; \hat{\phi} \vdash \hat{\pi} \gg\!\!= \tau}$

$$\frac{\Delta; \hat{\phi} \vdash \tau_a \gg\!\!= \tau_a' \qquad \Delta; \hat{\phi} \vdash \tau_b \gg\!\!= \tau_b'}{\Delta; \hat{\phi} \vdash \tau_a \to \tau_b \gg\!\!= \tau_a' \to \tau_b'} \qquad \frac{\vdash \Delta \qquad \Delta(\alpha) = \star}{\Delta; \hat{\phi} \vdash \alpha \gg\!\!= \alpha}$$

$$\frac{\Delta, \alpha{:}\star; \hat{\phi} \vdash \tau_b \gg\!\!= \tau_b'}{\Delta; \hat{\phi} \vdash \forall\alpha.\ \tau_b \gg\!\!= \forall\alpha.\ \tau_b'} \qquad \frac{\vdash \Delta \qquad \alpha \notin \mathsf{dom}(\Delta) \qquad \hat{\pi} \in \hat{\phi}(\alpha) \qquad \bullet; \hat{\phi} \vdash \hat{\pi} \gg\!\!= \tau}{\Delta; \hat{\phi} \vdash \alpha \gg\!\!= \tau}$$

Fig. 5. Analysis-time Type Compatibility

abstract argument $\hat{w}_a \in \hat{\rho}(x_a)$ that flows from the actual argument to the formal argument z must have an abstract type that is compatible with τ_z, the static type of formal argument.

The rules for the judgment $\hat{\phi} \vdash \hat{w} :\approx\!\!\gg \hat{\pi}$ simply form the abstract type of the recursive function or recursive type abstraction from the (static, syntactic) type of the μ-bound variable. The judgment $\hat{\phi} \vdash \hat{\pi}_1 \approx\!\!\gg \hat{\pi}_2$ asserts that the abstract types $\hat{\pi}_1$ and $\hat{\pi}_2$ are compatible under $\hat{\phi}$, by asserting that $\hat{\pi}_1$ and $\hat{\pi}_2$ expand to a common closed type. Finally, the judgment $\Delta; \hat{\phi} \vdash \hat{\pi} \gg\!\!= \tau$ asserts that the abstract type $\hat{\pi}$ expands under $\hat{\phi}$ to the type τ (which is well-formed in Δ). The first rule expands a function type by recursively expanding its argument and result types. The second rule expands a $\forall$-bound type variable to itself, while the third rule expands a universal type by recursively expanding its result type (in a type-variable context extended with α). The fourth rule expands a Λ-bound type variable to an abstract type according to the abstract type environment $\hat{\phi}$ and recursively expands the abstract type; the abstract type is expanded under the empty type-variable context, because it is not in the scope of the $\forall$-bound type variables appearing in Δ. Note that, when used in the context of the type compatibility judgment $\hat{\phi} \vdash \hat{\pi}_1 \approx\!\!\gg \hat{\pi}_2$, this rule must "guess" a satisfying abstract type from among the set of abstract types mapped from the type variable.

3.1 Flow Soundness

We show that, with respect to a given program, every acceptable pair of abstract environments soundly approximates the run-time behavior of the program. To formalize the approximation, we introduce "shallow" abstraction functions that take run-time types and values to abstract types and values and that take run-time type and value environments to abstract type and value environments:[7]

$$\begin{array}{rr} |\cdot| :: \mathit{RType} \to \mathit{AType} & |\cdot| :: \mathit{RValue} \to \mathit{AValue} \\ |\langle \tau; \phi \rangle| = \tau & |\langle \mu f\!:\!\tau_f \,.\, \lambda z\!:\!\tau_z \,.\, e; \phi; \rho \rangle| = \mu f\!:\!\tau_f \,.\, \lambda z\!:\!\tau_z \,.\, e_b \\ & |\langle \mu f\!:\!\tau_f \,.\, \Lambda \beta \,.\, e_b; \phi; \rho \rangle| = \mu f\!:\!\tau_f \,.\, \Lambda \beta \,.\, e_b \end{array}$$

$$\begin{array}{rr} |\cdot| :: \mathit{RTEnv} \to \mathit{ATEnv} & |\cdot| :: \mathit{RVEnv} \to \mathit{AVEnv} \\ |\phi|(\alpha) = \begin{cases} \{\} & \text{if } \alpha \notin \mathsf{dom}(\phi) \\ \{|\pi|\} & \text{if } \phi(\alpha) = \pi \end{cases} & |\rho|(x) = \begin{cases} \{\} & \text{if } x \notin \mathsf{dom}(\rho) \\ \{|w|\} & \text{if } \rho(x) = w \end{cases} \end{array}$$

A proof of flow soundness for well-typed programs, using a Preservation (aka, subject reduction) theorem, is given in Appendix B.

Theorem 2 (Flow Soundness).
If $\bullet; \bullet \vdash e : \tau$, $\hat{\phi}; \hat{\rho} \vDash e$, *and* $\langle e; \bullet; \bullet; \circ \rangle \longrightarrow^* \langle e'; \phi'; \rho'; \kappa' \rangle$, *then* $\langle |\phi'|, |\rho'| \rangle \sqsubseteq \langle \hat{\phi}, \hat{\rho} \rangle$.

In addition to the judgments of Figure 4, we introduce judgments that assert the acceptability of abstract environments with respect to run-time types ($\hat{\phi} \vDash \pi$), run-time type environments ($\hat{\phi} \vDash \phi$), run-time values ($\hat{\phi}; \hat{\rho} \vDash w$), run-time value environments ($\hat{\phi}; \hat{\rho} \vDash \rho$), continuations ($\hat{\phi}; \hat{\rho} \vDash \hat{W} \rightsquigarrow \kappa$), and states ($\hat{\phi}; \hat{\rho} \vDash \varsigma$); see Figure 7 in Appendix B. The judgments $\hat{\phi} \vDash \phi$ and $\hat{\phi}; \hat{\rho} \vDash \rho$ assert that the abstract environments are "deep" abstractions of the run-time environments.

A key lemma is the following, which establishes that two abstract types may be judged compatible if their expansions (induced by run-time type environments for which the abstract type environment is acceptable) are syntactically equal:

Lemma 3 (Syntactic Equality implies Analysis-Time Type Compatibility)
If $\vdash \phi_1 : \theta_1$, $\hat{\phi} \vDash \phi_1$, $\bullet \vdash \theta_1(\tau_1)$, $\vdash \phi_2 : \theta_2$, $\hat{\phi} \vDash \phi_2$, $\bullet \vdash \theta_2(\tau_2)$,
and $\theta_1(\tau_1) = \theta_2(\tau_2)$, *then* $\hat{\phi} \vdash \tau_1 \approx\!\!\approx \tau_2$.

In the Preservation proof, the necessary preconditions for this lemma are obtained from the well-typedness of the machine state undergoing transition.

3.2 Existence of Minimum, Finite Flows

Although presented in constraint form, our type- and control-flow analysis can be presented in an equivalent fixpoint form [6]. It is straightforward to read the

[7] These abstraction functions are "shallow" in the sense that they do not abstract and join the embedded run-time type and value environments of run-time types and values.

analysis of Figure 4 as defining a monotone function from pairs of abstract environments to pairs of abstract environments; the "input" abstract environments are used for terms of the form $\hat{w} \in \hat{\rho}(x)$ and $\hat{\phi} \vdash \hat{w} :\approx \tau$, while the "output" abstract type environment is formed from the "input" abstract type environment joined with $\hat{\phi}_\bot[\beta \mapsto \hat{\Pi}]$ for terms of the form $\hat{\Pi} \subseteq \hat{\phi}(\beta)$ and the "output" abstract value environment is formed from the "input" abstract value environment joined with $\hat{\phi}_\bot[x \mapsto \hat{W}]$ for terms of the form $\hat{W} \subseteq \hat{\rho}(x)$ For a given program, fixed points of this monotone function are acceptable pairs of abstract environments. Since pairs of abstract environments form complete lattices, Tarski's fixed point theorem establishes that:

Theorem 4 (Minimum Flows Exist).
For all expressions e, there exist minimum abstract environments $\hat{\phi}_{\min}$ and $\hat{\rho}_{\min}$ such that $\hat{\phi}_{\min}; \hat{\rho}_{\min} \vDash e$.

Furthermore, for a given program e, the minimum abstract type environment must be an element of $\mathit{ATEnv}^e = \mathit{TyVar}^e \to \mathcal{P}(\mathit{AType}^e)$ (where TyVar^e is the set of Λ-bound type variables that occur in the program and AType^e is the set of (syntactic) types that occur in the program) and the minimum abstract value environment must be an element of $\mathit{AEnv}^e = \mathit{Var}^e \to \mathcal{P}(\mathit{AValue}^e)$ (where Var^e is the set of `let`-, μ-, and λ-bound variables that occur in the program and AValue^e is the set of (syntactic) values (recursive functions and recursive type abstractions) that occur in the program). These abstract environments are "finite", in the sense that they map finite domains to finite sets, and form complete lattices.

3.3 Decidability and Computability of Flows

While Theorems 2 and 4 establish that, for every program, there is a "best" pair of abstract environments that soundly approximates the run-time behavior of the program, we would like this pair of abstract environments to be computable. The key concern is the decidability of the $\hat{\phi} \vdash \hat{\pi}_1 \approx \hat{\pi}_2$ judgment. Even simply verifying that a pair of abstract environments is acceptable for a given program requires showing that constraints of the form $\{\hat{w}_a \in \hat{\rho}(x_a) \mid \hat{\phi} \vdash \hat{w}_a :\approx \tau_z\} \subseteq \hat{\rho}(z)$ are satisfied; this, in turn, requires showing, for each abstract value $\hat{w}_a$ that is an element of $\hat{\rho}(x_a)$ but is not an element of $\hat{\rho}(z)$, that the judgment $\hat{\phi} \vdash \hat{w}_a :\approx \tau_z$ is not derivable.[8]

Due to "recursion" in the abstract type environment, whereby a type variable may be mapped to a set of abstract types in which the type variable itself occurs free, we cannot simply enumerate the (potentially infinite sets of) closed types τ_1 and τ_2 such that $\bullet; \hat{\phi} \vdash \hat{\pi}_1 \twoheadrightarrow \tau_1$ and $\bullet; \hat{\phi} \vdash \hat{\pi}_2 \twoheadrightarrow \tau_2$ in order to decide whether or not the judgment $\hat{\phi} \vdash \hat{\pi}_1 \approx \hat{\pi}_2$ is derivable (via types τ_1 and τ_2 such that

[8] Note, however, that this does not require showing, for each abstract value $\hat{w}_a$ that is an element of both $\hat{\rho}(x_a)$ and $\hat{\rho}(z)$, that the judgment $\hat{\phi} \vdash \hat{w}_a :\approx \tau_z$ is derivable; the constraint is satisfied whether or not the judgment is derivable.

$\bullet; \hat{\phi} \vdash \hat{\pi}_1 \twoheadrightarrow \tau_1$, $\bullet; \hat{\phi} \vdash \hat{\pi}_2 \twoheadrightarrow \tau_2$, and $\tau_1 = \tau_2$) To address this issue, we take inspiration from the theory and implementation of regular-tree grammars [12,2,7], which has been used extensively for flow analysis [23,22,17,16] (including type inference [29,3]), but whereas previous work has applied regular-tree grammars to the analysis of values, we apply regular-tree grammars to the analysis of types.

Given a finite abstract type environment $\hat{\phi}$, we interpret it as a regular-tree grammar as follows: the set of non-terminals is $\mathsf{dom}(\hat{\phi})$ and the set of productions is $\{\alpha \Rightarrow \hat{\pi} \mid \alpha \in \mathsf{dom}(\hat{\phi}) \wedge \hat{\pi} \in \hat{\phi}(\alpha)\}$. The language generated by the grammar $\hat{\phi}$ for the starting term $\hat{\pi}$ is $\mathcal{L}_{\hat{\phi}}(\hat{\pi}) \stackrel{\text{def}}{=} \{\tau' \in \mathit{Type} \mid \bullet; \hat{\phi} \vdash \hat{\pi} \twoheadrightarrow \tau'\}$; a derivation of $\bullet; \hat{\phi} \vdash \hat{\pi} \twoheadrightarrow \tau'$ is exactly a parse tree witnessing the derivation of τ' from $\hat{\pi}$ by $\hat{\phi}$.

Consider deciding whether or not $\hat{\phi}_{\mathsf{ex}} \vdash \mathsf{int} \to \beta \approx\!\!> \alpha$ is derivable, where $\hat{\phi}_{\mathsf{ex}}(\alpha) = \{\mathsf{int} \to \mathsf{int}, \mathsf{int} \to \alpha\}$ and $\hat{\phi}_{\mathsf{ex}}(\beta) = \{\mathsf{int} \to \mathsf{bool}, \mathsf{int} \to \beta\}$. Intuitively, it is not derivable because

$$\mathcal{L}_{\hat{\phi}_{\mathsf{ex}}}(\mathsf{int} \to \beta) = \{\mathsf{int} \to \mathsf{int} \to \mathsf{bool}, \mathsf{int} \to \mathsf{int} \to \mathsf{int} \to \mathsf{bool}, \ldots\}$$
$$\mathcal{L}_{\hat{\phi}_{\mathsf{ex}}}(\alpha) = \{\mathsf{int} \to \mathsf{int}, \mathsf{int} \to \mathsf{int} \to \mathsf{int}, \mathsf{int} \to \mathsf{int} \to \mathsf{int} \to \mathsf{int}, \ldots\}$$

and $\mathcal{L}_{\hat{\phi}_{\mathsf{ex}}}(\mathsf{int} \to \beta) \cap \mathcal{L}_{\hat{\phi}_{\mathsf{ex}}}(\alpha) = \emptyset$; there is no (closed) type that is generated by $\hat{\phi}_{\mathsf{ex}}$ from both $\mathsf{int} \to \beta$ and α. Simply unfolding definitions establishes that:

Theorem 5 (Analysis-Time Type Compatibility iff Languages Intersect). *$\hat{\phi} \vdash \hat{\pi}_1 \approx\!\!> \hat{\pi}_2$ if and only if $\mathcal{L}_{\hat{\phi}}(\hat{\pi}_1) \cap \mathcal{L}_{\hat{\phi}}(\hat{\pi}_2) \neq \emptyset$.*

An immediate corollary of Theorem 5 is the decidability of type compatibility (under a finite abstract type environment), since regular-tree grammars are closed under intersection and the emptiness of a regular-tree grammar is decidable [12,29]. In turn, we have that the acceptability of a pair of finite abstract environments for a given program is decidable. Finally, we have that the minimum acceptable pair of abstract environments for a given program is computable, either by enumerating the finite abstract environments for the program and checking acceptability or by defining the analysis as a monotone function from pairs of abstract environments to pairs of abstract environmentsnd using a standard least fixed-point computation.

We briefly sketch implementations of testing emptiness and intersection of regular-tree grammars, based on those given by Aiken and Murphy [2]; both operations are (worst-case) quadratic time in the size of the regular-tree grammar. Recall that, for a given program e, it suffices to consider finite abstract type environments $\hat{\phi}^e \in \mathit{ATEnv}^e$, interpreted as (finite) regular-tree grammars.

To decide the emptiness of a language, we define the function Ψ as follows:

$$\Psi :: \mathit{AEnv} \to (\mathit{TyVar} \to \mathbb{B})$$
$$\Psi(\hat{\phi}) = \mathrm{lfp}\, F$$
$$\text{where} \qquad F :: (\mathit{TyVar} \to \mathbb{B}) \to (\mathit{TyVar} \to \mathbb{B})$$
$$F(\psi)(\alpha) = \begin{cases} \top & \text{if } \exists \hat{\pi} \in \hat{\phi}(\alpha).\ \forall \beta \in \mathsf{FTV}(\hat{\pi}).\ \psi(\beta) = \top \\ \bot & \text{if } \forall \hat{\pi} \in \hat{\phi}(\alpha).\ \exists \beta \in \mathsf{FTV}(\hat{\pi}).\ \psi(\beta) = \bot \end{cases}$$

where $\mathbb{B} = \{\top, \bot\}$ with the usual partial order ($\bot \sqsubseteq \top$); $\bot$ (resp. $\top$) denotes an empty (resp. non-empty) language. The language $\mathcal{L}_{\hat{\phi}}(\hat{\pi})$ is non-empty if and only if $\forall \beta \in \mathsf{FTV}(\hat{\pi}).\ \Psi(\hat{\phi})(\beta) = \top$. If $\hat{\phi}$ is a finite abstract type environment, then $\Psi(\hat{\phi})$ is computable using a standard least fixed-point computation.

In order to intersect the languages generated by the regular-tree grammar $\hat{\phi}$ for the starting terms $\hat{\pi}_1$ and $\hat{\pi}_2$, we extend $\hat{\phi}$ with finitely many additional non-terminals and productions to obtain $\hat{\phi}^\star$ and generate a starting term $\hat{\pi}^\star$ such that $\mathcal{L}_{\hat{\phi}}(\hat{\pi}_1) \cap \mathcal{L}_{\hat{\phi}}(\hat{\pi}_2) = \mathcal{L}_{\hat{\phi}^\star}(\hat{\pi}^\star)$. The idea is that each new non-terminal represents the intersection of a type variable in $\mathsf{dom}(\hat{\phi})$ and a type; a global mapping from pairs of type variables and types to new non-terminals is maintained to ensure that the same new non-terminal is used whenever the same pair is encountered.

To illustrate the technique, consider intersecting the languages generated by $\hat{\phi}_{\mathsf{ex}}$ for the starting terms $\mathsf{int} \to \beta$ and α. First, extend the grammar with a new non-terminal Z and no productions (i.e., extend $\hat{\phi}_{\mathsf{ex}}$ with the mapping $Z \mapsto \{\}$); the non-terminal Z will serve as the starting term for an empty language. We are trying to intersect $\mathsf{int} \to \beta$ and α; since α is a non-terminal, generate a new non-terminal A_0 mapped from the pair $\langle \mathsf{int} \to \beta; \alpha \rangle$, add the triple $\langle A_0; \{\mathsf{int} \to \beta\}; \hat{\phi}_{\mathsf{ex}}(\alpha) \rangle$ to a work list, and return A_0 as the result of intersecting $\mathsf{int} \to \beta$ and α. The work list contains new non-terminals whose productions should be generated by intersecting all pairs of elements from the two sets. Therefore, add productions corresponding to $A_0 \Rightarrow \mathsf{int} \to \beta \mathbin{⍉} \mathsf{int} \to \mathsf{int}$ and $A_0 \Rightarrow \mathsf{int} \to \beta \mathbin{⍉} \mathsf{int} \to \alpha$. Intersecting $\mathsf{int} \to \beta$ and $\mathsf{int} \to \mathsf{int}$ generates a new non-terminal A_1 mapped from $\langle \beta; \mathsf{int} \rangle$, adds $\langle A_1; \hat{\phi}_{\mathsf{ex}}(\beta); \{\mathsf{int}\} \rangle$ to the work list, and returns $\mathsf{int} \to A_1$. Intersecting $\mathsf{int} \to \beta$ and $\mathsf{int} \to \alpha$ generates a new non-terminal A_2 mapped from $\langle \beta; \alpha \rangle$, adds $\langle A_1; \hat{\phi}_{\mathsf{ex}}(\beta); \hat{\phi}_{\mathsf{ex}}(\alpha) \rangle$ to the work list, and returns $\mathsf{int} \to A_2$. Therefore, extend with the mapping $A_0 \mapsto \{\mathsf{int} \to A_1\} \cup \{\mathsf{int} \to A_2\}$. Returning to the work list, add productions corresponding to $A_1 \Rightarrow \mathsf{int} \to \mathsf{bool} \mathbin{⍉} \mathsf{int}$ and $A_1 \Rightarrow \mathsf{int} \to \beta \mathbin{⍉} \mathsf{int}$. Intersecting $\mathsf{int} \to \mathsf{bool}$ and int returns Z (since clearly the intersection of the languages generated from these two starting terms is empty), as does intersecting $\mathsf{int} \to \beta$ and int; therefore, extend with the mapping $A_1 \mapsto \{Z\} \cup \{Z\}$. Returning to the work list, add productions corresponding to $A_2 \Rightarrow \mathsf{int} \to \mathsf{bool} \mathbin{⍉} \mathsf{int} \to \mathsf{int}$ (returning Z), $A_2 \Rightarrow \mathsf{int} \to \mathsf{bool} \mathbin{⍉} \mathsf{int} \to \alpha$ (generating a new non-terminal A_3 mapped from $\langle \mathsf{bool}; \alpha \rangle$, adding $\langle A_3; \{\mathsf{bool}\}; \hat{\phi}_{\mathsf{ex}}(\alpha) \rangle$ to the work list, and returning $\mathsf{int} \to A_3$), $A_2 \Rightarrow \mathsf{int} \to \beta \mathbin{⍉} \mathsf{int} \to \mathsf{int}$ (returning $\mathsf{int} \to A_1$, using the global map), and $A_2 \Rightarrow \mathsf{int} \to \beta \mathbin{⍉} \mathsf{int} \to \alpha$ (returning $\mathsf{int} \to A_2$, using the global map); therefore, extend with the mapping $A_2 \mapsto \{Z\} \cup \{\mathsf{int} \to A_3\} \cup \{\mathsf{int} \to A_1\} \cup \{\mathsf{int} \to A_2\}$. Finally, add productions corresponding to $A_3 \Rightarrow \mathsf{bool} \mathbin{⍉} \mathsf{int} \to \mathsf{int}$ (returning Z) and $A_3 \Rightarrow \mathsf{bool} \mathbin{⍉} \mathsf{int} \to \alpha$ (returning Z); therefore, extend with the mapping

$A_3 \mapsto \{Z\} \cup \{Z\}$. In summary, we have

Global map	New productions
$\langle \mathsf{int} \to \beta; \alpha \rangle \mapsto A_0$	$A_0 \mapsto \{\mathsf{int} \to A_1\} \cup \{\mathsf{int} \to A_2\}$
$\langle \beta; \mathsf{int} \rangle \mapsto A_1$	$A_1 \mapsto \{Z\} \cup \{Z\}$
$\langle \beta; \alpha \rangle \mapsto A_2$	$A_2 \mapsto \{Z\} \cup \{\mathsf{int} \to A_3\} \cup \{\mathsf{int} \to A_1\} \cup \{\mathsf{int} \to A_2\}$
$\langle \mathsf{bool}; \alpha \rangle \mapsto A_3$	$A_3 \mapsto \{Z\} \cup \{Z\}$
	$Z \mapsto \{\}$

To conclude, return $\hat{\phi}^{\star}_{\mathsf{ex}}$ equal to $\hat{\phi}_{\mathsf{ex}}$ extended with the new productions and $\hat{\pi}^{\star}$ equal to A_0. Finally, note that $\Psi(\hat{\phi}^{\star}_{\mathsf{ex}})(\hat{\pi}^{\star}) = \bot$, confirming that $\mathcal{L}_{\hat{\phi}_{\mathsf{ex}}}(\mathsf{int} \to \beta) \cap \mathcal{L}_{\hat{\phi}_{\mathsf{ex}}}(\alpha) = \emptyset$ and that $\hat{\phi}_{\mathsf{ex}} \vdash \mathsf{int} \to \beta \approx\!\!\!\approx \alpha$ is not derivable.

We conclude with a crude upper-bound on the time complexity of our type- and control-flow analysis. Consider a program of size n and the analysis defined in fixedpoint form. The two abstract environments are lattices of height $O(n^2)$. Each (naïve) iteration of the monotone function is syntax directed ($O(n)$) and dominated by the function-application bind, which loops over all of the elements of $\hat{\rho}(x_f)$ ($O(n)$), loops over all of the elements of $\hat{\rho}(x_a)$ ($O(n)$), and computes type compatibility via a regular-tree grammar intersection ($O(n^2)$) and emptiness test ($O((n^2)^2)$, because the regular-tree grammar representing the intersection may be of size $O(n^2)$). Hence, our analysis is computable in polynomial time: $O((n^2 + n^2) * (n * n * n * (n^2 + n^4))) = O(n^9)$. Further considerations regarding implementations of our type and control-flow analysis are given in Section 5.

4 Related Work

There is surprisingly little work on control-flow analyses for statically-typed languages with polymorphic types. Control-flow analyses have typically been formulated for dynamically- or simply-typed languages.[9] Production implementations of control-flow analyses for Standard ML, a language with rank-1 polymorphism (i.e., "let"-polymorphism), typically handle the polymorphism either by monomorphisation [4] (explicitly eliminating polymorphism before analysis) or by polyvariance [16] (implicitly eliminating polymorphism during analysis).

The most closely related work is the "Type-Directed Flow Analysis for Typed Intermediate Languages" of Jagannathan, Weeks, and Wright [20], which describes a framework for polyvariant flow analyses of Λ_i, the predicative subset of System F extended with recursive functions. A specific analysis called $S_{\mathcal{RT}}$ uses types to control polyvariance; essentially, $S_{\mathcal{RT}}$ introduces a distinct polyvariance context for each closed type at which a polymorphic function is applied, yielding an analysis more precise than our type- and control-flow analysis. Furthermore, $S_{\mathcal{RT}}$ respects types, meaning that if $\hat{v} \in F(x)$ (the abstract value $\hat{v}$ is assigned to x by the analysis) and $x : \sigma$ (the type scheme σ is assigned to x by the type system), then $[\![\hat{v}]\!] \subseteq [\![\sigma]\!]$, where $[\![\cdot]\!]$ denotes a set of values. Unfortunately, $S_{\mathcal{RT}}$

[9] Again, we draw a distinction between flow analyses expressed as sophisticated type systems and flow analyses of languages with sophisticated type systems.

does not terminate on programs that use polymorphic recursion [32,18,24]; such programs may instantiate a polymorphic function at an infinite number of closed types during execution. In contrast, our type- and control-flow analysis is computable for all programs in (impredicative) System F extended with recursive functions.

Another closely related work is the "Type-sensitive Control-Flow Analysis" of Reppy [40], which describes an extension of Serrano's version of 0CFA [46] that uses a program's type information to compute more precise results. Serrano's and Reppy's analyses are modular and use an abstract value $\top$ to denote an unknown value; variables bound outside the unit of analysis are assigned $\top$, as are the parameters of functions that escape the unit of analysis. Reppy's insight is that values of an abstract type can only be created within their defining module; hence, "unknown" values of the abstract type can be soundly approximated by the known set of escaping values of the abstract type (a subset of the set of values of the abstract type created within the defining module). This leads to a type-indexed family of abstract values for unknown values, in addition to the $\top$ abstract value. Reppy's analysis is formulated for a simply-typed language with top-level abstract types; he suggests extending the analysis to a language with polymorphism by mapping type variables to the $\top$ abstract value. Our type- and control-flow analysis is a whole-program analysis, but has a more precise treatment of type variables.

5 Future Work

There are many directions for future work.

While we have established the computability of the minimum, finite acceptable pair of abstract environments for every program, we would like our type- and control-flow analysis to be efficiently computable. A popular approach for computing control-flow analyses is as a constraint-based analysis [1]; an initial phase generates constraints that a solution to the analysis must satisfy, while a subsequent phase solves the constraints.[10] The syntax-directed 0CFA that we adapt for our type- and control-flow analysis has an $O(n^3)$ algorithm following this approach [34, Section 3.4]. However, algorithms for solving a set of constraints are sensitive to the syntax of constraints; the filtering of sets by type compatibility may prove problematic, since the derivability of a type-compatibility judgment depends upon the abstract type environment, itself being solved for.

Independent of the overall approach, it seems clear that we will need to efficiently decide the derivability of a type-compatibility judgment under an abstract type environments. We have established that this decision can be made by intersecting and testing the emptiness of regular-tree grammars. Aiken and Murphy [2, Section 4] suggest maintaining a regular-tree grammar with an invariant that makes testing the emptiness (of a non-terminal) constant time. Aiken and Murphy [2, Section 5.3] also suggest that the algorithm given previously, which

[10] More sophisticated approaches exist where additional constraints are generated during the solving phase.

generates only the intersections necessary to express the result, performs well in the typical case. We further observe that, for a fixed abstract type environment, we can maintain the global map from pairs of type variables and types to new non-terminals (where each new non-terminal represents the intersection of the expansions of the type variable under the abstract type environment and the type) across decisions of the derivability of type-compatibility. Hence, the (worst-case) quartic time can be amortized over all queries under a given abstract type environment, not each query, and improves our crude upper-bound to $O(n^6)$. We may also be able to exploit the fact that we are only interested in the emptiness of an intersection of regular-tree grammars, and not the intersection itself.

Another direction of future work is to extend the type- and control-flow analysis to handle unknown and escaping values [49,28] and types [40]. It should be straightforward to introduce a $\top$ abstract type and a $\top$ abstract value; conservatively, the $\top$ abstract type should be judged compatible with any other abstract type. A more interesting direction is to consider primitives that make essential use of higher-rank polymorphism, such as Haskell's `runST` [25,26].

Yet another direction is to extend the monovariant type- and control-flow analysis to a polyvariant analysis.

Finally, we would like to extend type- and control-flow analysis to languages with even more sophisticated type systems. Of particular interest is System F with guarded algebraic data types (GADTs), as we would like to combine the flow-directed defunctionalization of Cejtin, Jagannathan, and Weeks [4] with the polymorphic typed defunctionalization of Pottier and Gauthier [37,38]. Also of interest is System F_ω, the higher-order polymorphic lambda-calculus: System F_ω has been used as a target language for the elaboration of a full-featured, higher-order ML-like module language [45] and System F_ω extended with type equality coercions [50] is used as a typed intermediate language in the Glasgow Haskell Compiler (GHC).

Acknowledgments

Many thanks to Jan Midtgaard for the excellent survey "Control-flow analysis of functional programs" [27] and companion bibliography and for valuable feedback on an earlier draft. Thanks to Jurriaan Hage, Fritz Henglein, and Peter Thiemann for thoughtful conversation at IFL'12. This material is based upon work supported by the National Science Foundation under Grant No. 1065099. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

References

1. Aiken, A.: Introduction to set constraint-based program analysis. Science of Computer Programming 35(2-3), 79–111 (1999)

2. Aiken, A., Murphy, B.R.: Implementing regular tree expressions. In: Hughes, J. (ed.) FPCA'91: Proceedings of the Fifth ACM Conference on Functional Programming Languages and Computer Architecture. Lecture Notes in Computer Science, vol. 523, pp. 427–447. Springer-Verlag, Cambridge, Massachusetts (Aug 1991)
3. Aiken, A., Murphy, B.R.: Static type inference in a dynamically typed language. In: Cartwright, R.C. (ed.) POPL'91: Proceedings of the Eighteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 279–290. Orlando, Florida (Jan 1991)
4. Cejtin, H., Jagannathan, S., Weeks, S.: Flow-directed closure conversion for typed languages. In: Smolka, G. (ed.) ESOP'00: Proceedings of the Ninth European Symposium on Programming. Lecture Notes in Computer Science, vol. 1782, pp. 56–71. Springer-Verlag, Berlin, Germany (Mar 2000)
5. Cousot, P., Cousot, R.: Static determination of dynamic properties of programs. In: Robinet, B. (ed.) Proceedings of the First International Symposium on Programming. pp. 106–130. Paris, France (Apr 1976)
6. Cousot, P., Cousot, R.: Compositional and inductive semantic definitions in fixpoint, equational, constraint, closure-condition, rule-based and game-theoretic form, invited paper. In: Wolper, P. (ed.) CAV'95: Proceedings of the Seventh International Conference on Computer Aided Verification. Lecture Notes in Computer Science, vol. 939, pp. 293–308. Springer-Verlag, Liège, Belgium (Jul 1995)
7. Cousot, P., Cousot, R.: Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In: Peyton Jones, S. (ed.) FPCA'95: Proceedings of the Seventh International Conference on Functional Programming Languages and Computer Architecture. pp. 170–181. La Jolla, California (Jun 1995)
8. Danvy, O.: Three steps for the CPS transformation. Tech. Rep. CIS-92-2, Kansas State University, Manhattan, Kansas (1991)
9. Faxén, K.F.: Polyvariance, polymorphism and flow analysis. In: Dam, M. (ed.) Selected Papers from the 5th LOMAPS Workshop on Analysis and Verification of Multiple-Agent Languages. Lecture Notes in Computer Science, vol. 1192, pp. 260–278. Springer-Verlag, Stockholm, Sweden (1997)
10. Flanagan, C., Sabry, A., Duba, B.F., Felleisen, M.: The essence of compiling with continuations. In: Cartwright, R. (ed.) PLDI'93: Proceedings of the ACM SIGPLAN 1993 Conference on Programming Languages Design and Implementation. pp. 237–247. ACM, Albuquerque, New Mexico (Jun 1993)
11. Fluet, M.: A type- and control-flow analaysis for System F. In: Hinze, R. (ed.) IFL'12: Post-Proceedings of the 24th International Symposium on Implementation and Application of Functional Languages. Lecture Notes in Computer Science, Springer-Verlag, Oxford, England (2013), to appear
12. Gecseg, F., Steinby, M.: Tree Automata. Akademiai Kiado, Budapest, Hungary (1984)
13. Girard, J.Y.: Une extension de l'interpretation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In: Fenstad, J.E. (ed.) Proceedings of the 2nd Scandinavian Logic Symposium. Studies in Logic and the Foundations of Mathematics, vol. 63, pp. 63–92. Elsevier, Amsterdam, Netherlands (1971)
14. Harper, R., Morrisett, G.: Compiling polymorphism using intensional type analysis. In: Lee, P. (ed.) POPL'95: Proceedings of the 22nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 130–141. San Francisco, California (Jan 1995)

15. Harrison III, W.L.: Compiler analysis of the value ranges for variables. IEEE Transactions on Software Engineering SE-3(3), 243–250 (May 1977)
16. Heintze, N.: Set-based program analysis of ML programs. In: Talcott, C.L. (ed.) LFP'94: Proceedings of the 1994 ACM Conference on Lisp and Functional Programming. pp. 306–317. Orlando, Florida (Jun 1994)
17. Heintze, N., Jaffar, J.: A finite presentation theorem for approximating logic programs. In: Hudak, P. (ed.) POPL'90: Proceedings of the Seventeenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 197–209. San Francisco, California (Jan 1990)
18. Henglein, F.: Type inference with polymorphic recursion. ACM Transactions on Programming Languages and Systems 15(2), 253–289 (1993)
19. Holdermans, S., Hage, J.: Polyvariant flow analysis with higher-ranked polymorphic types and higher-order effect operators. In: Weirich, S. (ed.) ICFP'10: Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming. pp. 63–74. ACM, Baltimore, Maryland (Sep 2010)
20. Jagannathan, S., Weeks, S., Wright, A.K.: Type-directed flow analysis for typed intermediate languages. In: Hentenryck, P.V. (ed.) SAS'97: Proceedings of the 4th International Symposium on Static Analysis. Lecture Notes in Computer Science, vol. 1302, pp. 232–249. Springer-Verlag, Paris, France (Sep 1997)
21. Jones, N.D.: Flow analysis of lambda expressions (preliminary version). In: Even, S., Kariv, O. (eds.) ICALP'81: Proceedings of the 8th International Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science, vol. 115, pp. 114–128. Springer-Verlag, Acre (Akko), Israel (Jul 1981)
22. Jones, N.D.: Flow analysis of lazy higher-order functional programs. In: Abramsky, S., Hankin, C. (eds.) Abstract Interpretation of Declarative Languages, chap. 4, pp. 103–122. Ellis Horwood (1987)
23. Jones, N.D., Muchnick, S.S.: Flow analysis and optimization of LISP-like structures. In: Rosen, B.K. (ed.) POPL'79: Proceedings of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 244–256. San Antonio, Texas (Jan 1979)
24. Kfoury, A., Tiuryn, J., Urzyczyn, P.: Type reconstruction in the presence of polymorphic recursion. ACM Transactions on Programming Languages and Systems 15(2), 290–311 (1993)
25. Launchbury, J., Peyton Jones, S.: Lazy functional state threads. In: Sarkar, V. (ed.) PLDI'94: Proceedings of the ACM SIGPLAN 1994 Conference on Programming Languages Design and Implementation. pp. 24–35. Orlando, Florida (Jun 1994)
26. Launchbury, J., Peyton Jones, S.: State in Haskell. Lisp and Symbolic Computation 8(4), 293–341 (1995)
27. Midtgaard, J.: Control-flow analysis of functional programs. ACM Computing Surveys 44(3), 10:1–10:33 (Jun 2012)
28. Midtgaard, J., Adams, M., Might, M.: A structural soundness proof for Shivers's escape technique. In: Miné, A., Schmidt, D. (eds.) SAS'12: Proceedings of the 19th International Symposium on Static Analysis. Lecture Notes in Computer Science, vol. 7460, pp. 352–369. Springer-Verlag, Deauville, France (Sep 2012)
29. Mishra, P., Reddy, U.S.: Declaration-free type checking. In: Van Deusen, M.S., Galil, Z., Reid, B.K. (eds.) POPL'85: Proceedings of the Twelfth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 7–21. ACM, ACM, New Orleans, Louisiana (Jan 1985)
30. Morrisett, G., Harper, R.: Semantics of memory management for polymorphic languages. In: Gordon, A.D., Pitts, A.M. (eds.) Higher-Order Operational Techniques

in Semantics, pp. 175–226. Publications of the Newton Institute, Cambridge University Press (1998)
31. Mossin, C.: Exact flow analysis. Mathematical Structures in Computer Science 13(1), 125–156 (2003)
32. Mycroft, A.: Polymorphic type schemes and recursive definitions. In: Paul, M., Robinet, B. (eds.) Proceedings of the Sixth International Symposium on Programming. Lecture Notes in Computer Science, vol. 167, pp. 217–228. Springer-Verlag, Toulouse, France (Apr 1984)
33. Nielson, F., Nielson, H.R.: Interprocedural control flow analysis. In: Swierstra, S.D. (ed.) ESOP'99: Proceedings of the Eighth European Symposium on Programming. Lecture Notes in Computer Science, vol. 1576, pp. 20–39. Springer-Verlag, Amsterdam, The Netherlands (Mar 1999)
34. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer-Verlag (1999)
35. Palsberg, J.: Type-based analysis and applications. In: Field, J., Snelting, G. (eds.) PASTE'01: Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. pp. 20–27. Snowbird, Utah, United States (2001)
36. Peyton Jones, S.: Compiling Haskell by program transformation: A report from the trenches. In: Nielson, H.R. (ed.) ESOP'96: Proceedings of the Sixth European Symposium on Programming. Lecture Notes in Computer Science, vol. 1058, pp. 18–44. Springer-Verlag, Linköping, Sweden (Apr 1996)
37. Pottier, F., Gauthier, N.: Polymorphic typed defunctionalization. In: Leroy, X. (ed.) POPL'04: Proceedings of the 31st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 89–98. Venice, Italy (Jan 2004)
38. Pottier, F., Gauthier, N.: Polymorphic typed defunctionalization and concretization. Higher-Order and Symbolic Computation 19(1), 125–162 (2006)
39. Rehof, J., Fähndrich, M.: Type-based flow analysis: from polymorphic subtyping to CFL-reachability. In: Nielson, H.R. (ed.) POPL'01: Proceedings of the 28th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 54–66. London, United Kingdom (Jan 2001)
40. Reppy, J.: Type-sensitive control-flow analysis. In: Kennedy, A., Pottier, F. (eds.) ML'06: Proceedings of the 2006 ACM SIGPLAN Workshop on ML. pp. 74–83. Portland, Oregon (Sep 2006)
41. Reynolds, J.: Towards a theory of type structure. In: Robinet, B. (ed.) Proceedings of the First International Symposium on Programming. Lecture Notes in Computer Science, vol. 19, pp. 408–425. Springer-Verlag, Paris, France (Apr 1974)
42. Reynolds, J.C.: Definitional interpreters for higher-order programming languages. In: Shields, R. (ed.) ACM'72: Proceedings of 25th ACM National Conference. pp. 717–740. Boston, Massachusetts (1972), reprinted as [43], with a foreword [44]
43. Reynolds, J.C.: Definitional interpreters for higher-order programming languages. Higher-Order and Symbolic Computation 11(4), 363–397 (1998), reprinting of [42]
44. Reynolds, J.C.: Definitional interpreters revisited. Higher-Order and Symbolic Computation 11(4), 355–361 (1998)
45. Rossberg, A., Russo, C., Dreyer, D.: F-ing modules. In: Benton, N. (ed.) TLDI'10: Proceedings of the Fifth ACM SIGPLAN Workshop on Types in Language Design and Implementation. pp. 89–102. Madrid, Spain (Jan 2010)
46. Serrano, M.: Control flow analysis: a functional languages compilation paradigm. In: George, K.M., Carroll, J., Oppenheim, D. (eds.) SAC'95: Proceedings of the

1995 ACM Symposium on Applied Computing. pp. 118–122. Nashville, Tennessee (Feb 1995)
47. Sestoft, P.: Replacing function parameters by global variables. In: Stoy, J.E. (ed.) FPCA'89: Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture. pp. 39–53. London, England (Sep 1989)
48. Shivers, O.: Control-flow analysis in Scheme. In: Schwartz, M.D. (ed.) PLDI'88: Proceedings of the ACM SIGPLAN 1988 Conference on Programming Languages Design and Implementation. pp. 164–174. Atlanta, Georgia (Jun 1988)
49. Shivers, O.: Control-Flow Analysis of Higher-Order Languages or Taming Lambda. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania (May 1991), Technical Report CMU-CS-91-145
50. Sulzmann, M., Chakravarty, M.M.T., Peyton Jones, S., Donnelly, K.: System F with type equality coercions. In: Necula, G. (ed.) TLDI'07: Proceedings of the 2003 ACM SIGPLAN International Workshop on Types in Language Design and Implementation. pp. 53–66. Nice, France (Jan 2007)
51. Tarditi, D.: Design and implementation of code optimizations for a type-directed compiler for Standard ML. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, Pensylvania (1997)
52. Tarditi, D., Morrisett, G., Cheng, P., Stone, C., Harper, R., Lee, P.: TIL: a type-directed optimizing compiler for ML. In: Fischer, C. (ed.) PLDI'96: Proceedings of the ACM SIGPLAN 1996 Conference on Programming Languages Design and Implementation. pp. 181–192. ACM, Philadelphia, Pennsylvania (May 1996)
53. Tolmach, A., Oliva, D.P.: From ML to Ada: Strongly-typed language interoperability via source translation. Journal of Functional Programming 8(4), 367–412 (1998)
54. Wells, J.B., Dimock, A., Muller, R., Turbak, F.: A calculus with polymorphic and polyvariant flow types. Journal of Functional Programming 12(3), 183–227 (2002)
55. Wright, A.K., Felleisen, M.: A syntactic approach to type soundness. Information and Computation 115(1), 38–94 (1994)

A Type Soundness

Type substitutions $\quad TSubst \ni \theta ::= \bullet \mid \theta, \alpha \rightarrowtail \tau$

$\boxed{\vdash \pi \Rrightarrow \tau}$

$$\frac{\vdash \phi : \theta \qquad \bullet \vdash \theta(\tau)}{\vdash \langle \tau ; \phi \rangle \Rrightarrow \theta(\tau)}$$

$\boxed{\vdash \phi : \theta}$

$$\frac{}{\vdash \bullet : \bullet} \qquad\qquad \frac{\vdash \phi : \theta \qquad \vdash \pi \Rrightarrow \tau}{\vdash \phi, \alpha \mapsto \pi : \theta, \alpha \rightarrowtail \tau}$$

$\boxed{\vdash w : \tau}$

$$\frac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet ; \Gamma \vdash \theta(\mu f{:}\tau_f \,.\, \lambda z{:}\tau_z \,.\, e_b) : \tau}{\vdash \langle \mu f{:}\tau_f \,.\, \lambda z{:}\tau_z \,.\, e_b ; \phi ; \rho \rangle : \tau}$$

$$\frac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet ; \Gamma \vdash \theta(\mu f{:}\tau_f \,.\, \Lambda \beta \,.\, e_b) : \tau}{\vdash \langle \mu f{:}\tau_f \,.\, \Lambda \beta \,.\, e_b ; \phi ; \rho \rangle : \tau}$$

$\boxed{\vdash \rho : \Gamma}$

$$\frac{}{\vdash \bullet : \bullet} \qquad\qquad \frac{\vdash \rho : \Gamma \qquad \vdash w : \tau}{\vdash \rho, x \mapsto w : \Gamma, x{:}\tau}$$

$\boxed{\vdash \tau \rightsquigarrow \kappa : \tau}$

$$\frac{}{\vdash \tau \rightsquigarrow \circ : \tau} \qquad\qquad \frac{\begin{array}{c}\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet \vdash \theta(\tau_z) \qquad \theta(\tau_z) = \tau_r \\ \bullet ; \Gamma, z{:}\theta(\tau_z) \vdash \theta(e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau\end{array}}{\vdash \tau_r \rightsquigarrow \langle z ; \tau_z ; e ; \phi ; \rho \rangle :: \kappa : \tau}$$

$\boxed{\vdash \varsigma : \tau}$

$$\frac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet ; \Gamma \vdash \theta(e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau}{\vdash \langle e ; \phi ; \rho ; \kappa \rangle : \tau}$$

Fig. 6. Static Semantics of ANF System F ($C_a EK$ machine)

Lemma 6
If $\vdash \rho : \Gamma$ *and* $\Gamma(x) = \tau$,
then there exists w *such that* $\vdash w : \tau$ *and* $\rho(x) = w$.

Proof.
By induction on the structure of ρ.

Lemma 7
If $\vdash \rho : \Gamma$, $\Gamma(x) = \tau$, *and* $\rho(x) = w$,
then $\vdash w : \tau$.

Proof.
By Lemma 6 with $\vdash \rho : \Gamma$ and $\Gamma(x) = \tau$,
we have there exists $w^\dagger$ such that $\rho(x) = w^\dagger$ and $\vdash w^\dagger : \tau$.
From $\rho(x) = w$ and $\rho(x) = w^\dagger$, we have $w = w^\dagger$.
From $\vdash w^\dagger : \tau$ and $w = w^\dagger$, we have $\vdash w : \tau$.
Thus, $\vdash w : \tau$.

Theorem 8 (Canonical Forms (Type Soundness)).
If $\vdash w : \tau_a \to \tau_b$,
then $w = \langle \mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b ; \phi_f ; \rho_f \rangle$.
If $\vdash w : \forall \alpha.\ \tau_b$,
then $w = \langle \mu f : \tau_f \,.\, \Lambda \beta \,.\, e_b ; \phi_f ; \rho_f \rangle$.

Proof.
By inspection of the typing rules.

Theorem 9 (Progress (Type Soundness)).
If $\vdash \varsigma : \tau$,
then either there exists x, ϕ, *and* ρ *such that* $\varsigma = \langle x; \phi; \rho; \circ \rangle$
or there exists ς' *such that* $\varsigma \longrightarrow \varsigma'$.

Proof.
Proceed by cases on the structure of ς.

– $\langle x_r; \phi_r; \rho_r; \circ \rangle$:

Take $x = x_r$, $\phi = \phi_r$, and $\rho = \rho_r$.
Thus, $\varsigma = \langle x; \phi; \rho; \circ \rangle$.

– $\langle x_r; \phi_r; \rho_r; \langle z; \tau_z; e; \phi; \rho \rangle :: \kappa \rangle$:

By inversion of $\vdash \langle x_r; \phi_r; \rho_r; \langle z; \tau_z; e; \phi; \rho \rangle :: \kappa \rangle : \tau$,
we have the derivation:

$$\dfrac{\bullet; \Gamma_r \vdash \theta_r(x_r) : \tau_r \qquad \dfrac{\vdash \phi_r : \theta_r \qquad \vdash \rho_r : \Gamma_r \qquad \dfrac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet \vdash \theta(\tau_z) \qquad \theta(\tau_z) = \tau_r \qquad \dfrac{\bullet; \Gamma, z{:}\theta(\tau_z) \vdash \theta(e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau}{\vdash \tau_r \rightsquigarrow \langle z; \tau_z; e; \phi; \rho \rangle :: \kappa : \tau}}{\bullet; \Gamma_r \vdash \theta_r(x_r) : \tau_r}}{\vdash \langle x_r; \phi_r; \rho_r; \langle z; \tau_z; e; \phi; \rho \rangle :: \kappa \rangle : \tau}$$

From $\bullet; \Gamma_r \vdash \theta_r(x_r) : \tau_r$ and $\theta_r(x_r) = x_r$,
we have $\bullet; \Gamma_r \vdash x_r : \tau_r$.
By inversion of $\bullet; \Gamma_r \vdash x_r : \tau_r$, we have the derivation:

$$\dfrac{\bullet \vdash \Gamma_r \qquad \Gamma_r(x_r) = \tau_r}{\bullet; \Gamma_r \vdash x_r : \tau_r}$$

By Lemma 6 with $\vdash \rho_r : \Gamma_r$ and $\Gamma_r(x_r) = \tau_r$,
we have there exists w_r such that $\vdash w_r : \tau_r$ and $\rho_r(x_r) = w_r$.
We can construct the derivation:

$$\dfrac{\rho_r(x_r) = w_r}{\langle x_r; \phi_r; \rho_r; \langle z; \tau_z; e; \phi; \rho \rangle :: \kappa \rangle \longrightarrow \langle e; \phi; \rho, z \mapsto w_r; \kappa \rangle}$$

Take $\varsigma' = \langle e; \phi; \rho, z \mapsto w_r; \kappa \rangle$.
Thus, $\varsigma \longrightarrow \varsigma'$.

– $\langle \texttt{let}\ x{:}\tau_x\ \texttt{=}\ \mu f{:}\tau_f\,.\,\lambda z{:}\tau_z\,.\,e_b\ \texttt{in}\ e; \phi; \rho; \kappa\rangle$:

Let $w_r = \langle \mu f{:}\tau_f\,.\,\lambda z{:}\tau_z\,.\,e_b; \phi; \rho\rangle$.
We can construct the derivation:

$$\frac{w_r = \langle \mu f{:}\tau_f\,.\,\lambda z{:}\tau_z\,.\,e_b; \phi; \rho\rangle}{\langle \texttt{let}\ x{:}\tau_x\ \texttt{=}\ \mu f{:}\tau_f\,.\,\lambda z{:}\tau_z\,.\,e_b\ \texttt{in}\ e; \phi; \rho; \kappa\rangle \longrightarrow \langle e; \phi; \rho, x \mapsto w_r; \kappa\rangle}$$

Take $\varsigma' = \langle e; \phi; \rho, x \mapsto w_r; \kappa\rangle$.
Thus, $\varsigma \longrightarrow \varsigma'$.

– $\langle \texttt{let}\ x{:}\tau_x\ \texttt{=}\ \mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e; \phi; \rho; \kappa\rangle$:

Let $w_r = \langle \mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi; \rho\rangle$.
We can construct the derivation:

$$\frac{w_r = \langle \mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi; \rho\rangle}{\langle \texttt{let}\ x{:}\tau_x\ \texttt{=}\ \mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e; \phi; \rho; \kappa\rangle \longrightarrow \langle e; \phi; \rho, x \mapsto w_r; \kappa\rangle}$$

Take $\varsigma' = \langle e; \phi; \rho, x \mapsto w_r; \kappa\rangle$.
Thus, $\varsigma \longrightarrow \varsigma'$.

– $\langle \texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e; \phi; \rho; \kappa\rangle$:

By inversion of $\vdash \langle \texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e; \phi; \rho; \kappa\rangle : \tau$,
we have the derivation:

$$\frac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet; \Gamma \vdash \theta(\texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau}{\vdash \langle \texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e; \phi; \rho; \kappa\rangle : \tau}$$

From $\bullet; \Gamma \vdash \theta(\texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e) : \tau_e$
and $\theta(\texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e) = \texttt{let}\ x{:}\theta(\tau_x)\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ \theta(e)$,
we have $\bullet; \Gamma \vdash \texttt{let}\ x{:}\theta(\tau_x)\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ \theta(e) : \tau_e$.
By inversion of $\bullet; \Gamma \vdash \texttt{let}\ x{:}\theta(\tau_x)\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ \theta(e) : \tau_e$,
we have the derivation:

$$\frac{\bullet \vdash \Gamma \qquad \Gamma(x_f) = \tau_a \to \tau_b \qquad \Gamma(x_a) = \tau_a}{\bullet; \Gamma \vdash x_f\ x_a : \tau_b}$$

$$\frac{\bullet \vdash \theta(\tau_x) \qquad \theta(\tau_x) = \tau_b \qquad \bullet; \Gamma, x{:}\theta(\tau_x) \vdash e : \tau}{\bullet; \Gamma \vdash \texttt{let}\ x{:}\theta(\tau_x)\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ \theta(e) : \tau_e}$$

By Lemma 6 with $\vdash \rho : \Gamma$ and $\Gamma(x_f) = \tau_a \to \tau_b$,
we have there exists w_f such that $\vdash w_f : \tau_a \to \tau_b$ and $\rho(x_f) = w_f$.
By Theorem 8 with $\vdash w_f : \tau_a \to \tau_b$,
we have $w_f = \langle \mu f{:}\tau_f\,.\,\lambda z{:}\tau_z\,.\,e_b; \phi_f; \rho_f\rangle$.
By Lemma 6 with $\vdash \rho : \Gamma$ and $\Gamma(x_a) = \tau_a$,
we have there exists w_a such that $\vdash w_a : \tau_a$ and $\rho(x_a) = w_a$.
We can construct the derivation:

$$\frac{\rho(x_f) = w_f \qquad w_f = \langle \mu f{:}\tau_f\,.\,\lambda z{:}\tau_z\,.\,e_b; \phi_f; \rho_f\rangle \qquad \rho(x_a) = w_a}{\begin{array}{l}\langle \texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e; \phi; \rho; \kappa\rangle \\ \quad \longrightarrow \langle e_b; \phi_f; \rho_f, f \mapsto w_f, z \mapsto w_a; \langle x; \tau_x; e; \phi; \rho\rangle{::}\kappa\rangle\end{array}}$$

Take $\varsigma' = \langle e_b; \phi_f; \rho_f, f \mapsto w_f, z \mapsto w_a; \langle x; \tau_x; e; \phi; \rho\rangle{::}\kappa\rangle$.
Thus, $\varsigma \longrightarrow \varsigma'$.

– $\langle \texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ \texttt{[}\tau_a\texttt{]}\ \texttt{in}\ e; \phi; \rho; \kappa\rangle$:

By inversion of $\vdash \langle \texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ \texttt{[}\tau_a\texttt{]}\ \texttt{in}\ e; \phi; \rho; \kappa\rangle : \tau$,
we have the derivation:

$$\frac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet; \Gamma \vdash \theta(\texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ \texttt{[}\tau_a\texttt{]}\ \texttt{in}\ e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau}{\vdash \langle \texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ \texttt{[}\tau_a\texttt{]}\ \texttt{in}\ e; \phi; \rho; \kappa\rangle : \tau}$$

From $\bullet; \Gamma \vdash \theta(\texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ \texttt{[}\tau_a\texttt{]}\ \texttt{in}\ e) : \tau_e$
and $\theta(\texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ \texttt{[}\tau_a\texttt{]}\ \texttt{in}\ e) = \texttt{let}\ x{:}\theta(\tau_x)\ \texttt{=}\ x_f\ \texttt{[}\theta(\tau_a)\texttt{]}\ \texttt{in}\ \theta(e)$,
we have $\bullet; \Gamma \vdash \texttt{let}\ x{:}\theta(\tau_x)\ \texttt{=}\ x_f\ \texttt{[}\theta(\tau_a)\texttt{]}\ \texttt{in}\ \theta(e) : \tau_e$.
By inversion of $\bullet; \Gamma \vdash \texttt{let}\ x{:}\theta(\tau_x)\ \texttt{=}\ x_f\ \texttt{[}\theta(\tau_a)\texttt{]}\ \texttt{in}\ \theta(e) : \tau_e$,
we have the derivation:

$$\frac{\bullet \vdash \Gamma \qquad \Gamma(x_f) = \forall\beta.\ \tau_b \qquad \bullet \vdash \theta(\tau_a)}{\bullet; \Gamma \vdash x_f\ \texttt{[}\theta(\tau_a)\texttt{]} : [\beta \rightarrowtail \theta(\tau_a)](\tau_b)}$$

$$\frac{\bullet \vdash \theta(\tau_x) \qquad \theta(\tau_x) = [\beta \rightarrowtail \theta(\tau_a)](\tau_b) \qquad \bullet; \Gamma, x{:}\theta(\tau_x) \vdash e : \tau}{\bullet; \Gamma \vdash \texttt{let}\ x{:}\theta(\tau_x)\ \texttt{=}\ x_f\ \texttt{[}\theta(\tau_a)\texttt{]}\ \texttt{in}\ \theta(e) : \tau_e}$$

By Lemma 6 with $\vdash \rho : \Gamma$ and $\Gamma(x_f) = \forall\beta.\ \tau_b$,
we have there exists w_f such that $\vdash w_f : \forall\beta.\ \tau_b$ and $\rho(x_f) = w_f$.
By Theorem 8 with $\vdash w_f : \forall\beta.\ \tau_b$,
we have $w_f = \langle \mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi_f; \rho_f\rangle$.
Let $\pi_a = \langle \tau_a; \phi\rangle$.
We can construct the derivation:

$$\frac{\rho(x_f) = w_f \qquad w_f = \langle \mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi_f; \rho_f\rangle \qquad \pi_a = \langle \tau_a; \phi\rangle}{\begin{array}{l}\langle \texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ \texttt{[}\tau_a\texttt{]}\ \texttt{in}\ e; \phi; \rho; \kappa\rangle \\ \quad \longrightarrow \langle e_b; \phi_f, \beta \mapsto \pi_a; \rho_f, f \mapsto w_f; \langle x; \tau_x; e; \phi; \rho\rangle{::}\kappa\rangle\end{array}}$$

Take $\varsigma' = \langle e_b; \phi_f, \beta \mapsto \pi_a; \rho_f, f \mapsto w_f; \langle x; \tau_x; e; \phi; \rho\rangle{::}\kappa\rangle$.
Thus, $\varsigma \longrightarrow \varsigma'$.

Theorem 10 (Type Substitution - Types).
If $\Delta_1, \alpha' {:} \star, \Delta_2 \vdash \tau$ *and* $\Delta_1, \Delta_2 \vdash \tau'$,
then $\Delta_1, \Delta_2 \vdash [\alpha' \rightarrowtail \tau']\tau$.

Proof.
By induction on the structure of τ.

Theorem 11 (Type Substitution - Variable Contexts).
If $\Delta_1, \alpha' {:} \star, \Delta_2 \vdash \Gamma$ *and* $\Delta_1, \Delta_2 \vdash \tau'$,
then $\Delta_1, \Delta_2 \vdash [\alpha' \rightarrowtail \tau']\Gamma$.

Proof.
By induction on the structure of Γ,
using Theorem 10.

Theorem 12 (Type Substitution - Variable Context Lookups).
If $\Gamma(x) = \tau$,
then $([\alpha' \rightarrowtail \tau']\Gamma)(x) = [\alpha' \rightarrowtail \tau']\tau$.

Proof.
By induction on the structure of Γ.

Theorem 13 (Type Substitution - Expressions & Binds).
If $\Delta_1, \alpha' {:} \star, \Delta_2; \Gamma \vdash e : \tau$ *and* $\Delta_1, \Delta_2 \vdash \tau'$,
then $\Delta_1, \Delta_2; [\alpha' \rightarrowtail \tau']\Gamma \vdash [\alpha' \rightarrowtail \tau']e : [\alpha' \rightarrowtail \tau']\tau$.
If $\Delta_1, \alpha' {:} \star, \Delta_2; \Gamma \vdash b : \tau$ *and* $\Delta_1, \Delta_2 \vdash \tau'$,
then $\Delta_1, \Delta_2; [\alpha' \rightarrowtail \tau']\Gamma \vdash [\alpha' \rightarrowtail \tau']b : [\alpha' \rightarrowtail \tau']\tau$.

Proof.
By mutual induction on the structures of e and b,
using Theorems 10, 11, and 12.

Theorem 14 (Preservation (Type Soundness)).
If $\vdash \varsigma : \tau$ *and* $\varsigma \longrightarrow \varsigma'$,
then $\vdash \varsigma' : \tau$.

Proof.
Proceed by cases on the derivation of $\varsigma \longrightarrow \varsigma'$.

– $\dfrac{w_r = \rho_r(x_r)}{\langle x_r; \phi_r; \rho_r; \langle z; \tau_z; e; \phi; \rho\rangle{::}\kappa\rangle \longrightarrow \langle e; \phi; \rho, z \mapsto w_r; \kappa\rangle}$:

By inversion of $\vdash \langle x_r; \phi_r; \rho_r; \langle z; \tau_z; e; \phi; \rho\rangle{::}\kappa\rangle : \tau$,
we have the derivation:

$$\dfrac{\bullet; \Gamma_r \vdash \theta_r(x_r) : \tau_r \qquad \dfrac{\vdash \phi_r : \theta_r \qquad \vdash \rho_r : \Gamma_r \qquad \dfrac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet \vdash \theta(\tau_z) \qquad \theta(\tau_z) = \tau_r \qquad \dfrac{\bullet; \Gamma, z{:}\theta(\tau_z) \vdash \theta(e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau}{\vdash \tau_r \rightsquigarrow \langle z; \tau_z; e; \phi; \rho\rangle{::}\kappa : \tau}}{\vdash \langle x_r; \phi_r; \rho_r; \langle z; \tau_z; e; \phi; \rho\rangle{::}\kappa\rangle : \tau}}$$

From $\bullet; \Gamma_r \vdash \theta_r(x_r) : \tau_r$ and $\theta_r(x_r) = x_r$,
we have $\bullet; \Gamma_r \vdash x_r : \tau_r$.
By inversion of $\bullet; \Gamma_r \vdash x_r : \tau_r$, we have the derivation:

$$\dfrac{\bullet \vdash \Gamma_r \qquad \Gamma_r(x_r) = \tau_r}{\bullet; \Gamma_r \vdash x_r : \tau_r}$$

By Lemma 7 with $\vdash \rho_r : \Gamma_r$, $\Gamma_r(x_r) = \tau_r$, and $\rho_r(x_r) = w_r$
we have $\vdash w_r : \tau_r$.
From $\vdash w_r : \tau_r$ and $\theta(\tau_z) = \tau_r$,
we have $\vdash w_r : \theta(\tau_z)$.
We can construct the derivation:

$$\dfrac{\vdash \phi : \theta \qquad \dfrac{\vdash \rho : \Gamma \qquad \vdash w_r : \theta(\tau_z)}{\vdash \rho, z \mapsto w_r : \Gamma, z{:}\theta(\tau_z)} \qquad \dfrac{\bullet; \Gamma, z{:}\theta(\tau_z) \vdash \theta(e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau}{\vdash \langle e; \phi; \rho, z \mapsto w_r; \kappa\rangle : \tau}}{\vdash \langle e; \phi; \rho, z \mapsto w_r; \kappa\rangle : \tau}$$

Thus, $\vdash \langle e; \phi; \rho, z \mapsto w_r; \kappa\rangle : \tau$.

– $\dfrac{w_r = \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi; \rho\rangle}{\langle \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\ \texttt{in}\ e; \phi; \rho; \kappa\rangle \longrightarrow \langle e; \phi; \rho, x \mapsto w_r; \kappa\rangle}$:

By inversion of $\vdash \langle \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\ \texttt{in}\ e; \phi; \rho; \kappa\rangle : \tau$,
we have the derivation:

$$\dfrac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet; \Gamma \vdash \theta(\texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\ \texttt{in}\ e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau}{\vdash \langle \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\ \texttt{in}\ e; \phi; \rho; \kappa\rangle : \tau}$$

From $\bullet; \Gamma \vdash \theta(\texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\ \texttt{in}\ e) : \tau_e$
and $\theta(\texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\ \texttt{in}\ e) = \texttt{let}\ x\!:\!\theta(\tau_x)\ \texttt{=}\ \theta(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b)\ \texttt{in}\ \theta(e)$,
we have $\bullet; \Gamma \vdash \texttt{let}\ x\!:\!\theta(\tau_x)\ \texttt{=}\ \theta(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b)\ \texttt{in}\ \theta(e) : \tau_e$.
By inversion of $\bullet; \Gamma \vdash \texttt{let}\ x\!:\!\theta(\tau_x)\ \texttt{=}\ \theta(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b)\ \texttt{in}\ \theta(e) : \tau_e$,
we have the derivation:

$$\dfrac{\bullet; \Gamma \vdash \theta(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b) : \tau_s \qquad \bullet \vdash \theta(\tau_x) \qquad \theta(\tau_x) = \tau_s \qquad \bullet; \Gamma, x\!:\!\theta(\tau_x) \vdash \theta(e) : \tau_e}{\bullet; \Gamma \vdash \texttt{let}\ x\!:\!\theta(\tau_x)\ \texttt{=}\ \theta(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b)\ \texttt{in}\ \theta(e) : \tau_e}$$

From $\bullet; \Gamma \vdash \theta(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b) : \tau_s$ and $\theta(\tau_x) = \tau_s$,
we have $\bullet; \Gamma \vdash \theta(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b) : \theta(\tau_x)$.
We can construct the derivation:

$$\dfrac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet; \Gamma \vdash \theta(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b) : \theta(\tau_x)}{\vdash \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi; \rho\rangle : \theta(\tau_x)}$$

From $\vdash \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi; \rho\rangle : \theta(\tau_x)$ and $w_r = \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi; \rho\rangle$,
we have $\vdash w_r : \theta(\tau_x)$.
We can construct the derivation:

$$\dfrac{\vdash \phi : \theta \qquad \dfrac{\vdash \rho : \Gamma \qquad \vdash w_r : \theta(\tau_x)}{\vdash \rho, x \mapsto w_r : \Gamma, x\!:\!\theta(\tau_x)} \qquad \bullet; \Gamma, x\!:\!\theta(\tau_x) \vdash \theta(e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau}{\vdash \langle e; \phi; \rho, x \mapsto w_r; \kappa\rangle : \tau}$$

Thus, $\vdash \langle e; \phi; \rho, x \mapsto w_r; \kappa\rangle : \tau$.

– $\dfrac{w_r = \langle \mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi; \rho\rangle}{\langle \texttt{let}\ x{:}\tau_x\ \texttt{=}\ \mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e; \phi; \rho; \kappa\rangle \longrightarrow \langle e; \phi; \rho, x \mapsto w_r; \kappa\rangle}$:

By inversion of $\vdash \langle \texttt{let}\ x{:}\tau_x\ \texttt{=}\ \mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e; \phi; \rho; \kappa\rangle : \tau$,
we have the derivation:

$$\frac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet; \Gamma \vdash \theta(\texttt{let}\ x{:}\tau_x\ \texttt{=}\ \mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau}{\vdash \langle \texttt{let}\ x{:}\tau_x\ \texttt{=}\ \mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e; \phi; \rho; \kappa\rangle : \tau}$$

From $\bullet; \Gamma \vdash \theta(\texttt{let}\ x{:}\tau_x\ \texttt{=}\ \mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e) : \tau_e$
and $\theta(\texttt{let}\ x{:}\tau_x\ \texttt{=}\ \mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e) = \texttt{let}\ x{:}\theta(\tau_x)\ \texttt{=}\ \theta(\mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b)\ \texttt{in}\ \theta(e)$,
we have $\bullet; \Gamma \vdash \texttt{let}\ x{:}\theta(\tau_x)\ \texttt{=}\ \theta(\mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b)\ \texttt{in}\ \theta(e) : \tau_e$.
By inversion of $\bullet; \Gamma \vdash \texttt{let}\ x{:}\theta(\tau_x)\ \texttt{=}\ \theta(\mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b)\ \texttt{in}\ \theta(e) : \tau_e$,
we have the derivation:

$$\frac{\bullet; \Gamma \vdash \theta(\mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b) : \tau_s \qquad \bullet \vdash \theta(\tau_x) \qquad \theta(\tau_x) = \tau_s \qquad \bullet; \Gamma, x{:}\theta(\tau_x) \vdash \theta(e) : \tau_e}{\bullet; \Gamma \vdash \texttt{let}\ x{:}\theta(\tau_x)\ \texttt{=}\ \theta(\mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b)\ \texttt{in}\ \theta(e) : \tau_e}$$

From $\bullet; \Gamma \vdash \theta(\mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b) : \tau_s$ and $\theta(\tau_x) = \tau_s$,
we have $\bullet; \Gamma \vdash \theta(\mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b) : \theta(\tau_x)$.
We can construct the derivation:

$$\frac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet; \Gamma \vdash \theta(\mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b) : \theta(\tau_x)}{\vdash \langle \mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi; \rho\rangle : \theta(\tau_x)}$$

From $\vdash \langle \mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi; \rho\rangle : \theta(\tau_x)$ and $w_r = \langle \mu f{:}\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi; \rho\rangle$,
we have $\vdash w_r : \theta(\tau_x)$.
We can construct the derivation:

$$\frac{\vdash \phi : \theta \qquad \dfrac{\vdash \rho : \Gamma \qquad \vdash w_r : \theta(\tau_x)}{\vdash \rho, x \mapsto w_r : \Gamma, x{:}\theta(\tau_x)} \qquad \bullet; \Gamma, x{:}\theta(\tau_x) \vdash \theta(e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau}{\vdash \langle e; \phi; \rho, x \mapsto w_r; \kappa\rangle : \tau}$$

Thus, $\vdash \langle e; \phi; \rho, x \mapsto w_r; \kappa\rangle : \tau$.

– $\dfrac{\rho(x_f) = w_f \qquad w_f = \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi_f; \rho_f\rangle \qquad \rho(x_a) = w_a}{\langle \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e; \phi; \rho; \kappa\rangle \longrightarrow \langle e_b; \phi_f; \rho_f, f \mapsto w_f, z \mapsto w_a; \langle x; \tau_x; e; \phi; \rho\rangle{::}\kappa\rangle}$:

By inversion of $\vdash \langle \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e; \phi; \rho; \kappa\rangle : \tau$,
we have the derivation:

$$\dfrac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet; \Gamma \vdash \theta(\texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau}{\vdash \langle \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e; \phi; \rho; \kappa\rangle : \tau}$$

From $\bullet; \Gamma \vdash \theta(\texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e) : \tau_e$
and $\theta(\texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e) = \texttt{let}\ x\!:\!\theta(\tau_x)\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ \theta(e)$,
we have $\bullet; \Gamma \vdash \texttt{let}\ x\!:\!\theta(\tau_x)\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ \theta(e) : \tau_e$.
By inversion of $\bullet; \Gamma \vdash \texttt{let}\ x\!:\!\theta(\tau_x)\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ \theta(e) : \tau_e$,
we have the derivation:

$$\dfrac{\bullet \vdash \Gamma \qquad \Gamma(x_f) = \tau_a \to \tau_b \qquad \Gamma(x_a) = \tau_a}{\bullet; \Gamma \vdash x_f\ x_a : \tau_b}$$

$$\dfrac{\bullet \vdash \theta(\tau_x) \qquad \theta(\tau_x) = \tau_b \qquad \bullet; \Gamma, x{:}\theta(\tau_x) \vdash \theta(e) : \tau_e}{\bullet; \Gamma \vdash \texttt{let}\ x\!:\!\theta(\tau_x)\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ \theta(e) : \tau_e}$$

By Lemma 7 with $\vdash \rho : \Gamma$, $\Gamma(x_f) = \tau_a \to \tau_b$, $\rho(x_f) = w_f$,
we have $\vdash w_f : \tau_a \to \tau_b$.
From $\vdash w_f : \tau_a \to \tau_b$ and $w_f = \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi_f; \rho_f\rangle$,
we have $\vdash \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi_f; \rho_f\rangle : \tau_a \to \tau_b$.
By inversion of $\vdash \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi_f; \rho_f\rangle : \tau_a \to \tau_b$,
we have the derivation:

$$\dfrac{\vdash \phi_f : \theta_f \qquad \vdash \rho_f : \Gamma_f \qquad \bullet; \Gamma_f \vdash \theta_f(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b) : \tau_a \to \tau_b}{\vdash \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi_f; \rho_f\rangle : \tau_a \to \tau_b}$$

From $\bullet; \Gamma_f \vdash \theta_f(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b) : \tau_a \to \tau_b$
and $\theta_f(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b) = \mu f\!:\!\theta_f(\tau_f)\,.\,\lambda z\!:\!\theta_f(\tau_z)\,.\,\theta_f(e_b)$,
we have $\bullet; \Gamma_f \vdash \mu f\!:\!\theta_f(\tau_f)\,.\,\lambda z\!:\!\theta_f(\tau_z)\,.\,\theta_f(e_b) : \tau_a \to \tau_b$.
By inversion of $\bullet; \Gamma_f \vdash \mu f\!:\!\theta_f(\tau_f)\,.\,\lambda z\!:\!\theta_f(\tau_z)\,.\,\theta_f(e_b) : \tau_a \to \tau_b$,
we have the derivation:

$$\dfrac{\begin{array}{c}\bullet \vdash \Gamma_f \qquad \bullet \vdash \theta_f(\tau_f) \qquad \bullet \vdash \theta_f(\tau_z) \\ \bullet; \Gamma_f, f{:}\theta_f(\tau_f), z{:}\theta_f(\tau_z) \vdash \theta_f(e_b) : \tau_b \\ \theta_f(\tau_f) = \tau_a \to \tau_b \qquad \theta_f(\tau_z) = \tau_a\end{array}}{\bullet; \Gamma_f \vdash \mu f\!:\!\theta_f(\tau_f)\,.\,\lambda z\!:\!\theta_f(\tau_z)\,.\,\theta_f(e_b) : \tau_a \to \tau_b}$$

From $\vdash w_f : \tau_a \to \tau_b$ and $\theta_f(\tau_f) = \tau_a \to \tau_b$,
we have $\vdash w_f : \theta_f(\tau_f)$.
By Lemma 7 with $\vdash \rho : \Gamma$, $\Gamma(x_a) = \tau_a$, and $\rho(x_a) = w_a$,
we have $\vdash w_a : \tau_a$.

From $\vdash w_a : \tau_a$ and $\theta_f(\tau_z) = \tau_a$,
we have $\vdash w_a : \theta_f(\tau_z)$.
We can construct the derivation:

$$\dfrac{\vdash \phi_f : \theta_f \qquad \dfrac{\dfrac{\vdash \rho_f : \Gamma_f \qquad \vdash w_f : \theta_f(\tau_f)}{\vdash \rho_f, f \mapsto w_f : \Gamma_f, f{:}\theta_f(\tau_f)} \qquad \vdash w_a : \theta_f(\tau_z)}{\vdash \rho_f, f \mapsto w_f, z \mapsto w_a : \Gamma_f, f{:}\theta_f(\tau_f), z{:}\theta_f(\tau_z)}}{\bullet; \Gamma_f, f{:}\theta_f(\tau_f), z{:}\theta_f(\tau_z) \vdash \theta_f(e_b) : \tau_b}$$

$$\dfrac{\dfrac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet \vdash \theta(\tau_x) \qquad \theta(\tau_x) = \tau_b \qquad \Gamma, x{:}\theta(\tau_x) \vdash \theta(e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau}{\vdash \tau_b \rightsquigarrow \langle x; \tau_x; e; \phi; \rho\rangle{::}\kappa : \tau}}{\vdash \langle e_b; \phi_f; \rho_f, f \mapsto w_f, z \mapsto w_a; \langle x; \tau_x; e; \phi; \rho\rangle{::}\kappa\rangle : \tau}$$

Thus, $\vdash \langle e_b; \phi_f; \rho_f, f \mapsto w_f, z \mapsto w_a; \langle x; \tau_x; e; \phi; \rho\rangle{::}\kappa\rangle : \tau$.

– $\dfrac{\rho(x_f) = w_f \qquad w_f = \langle \mu f{:}\tau_f \,.\, \Lambda\beta \,.\, e_b; \phi_f; \rho_f\rangle \qquad \pi_a = \langle \tau_a; \phi\rangle}{\begin{array}{l}\langle \texttt{let } x{:}\tau_x \texttt{ = } x_f \texttt{ [}\tau_a\texttt{] in } e; \phi; \rho; \kappa\rangle \\ \qquad \longrightarrow \langle e_b; \phi_f, \beta \mapsto \pi_a; \rho_f, f \mapsto w_f; \langle x; \tau_x; e; \phi; \rho\rangle{::}\kappa\rangle\end{array}}$:

By inversion of $\vdash \langle \texttt{let } x{:}\tau_x \texttt{ = } x_f \texttt{ [}\tau_a\texttt{] in } e; \phi; \rho; \kappa\rangle : \tau$,
we have the derivation:

$$\dfrac{\begin{array}{c}\vdash \phi : \theta \qquad \vdash \rho : \Gamma \\ \bullet; \Gamma \vdash \theta(\texttt{let } x{:}\tau_x \texttt{ = } x_f \texttt{ [}\tau_a\texttt{] in } e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau\end{array}}{\vdash \langle \texttt{let } x{:}\tau_x \texttt{ = } x_f \texttt{ [}\tau_a\texttt{] in } e; \phi; \rho\rangle{::}\kappa : \tau}$$

From $\bullet; \Gamma \vdash \theta(\texttt{let } x{:}\tau_x \texttt{ = } x_f \texttt{ [}\tau_a\texttt{] in } e) : \tau_e$
and $\theta(\texttt{let } x{:}\tau_x \texttt{ = } x_f \texttt{ [}\tau_a\texttt{] in } e) = \texttt{let } x{:}\theta(\tau_x) \texttt{ = } x_f \texttt{ [}\theta(\tau_a)\texttt{] in } \theta(e)$,
we have $\bullet; \Gamma \vdash \texttt{let } x{:}\theta(\tau_x) \texttt{ = } x_f \texttt{ [}\theta(\tau_a)\texttt{] in } \theta(e) : \tau_e$.
By inversion of $\bullet; \Gamma \vdash \texttt{let } x{:}\theta(\tau_x) \texttt{ = } x_f \texttt{ [}\theta(\tau_a)\texttt{] in } \theta(e) : \tau_e$,
we have the derivation:

$$\dfrac{\bullet \vdash \Gamma \qquad \Gamma(x_f) = \forall\beta.\ \tau_b \qquad \bullet \vdash \theta(\tau_a)}{\bullet; \Gamma \vdash x_f \texttt{ [}\theta(\tau_a)\texttt{]} : [\beta \rightarrowtail \theta(\tau_a)]\tau_b}$$

$$\dfrac{\bullet \vdash \theta(\tau_x) \qquad \theta(\tau_x) = [\beta \rightarrowtail \theta(\tau_a)]\tau_b \qquad \bullet; \Gamma, x{:}\theta(\tau_x) \vdash \theta(e) : \tau_e}{\bullet; \Gamma \vdash \texttt{let } x{:}\theta(\tau_x) \texttt{ = } x_f \texttt{ [}\theta(\tau_a)\texttt{] in } \theta(e) : \tau_e}$$

By Lemma 7 with $\vdash \rho : \Gamma$, $\Gamma(x_f) = \forall\beta.\ \tau_b$, and $\rho(x_f) = w_f$,
we have $\vdash w_f : \forall\beta.\ \tau_b$.
From $\vdash w_f : \forall\beta.\ \tau_b$ and $w_f = \langle \mu f{:}\tau_f \,.\, \Lambda\beta \,.\, e_b; \phi_f; \rho_f\rangle$,
we have $\vdash \langle \mu f{:}\tau_f \,.\, \Lambda\beta \,.\, e_b; \phi_f; \rho_f\rangle : \forall\beta.\ \tau_b$.
By inversion of $\vdash \langle \mu f{:}\tau_f \,.\, \Lambda\beta \,.\, e_b; \phi_f; rho_f\rangle : \forall\beta.\ \tau_b$, we have the derivation:

$$\dfrac{\vdash \phi_f : \theta_f \qquad \vdash \rho_f : \Gamma_f \qquad \bullet; \Gamma_f \vdash \theta_f(\mu f{:}\tau_f \,.\, \Lambda\beta \,.\, e_b) : \forall\beta.\ \tau_b}{\vdash \langle \mu f{:}\tau_f \,.\, \Lambda\beta \,.\, e_b; \phi_f; \rho_f\rangle : \forall\beta.\ \tau_b}$$

From $\bullet; \Gamma_f \vdash \theta_f(\mu f{:}\tau_f \,.\, \Lambda\beta \,.\, e_b) : \forall\beta.\ \tau_b$
and $\theta_f(\mu f{:}\tau_f \,.\, \Lambda\beta \,.\, e_b) = \mu f{:}\theta_f(\tau_f) \,.\, \Lambda\beta \,.\, \theta'_f(e_b)$,
where $\theta'_f = \theta_f|_{\mathsf{dom}(\theta_f)\setminus\{\beta\}}$,
we have $\bullet; \Gamma_f \vdash \mu f{:}\theta_f(\tau_f) \,.\, \Lambda\beta \,.\, \theta'_f(e_b) : \forall\beta.\ \tau_b$.
By inversion of $\bullet; \Gamma_f \vdash \mu f{:}\theta_f(\tau_f) \,.\, \Lambda\beta \,.\, \theta'_f(e_b) : \forall\beta.\ \tau_b$, we have the derivation:

$$\dfrac{\begin{array}{c}\bullet \vdash \Gamma_f \qquad \bullet \vdash \theta_f(\tau_f) \\ \bullet, \beta{:}\star; \Gamma_f, f{:}\theta_f(\tau_f) \vdash \theta'_f(e_b) : \tau_b \\ \theta_f(\tau_f) = \forall\beta.\ \tau_b\end{array}}{\bullet; \Gamma_f \vdash \mu f{:}\theta_f(\tau_f) \,.\, \Lambda\beta \,.\, \theta'_f(e_b) : \forall\beta.\ \tau_b}$$

From $\vdash w_f : \forall\beta.\ \tau_b$ and $\theta_f(\tau_f) = \forall\beta.\ \tau_b$,
we have $\vdash w_f : \theta_f(\tau_f)$.
By Theorem 13 with $\bullet, \beta{:}\star; \Gamma_f, f{:}\theta_f(\tau_f) \vdash \theta'_f(e_b) : \tau_b$ and $\bullet \vdash \theta(\tau_a)$,
we have $\bullet; [\beta \rightarrowtail \theta(\tau_a)](\Gamma_f, f{:}\theta_f(\tau_f)) \vdash [\beta \rightarrowtail \theta(\tau_a)](\theta'_f(e_b)) : [\beta \rightarrowtail \theta(\tau_a)]\tau_b$.

From $\vdash \rho_f : \Gamma_f$,
we have $[\beta \rightarrowtail \theta(\tau_a)]\Gamma_f = \Gamma_f$.
From $\bullet \vdash \theta_f(\tau_f)$,
we have $[\beta \rightarrowtail \theta(\tau_a)](\theta_f(\tau_f)) = \theta_f(\tau_f)$.
From $\vdash \phi_f : \theta_f$, $\theta'_f = \theta_f|_{\mathsf{dom}(\theta_f)\setminus\{\beta\}}$, and $\vdash \theta(\tau_a)$,
we have $[\beta \rightarrowtail \theta(\tau_a)](\theta'_f(e_b)) = [\theta_f, \beta \rightarrowtail \theta(\tau_a)]e_b$.
From $\bullet; [\beta \rightarrowtail \theta(\tau_a)](\Gamma_f, f{:}\theta_f(\tau_f)) \vdash [\beta \rightarrowtail \theta(\tau_a)](\theta'_f(e_b)) : [\beta \rightarrowtail \theta(\tau_a)]\tau_b$,
$[\beta \rightarrowtail \theta(\tau_a)]\Gamma_f = \Gamma_f$, $[\beta \rightarrowtail \theta(\tau_a)](\theta_f(\tau_f)) = \theta_f(\tau_f)$,
and $[\beta \rightarrowtail \theta(\tau_a)](\theta'_f(e_b)) = [\theta_f, \beta \rightarrowtail \theta(\tau_a)]e_b$,
we have $\bullet; \Gamma_f, f{:}\theta_f(\tau_f) \vdash [\theta_f, \beta \rightarrowtail \theta(\tau_a)]e_b : [\beta \rightarrowtail \theta(\tau_a)]\tau_b$.
We can construct the derivation:

$$\frac{\vdash \phi : \theta \qquad \bullet \vdash \theta(\tau_a)}{\vdash \langle \tau_a; \phi \rangle \Rrightarrow \theta(\tau_a)}$$

From $\vdash \langle \tau_a; \phi \rangle \Rrightarrow \theta(\tau_a)$ and $\pi_a = \langle \tau_a; \phi \rangle$,
we have $\vdash \pi_a \Rrightarrow \theta(\tau_a)$.
We can construct the derivation:

$$\frac{\vdash \phi_f : \theta_f \qquad \vdash \pi_a \Rrightarrow \theta(\tau_a)}{\vdash \phi_f, \beta \mapsto \pi_a : \theta_f, \beta \rightarrowtail \theta(\tau_a)} \qquad \frac{\vdash \rho_f : \Gamma_f \qquad \vdash w_f : \theta_f(\tau_f)}{\vdash \rho_f, f \mapsto w_f : \Gamma_f, f{:}\theta_f(\tau_f)}$$

$$\bullet; \Gamma_f, f{:}\theta_f(\tau_f) \vdash [\theta_f, \beta \rightarrowtail \theta(\tau_a)](e_b) : [\beta \rightarrowtail \theta(\tau_a)](\tau_b)$$

$$\frac{\dfrac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet \vdash \theta(\tau_x) \qquad \theta(\tau_x) = [\beta \rightarrowtail \theta(\tau_a)]\tau_b \qquad \bullet; \Gamma, x{:}\theta(\tau_x) \vdash \theta(e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau}{\vdash [\beta \rightarrowtail \theta(\tau_a)]\tau_b \rightsquigarrow \langle x; \tau_x; e; \phi; \rho \rangle{::}\kappa : \tau}}{\vdash \langle e_b; \phi_f, \beta \mapsto \langle \tau_a; \phi \rangle; \rho_f, f \mapsto w_f; \langle x; \tau_x; e; \phi; \rho \rangle{::}\kappa \rangle : \tau}$$

Thus, $\vdash \langle e_b; \phi_f, \beta \mapsto \langle \tau_a; \phi \rangle; \rho_f, f \mapsto w_f; \langle x; \tau_x; e; \phi; \rho \rangle{::}\kappa \rangle : \tau$.

Theorem 15 (Preservation$*$ (Type Soundness)).
If $\vdash \varsigma : \tau$ *and* $\varsigma \longrightarrow^* \varsigma'$,
then $\vdash \varsigma' : \tau$.

Proof.
By induction on the derivation of $\varsigma \longrightarrow^* \varsigma'$, using Theorem 14.

Theorem 1 (Type Soundness).
If $\bullet; \bullet \vdash e : \tau$ *and* $\langle e; \bullet; \bullet; \circ \rangle \longrightarrow^* \varsigma'$,
then either there exists x', ϕ', *and* ρ' *such that* $\varsigma' = \langle x'; \phi'; \rho'; \circ \rangle$
or there exists ς' *such that* $\varsigma \longrightarrow \varsigma'$.

Proof.
From $\bullet; \bullet \vdash e : \tau$ and $e = [\bullet](e)$, we have $\bullet; \bullet \vdash [\bullet](e) : \tau$.
We can construct the derivation:

$$\frac{\dfrac{}{\vdash \bullet : \bullet} \qquad \dfrac{}{\vdash \bullet : \bullet} \qquad \bullet; \bullet \vdash [\bullet](e) : \tau \qquad \dfrac{}{\vdash \tau \rightsquigarrow \circ : \tau}}{\vdash \langle e; \bullet; \bullet; \circ \rangle : \tau}$$

By Theorem 15 with $\vdash \langle e; \bullet; \bullet; \circ \rangle : \tau$ and $\langle e; \bullet; \bullet; \circ \rangle \longrightarrow^* \varsigma'$, we have $\vdash \varsigma' : \tau$.
By Theorem 9 with $\vdash \varsigma' : \tau$, we have either $\varsigma' = \langle x'; \phi'; \rho'; \circ \rangle$ or $\varsigma' \longrightarrow \varsigma''$.

B Flow Soundness

$\boxed{\hat{\phi} \vDash \pi}$

$$\frac{\hat{\phi} \vDash \phi}{\hat{\phi} \vDash \langle \tau ; \phi \rangle}$$

$\boxed{\hat{\phi} \vDash \phi}$

$$\frac{}{\hat{\phi} \vDash \bullet} \qquad \frac{\hat{\phi} \vDash \phi \qquad \hat{\phi} \vDash \pi \qquad |\pi| \in \hat{\phi}(\alpha)}{\hat{\phi} \vDash \phi, \alpha \mapsto \pi}$$

$\boxed{\hat{\phi}; \hat{\rho} \vDash w}$

$$\frac{\{\mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b\} \in \hat{\rho}(f) \qquad \hat{\phi}; \hat{\rho} \vDash e_b \qquad \hat{\phi} \vDash \phi \qquad \hat{\phi}; \hat{\rho} \vDash \rho}{\hat{\phi}; \hat{\rho} \vDash \langle \mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b ; \phi ; \rho \rangle} \qquad \frac{\{\mu f : \tau_f \,.\, \Lambda \beta \,.\, e_b\} \in \hat{\rho}(f) \qquad \hat{\phi}; \hat{\rho} \vDash e_b \qquad \hat{\phi} \vDash \phi \qquad \hat{\phi}; \hat{\rho} \vDash \rho}{\hat{\phi}; \hat{\rho} \vDash \langle \mu f : \tau_f \,.\, \Lambda \beta \,.\, e_b ; \phi ; \rho \rangle}$$

$\boxed{\hat{\phi}; \hat{\rho} \vDash \rho}$

$$\frac{}{\hat{\phi}; \hat{\rho} \vDash \bullet} \qquad \frac{\hat{\phi}; \hat{\rho} \vDash \rho \qquad \hat{\phi}; \hat{\rho} \vDash w \qquad |w| \in \hat{\rho}(x)}{\hat{\phi}; \hat{\rho} \vDash \rho, x \mapsto w}$$

$\boxed{\hat{\phi}; \hat{\rho} \vDash \hat{W} \rightsquigarrow \kappa}$

$$\frac{}{\hat{\phi}; \hat{\rho} \vDash \hat{W} \rightsquigarrow \circ} \qquad \frac{\{\hat{w} \in \hat{W} \mid \vdash \hat{\phi} \Rrightarrow \hat{w} :\approx \tau_z\} \subseteq \hat{\rho}(z) \qquad \hat{\phi}; \hat{\rho} \vDash e \qquad \hat{\phi} \vDash \phi \qquad \hat{\phi}; \hat{\rho} \vDash \rho \qquad \hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor e \rfloor) \rightsquigarrow \kappa}{\hat{\phi}; \hat{\rho} \vDash \hat{W} \rightsquigarrow \langle z ; \tau_z ; e ; \phi ; \rho \rangle :: \kappa}$$

$\boxed{\hat{\phi}; \hat{\rho} \vDash \varsigma}$

$$\frac{\hat{\phi}; \hat{\rho} \vDash e \qquad \hat{\phi} \vDash \phi \qquad \hat{\phi}; \hat{\rho} \vDash \rho \qquad \hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor e \rfloor) \rightsquigarrow \kappa}{\hat{\phi}; \hat{\rho} \vDash \langle e ; \phi ; \rho ; \kappa \rangle}$$

Fig. 7. Type- and Control-Flow Analysis of ANF System F (C_aEK machine)

Lemma 16
If $\hat{\phi} \vDash \phi$, *then* $|\phi| \sqsubseteq \hat{\phi}$.

Proof.
By induction on the structure of ϕ.

Lemma 17
If $\hat{\phi}; \hat{\rho} \vDash \rho$, *then* $|\rho| \sqsubseteq \hat{\rho}$.

Proof.
By induction on the structure of ρ.

Lemma 18
If $\hat{\phi}; \hat{\rho} \vDash \rho$ *and* $\rho(x) = w$, *then* $\hat{\phi}; \hat{\rho} \vDash w$ *and* $|w| \in \hat{\rho}(x)$.

Proof.
By induction on the structure of ρ.

Lemma 19
If $\vdash \phi : \theta$, $\hat{\phi} \vDash \phi$, $\Delta \vdash \theta(\tau)$, *and* $\mathsf{dom}(\phi) \cap \mathsf{dom}(\Delta) = \emptyset$,
then $\Delta; \hat{\phi} \vdash \tau \twoheadrightarrow \theta(\tau)$.

Proof.
By well-founded induction on $\langle \mathit{RTEnv} \times \mathit{Type}; \prec \rangle$,
where $\prec$ is the well-founded relation
formed by the lexicographic combination of $\prec_\phi$ and $\prec_\tau$:

$$\frac{\phi' \prec_\phi \phi}{\langle \phi'; \tau' \rangle \prec \langle \phi; \tau \rangle} \qquad \frac{\tau' \prec_\tau \tau}{\langle \phi; \tau' \rangle \prec \langle \phi; \tau \rangle}$$

where $\prec_\phi$ is the well-founded proper-subterm relation on *RTEnv*:

$$\frac{}{\phi' \prec \phi, \alpha \mapsto \langle \tau'; \phi' \rangle} \qquad \frac{}{\phi \prec \phi, \alpha \mapsto \langle \tau'; \phi' \rangle}$$

and $\prec_\tau$ is the well-founded proper-subterm relation on *Type*:

$$\frac{}{\tau_a \prec \tau_a \to \tau_b} \qquad \frac{}{\tau_b \prec \tau_a \to \tau_b} \qquad \frac{}{\tau_b \prec \forall \alpha.\ \tau_b}$$

Proceed by cases on the structure of τ.

- α:

  Either $\alpha \notin \mathsf{dom}(\phi)$ or $\alpha \in \mathsf{dom}(\phi)$.

  - $\alpha \notin \mathsf{dom}(\phi)$:

    From $\alpha \notin \mathsf{dom}(\phi)$ and $\vdash \phi : \theta$,
    we have $\theta(\alpha) = \alpha$.
    From $\Delta \vdash \theta(\alpha)$ and $\theta(\alpha) = \alpha$,
    we have $\Delta \vdash \alpha$.
    By inversion of $\Delta \vdash \alpha$,
    we have the derivation:

    $$\frac{\vdash \Delta \qquad \Delta(\alpha) = \star}{\Delta \vdash \alpha}$$

    We can construct the derivation:

    $$\frac{\vdash \Delta \qquad \Delta(\alpha) = \star}{\Delta; \hat{\phi} \vdash \alpha \twoheadrightarrow \alpha}$$

    From $\Delta; \hat{\phi} \vdash \alpha \twoheadrightarrow \alpha$ and $\theta(\alpha) = \alpha$,
    we have $\Delta; \hat{\phi} \vdash \alpha \twoheadrightarrow \theta(\alpha)$.
    Thus, $\Delta; \hat{\phi} \vdash \alpha \twoheadrightarrow \theta(\alpha)$.

- $\alpha \in \mathsf{dom}(\phi)$:

  Proceed by cases on the structure of ϕ.
  * $\bullet$:

    By inversion of $\alpha \in \mathsf{dom}(\bullet)$,
    we have a contradiction. $\Rightarrow\Leftarrow$.

  * $\phi'', \beta \mapsto \langle \tau'; \phi' \rangle$:

    By inversion of $\vdash \phi'', \beta \mapsto \langle \tau'; \phi' \rangle : \theta$,
    we have the derivation:

$$\dfrac{\vdash \phi'' : \theta'' \qquad \dfrac{\vdash \phi' : \theta' \qquad \bullet \vdash \theta'(\tau')}{\vdash \langle \tau'; \phi' \rangle \Rrightarrow \theta'(\tau')}}{\vdash \phi'', \beta \mapsto \langle \tau'; \phi' \rangle : \theta'', \beta \rightarrowtail \theta'(\tau')}$$

    and $\theta = \theta'', \beta \rightarrowtail \theta'(\tau')$.
    By inversion of $\hat{\phi} \vDash \phi'', \beta \mapsto \langle \tau'; \phi' \rangle$,
    we have the derivation:

$$\dfrac{\hat{\phi} \vDash \phi'' \qquad \dfrac{\hat{\phi} \vDash \phi'}{\hat{\phi} \vDash \langle \tau'; \phi' \rangle} \qquad |\langle \tau'; \phi' \rangle| \in \hat{\phi}(\beta)}{\hat{\phi} \vDash \phi'', \beta \mapsto \langle \tau'; \phi' \rangle}$$

    Either $\alpha \neq \beta$ or $\alpha = \beta$.

· $\alpha \neq \beta$:

From $\alpha \neq \beta$ and $\theta = \theta'', \beta \rightarrowtail \theta'(\tau')$,
we have $\theta(\alpha) = \theta''(\alpha)$.
From $\Delta \vdash \theta(\alpha)$ and $\theta(\alpha) = \theta''(\alpha)$,
we have $\Delta \vdash \theta''(\alpha)$.
By the induction hypothesis
applied to $\langle \phi''; \alpha \rangle \prec \langle \phi'', \beta \mapsto \langle \tau'; \phi' \rangle; \alpha \rangle$
with $\vdash \phi'' : \theta''$, $\hat{\phi} \vDash \phi''$, $\Delta \vdash \theta''(\alpha)$, and $\mathsf{dom}(\phi'') \cap \mathsf{dom}(\Delta) = \emptyset$,
we have $\Delta; \hat{\phi} \vdash \alpha \twoheadrightarrow \theta''(\alpha)$.
From $\Delta; \hat{\phi} \vdash \alpha \twoheadrightarrow \theta''(\alpha)$ and $\theta(\alpha) = \theta''(\alpha)$,
we have $\Delta; \hat{\phi} \vdash \alpha \twoheadrightarrow \theta(\alpha)$.
Thus, $\Delta; \hat{\phi} \vdash \alpha \twoheadrightarrow \theta(\alpha)$.

· $\alpha = \beta$:

From $\alpha = \beta$ and $\theta = \theta'', \beta \rightarrowtail \theta'(\tau')$,
we have $\theta(\alpha) = \theta'(\tau')$.
From $\Delta \vdash \theta(\alpha)$,
we have $\vdash \Delta$.
From $\mathsf{dom}(\phi'', \beta \mapsto \langle \tau'; \phi' \rangle) \cap \mathsf{dom}(\Delta) = \emptyset$ and $\alpha = \beta$,
we have $\alpha \notin \mathsf{dom}(\Delta)$.
From $|\langle \tau'; \phi' \rangle| \in \hat{\phi}(\beta)$, $|\langle \tau'; \phi' \rangle| = \tau'$, and $\alpha = \beta$,
we have $\tau' \in \hat{\phi}(\alpha)$.
By the induction hypothesis
applied to $\langle \phi'; \tau' \rangle \prec \langle \phi'', \beta \mapsto \langle \tau'; \phi' \rangle; \alpha \rangle$
with $\vdash \phi' : \theta'$, $\hat{\phi} \vDash \phi'$, $\bullet \vdash \theta'(\tau')$, and $\mathsf{dom}(\phi') \cap \mathsf{dom}(\bullet) = \emptyset$,
we have $\Delta; \hat{\phi} \vdash \tau' \twoheadrightarrow \theta'(\tau')$.
We can construct the derivation:

$$\frac{\vdash \Delta \qquad \alpha \notin \mathsf{dom}(\Delta) \qquad \tau' \in \hat{\phi}(\alpha) \qquad \bullet; \hat{\phi} \vdash \tau' \twoheadrightarrow \theta'(\tau')}{\Delta; \hat{\phi} \vdash \alpha \twoheadrightarrow \theta'(\tau')}$$

From $\Delta; \hat{\phi} \vdash \alpha \twoheadrightarrow \theta'(\tau')$ and $\theta(\alpha) = \theta'(\tau')$,
we have $\Delta; \hat{\phi} \vdash \alpha \twoheadrightarrow \theta(\alpha)$.
Thus, $\Delta; \hat{\phi} \vdash \alpha \twoheadrightarrow \theta(\alpha)$.

– $\tau_a \to \tau_b$:

From $\Delta \vdash \theta(\tau_a \to \tau_b)$ and $\theta(\tau_a \to \tau_b) = \theta(\tau_a) \to \theta(\tau_b)$,
we have $\Delta \vdash \theta(\tau_a) \to \theta(\tau_b)$.
By inversion of $\Delta \vdash \theta(\tau_a) \to \theta(\tau_b)$,
we have the derivation:

$$\frac{\Delta \vdash \theta(\tau_a) \qquad \Delta \vdash \theta(\tau_b)}{\Delta \vdash \theta(\tau_a) \to \theta(\tau_b)}$$

By the induction hypothesis applied to $\langle \phi; \tau_a \rangle \prec \langle \phi; \tau_a \to \tau_b \rangle$
with $\vdash \phi : \theta$, $\hat{\phi} \vDash \phi$, $\Delta \vdash \theta(\tau_a)$, and $\mathsf{dom}(\phi) \cap \mathsf{dom}(\Delta) = \emptyset$,
we have $\Delta; \hat{\phi} \vdash \tau_a \twoheadrightarrow \theta(\tau_a)$.
By the induction hypothesis applied to $\langle \phi; \tau_b \rangle \prec \langle \phi; \tau_b \to \tau_b \rangle$
with $\vdash \phi : \theta$, $\hat{\phi} \vDash \phi$, $\Delta \vdash \theta(\tau_b)$, and $\mathsf{dom}(\phi) \cap \mathsf{dom}(\Delta) = \emptyset$,
we have $\Delta; \hat{\phi} \vdash \tau_b \twoheadrightarrow \theta(\tau_b)$.
We can construct the derivation:

$$\frac{\Delta; \hat{\phi} \vdash \tau_a \twoheadrightarrow \theta(\tau_a) \qquad \Delta; \hat{\phi} \vdash \tau_b \twoheadrightarrow \theta(\tau_b)}{\Delta; \hat{\phi} \vdash \tau_a \to \tau_b \twoheadrightarrow \theta(\tau_a) \to \theta(\tau_b)}$$

From $\Delta; \hat{\phi} \vdash \tau_a \to \tau_b \twoheadrightarrow \theta(\tau_a) \to \theta(\tau_b)$ and $\theta(\tau_a \to \tau_b) = \theta(\tau_a) \to \theta(\tau_b)$,
we have $\Delta; \hat{\phi} \vdash \tau_a \to \tau_b \twoheadrightarrow \theta(\tau_a \to \tau_b)$.
Thus, $\Delta; \hat{\phi} \vdash \tau_a \to \tau_b \twoheadrightarrow \theta(\tau_a \to \tau_b)$.

– $\forall\alpha.\ \tau_b$:

By "up to α-conversion", we ensure $\alpha \notin \mathsf{dom}(\phi)$.
From $\alpha \notin \mathsf{dom}(\phi)$ and $\vdash \phi : \theta$,
we have $\theta(\forall\alpha.\ \tau_b) = \forall\alpha.\ \theta(\tau_a)$.
From $\Delta \vdash \theta(\forall\alpha.\ \tau_b)$ and $\theta(\forall\alpha.\ \tau_b) = \forall\alpha.\ \theta(\tau_b)$,
we have $\Delta \vdash \forall\alpha.\ \theta(\tau_b)$.
By inversion of $\Delta \vdash \forall\alpha.\ \theta(\tau_b)$,
we have the derivation:

$$\frac{\Delta, \alpha{:}\star \vdash \theta(\tau_b)}{\Delta \vdash \forall\alpha.\ \theta(\tau_b)}$$

By the induction hypothesis applied to $\langle\phi; \tau_b\rangle \prec \langle\phi; \forall\alpha.\ \tau_b\rangle$
with $\vdash \phi : \theta$, $\hat{\phi} \vDash \phi$, $\Delta, \alpha{:}\star \vdash \theta(\tau_b)$, and $\mathsf{dom}(\phi) \cap \mathsf{dom}(\Delta, \alpha{:}\star) = \emptyset$,
we have $\Delta, \alpha{:}\star; \hat{\phi} \vdash \tau_b \twoheadrightarrow \theta(\tau_b)$.
We can construct the derivation:

$$\frac{\Delta, \alpha{:}\star; \hat{\phi} \vdash \tau_b \twoheadrightarrow \theta(\tau_b)}{\Delta; \hat{\phi} \vdash \forall\alpha.\ \tau_b \twoheadrightarrow \forall\alpha.\ \theta(\tau_b)}$$

From $\Delta; \hat{\phi} \vdash \forall\alpha.\ \tau_b \twoheadrightarrow \forall\alpha.\ \theta(\tau_b)$ and $\theta(\forall\alpha.\ \tau_b) = \forall\alpha.\ \theta(\tau_b)$,
we have $\Delta; \hat{\phi} \vdash \forall\alpha.\ \tau_b \twoheadrightarrow \theta(\forall\alpha.\ \tau_b)$.
Thus, $\Delta; \hat{\phi} \vdash \forall\alpha.\ \tau_b \twoheadrightarrow \theta(\forall\alpha.\ \tau_b)$.

Lemma 3 (Syntactic Equality implies Analysis-Time Type Compatibility) *If* $\vdash \phi_1 : \theta_1$, $\hat{\phi} \vDash \phi_1$, $\bullet \vdash \theta_1(\tau_1)$, $\vdash \phi_2 : \theta_2$, $\hat{\phi} \vDash \phi_2$, $\bullet \vdash \theta_2(\tau_2)$, *and* $\theta_1(\tau_1) = \theta_2(\tau_2)$, *then* $\hat{\phi} \vdash \tau_1 \approx\!\!\!\approx \tau_2$.

Proof.
By Lemma 19 with $\vdash \phi_1 : \theta_1$, $\hat{\phi} \vDash \phi_1$, $\bullet \vdash \theta_1(\tau_1)$, and $\mathsf{dom}(\phi_1) \cap \mathsf{dom}(\bullet) = \emptyset$, we have $\bullet; \hat{\phi} \vdash \tau_1 \twoheadrightarrow \theta_1(\tau_1)$.
By Lemma 19 with $\vdash \phi_2 : \theta_2$, $\hat{\phi} \vDash \phi_2$, $\bullet \vdash \theta_2(\tau_2)$, and $\mathsf{dom}(\phi_2) \cap \mathsf{dom}(\bullet) = \emptyset$, we have $\bullet; \hat{\phi} \vdash \tau_2 \twoheadrightarrow \theta_2(\tau_2)$.
We can construct the derivation:

$$\frac{\bullet; \hat{\phi} \vdash \tau_1 \twoheadrightarrow \theta_1(\tau_1) \qquad \bullet; \hat{\phi} \vdash \tau_2 \twoheadrightarrow \theta_2(\tau_2) \qquad \theta_1(\tau_1) = \theta_2(\tau_2)}{\hat{\phi} \vdash \tau_1 \approx\!\!\!\approx \tau_2}$$

Thus, $\hat{\phi} \vdash \tau_1 \approx\!\!\!\approx \tau_2$.

Lemma 20
If $\vdash \phi : \theta$, $\hat{\phi} \vDash \phi$, $\vdash w : \theta(\tau)$, *and* $\hat{\phi}; \hat{\rho} \vDash w$,
then $\hat{\phi} \vdash |w| :\approx \tau$.

Proof.
Proceed by cases on the structure of w.

– $\langle \mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b; \phi_f; \rho_f \rangle$:

By inversion of $\hat{\phi}; \hat{\rho} \vDash \langle \mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b; \phi_f; \rho_f \rangle$,
we have the derivation:

$$\frac{\{\mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b\} \in \hat{\rho}(f) \qquad \hat{\phi}; \hat{\rho} \vDash e_b \qquad \hat{\phi} \vDash \phi_f \qquad \hat{\phi}; \hat{\rho} \vDash \rho_f}{\hat{\phi}; \hat{\rho} \vDash \langle \mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b; \phi_f; \rho_f \rangle}$$

From $\vdash \langle \mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b; \phi_f; \rho_f \rangle : \theta(\tau)$,
we have $\bullet \vdash \theta(\tau)$.
By inversion of $\vdash \langle \mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b; \phi_f; \rho_f \rangle : \theta(\tau)$,
we have the derivation:

$$\frac{\vdash \phi_f : \theta_f \qquad \vdash \rho_f : \Gamma_f \qquad \bullet; \Gamma_f \vdash \theta_f(\mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b) : \theta(\tau)}{\vdash \langle \mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b; \phi_f; \rho_f \rangle : \theta(\tau)}$$

From $\bullet; \Gamma_f \vdash \theta_f(\mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b) : \theta(\tau)$
and $\theta_f(\mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b) = \mu f : \theta_f(\tau_f) \,.\, \lambda z : \theta_f(\tau_z) \,.\, \theta_f(e_b)$,
we have $\bullet; \Gamma_f \vdash \mu f : \theta_f(\tau_f) \,.\, \lambda z : \theta_f(\tau_z) \,.\, \theta_f(e_b) : \theta(\tau)$.
By inversion of $\bullet; \Gamma_f \vdash \mu f : \theta_f(\tau_f) \,.\, \lambda z : \theta_f(\tau_z) \,.\, \theta_f(e_b) : \theta(\tau)$,
we have the derivation:

$$\frac{\begin{array}{c}\bullet \vdash \Gamma_f \qquad \bullet \vdash \theta_f(\tau_f) \qquad \bullet \vdash \theta_f(\tau_z) \\ \bullet; \Gamma_f, f{:}\theta_f(\tau_f), z{:}\theta_z(\tau_z) \vdash \theta_f(e_b) : \tau_b \\ \theta_f(\tau_f) = \tau_a \to \tau_b \qquad \theta_f(\tau_x) = \tau_a\end{array}}{\bullet; \Gamma_f \vdash \mu f : \theta_f(\tau_f) \,.\, \lambda z : \theta_f(\tau_z) \,.\, \theta_f(e_b) : \tau_a \to \tau_b}$$

and $\tau_a \to \tau_b = \theta(\tau)$.
By Lemma 3 with $\vdash \phi_f : \theta_f$, $\hat{\phi} \vDash \phi_f$, $\bullet \vdash \theta_f(\tau_f)$,
$\vdash \phi : \theta$, $\hat{\phi} \vDash \phi$, $\bullet \vdash \theta(\tau)$, and $\theta_f(\tau_f) = \tau_a \to \tau_b = \theta(\tau)$,
we have $\hat{\phi} \vdash \tau_f \approx \tau$.
We can construct the derivation:

$$\frac{\hat{\phi} \vdash \tau_f \approx \tau}{\hat{\phi} \vdash \mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b :\approx \tau}$$

From $\hat{\phi} \vdash \mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b :\approx \tau$
and $|\langle \mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b; \phi_f; \rho_f \rangle| = \mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b$,
we have $\hat{\phi} \vdash |\langle \mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b; \phi_f; \rho_f \rangle| :\approx \tau$.
Thus, $\hat{\phi} \vdash |\langle \mu f : \tau_f \,.\, \lambda z : \tau_z \,.\, e_b; \phi_f; \rho_f \rangle| :\approx \tau$.

– $\langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi_f; \rho_f \rangle$:

By inversion of $\hat{\phi}; \hat{\rho} \vDash \langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi_f; \rho_f \rangle$,
we have the derivation:

$$\frac{\{\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\} \in \hat{\rho}(f) \qquad \hat{\phi}; \hat{\rho} \vDash e_b \qquad \hat{\phi} \vDash \phi_f \qquad \hat{\phi}; \hat{\rho} \vDash \rho_f}{\hat{\phi}; \hat{\rho} \vDash \langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi_f; \rho_f \rangle}$$

From $\vdash \langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi_f; \rho_f \rangle : \theta(\tau)$,
we have $\bullet \vdash \theta(\tau)$.
By inversion of $\vdash \langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi_f; \rho_f \rangle : \theta(\tau)$,
we have the derivation:

$$\frac{\vdash \phi_f : \theta_f \qquad \vdash \rho_f : \Gamma_f \qquad \bullet; \Gamma_f \vdash \theta_f(\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b) : \theta(\tau)}{\vdash \langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi_f; \rho_f \rangle : \theta(\tau)}$$

From $\bullet; \Gamma_f \vdash \theta_f(\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b) : \theta(\tau)$
and $\theta_f(\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b) = \mu f\!:\!\theta_f(\tau_f)\,.\,\Lambda\beta\,.\,\theta'_f(e_b)$,
where $\theta'_f = \theta_f|_{\mathsf{dom}(\theta_f)\setminus\{\beta\}}$,
we have $\bullet; \Gamma_f \vdash \mu f\!:\!\theta_f(\tau_f)\,.\,\Lambda\beta\,.\,\theta'_f(e_b) : \theta(\tau)$.
By inversion of $\bullet; \Gamma_f \vdash \mu f\!:\!\theta_f(\tau_f)\,.\,\Lambda\beta\,.\,\theta'_f(e_b) : \theta(\tau)$,
we have the derivation:

$$\frac{\bullet \vdash \Gamma_f \qquad \bullet \vdash \theta_f(\tau_f) \qquad \bullet, \beta\!:\!\star; \Gamma_f, f\!:\!\theta_f(\tau_f) \vdash \theta'_f(e_b) : \tau_b \qquad \theta_f(\tau_f) = \forall\beta.\ \tau_b}{\bullet; \Gamma_f \vdash \mu f\!:\!\theta(\tau_f)\,.\,\Lambda\beta\,.\,\theta_f(e_b) : \forall\beta.\ \tau_b}$$

and $\forall\beta.\ \tau_b = \theta(\tau)$.
By Lemma 3 with $\vdash \phi_f : \theta_f$, $\hat{\phi} \vDash \phi_f$, $\bullet \vdash \theta_f(\tau_f)$,
$\vdash \phi : \theta$, $\hat{\phi} \vDash \phi$, $\bullet \vdash \theta(\tau)$, and $\theta_f(\tau_f) = \forall\beta.\ \tau_b = \theta(\tau)$,
we have $\hat{\phi} \vdash \tau_f \approx\!\!\gg \tau$.
We can construct the derivation:

$$\frac{\hat{\phi} \vdash \tau_f \approx\!\!\gg \tau}{\hat{\phi} \vdash \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b :\!\approx\!\!\gg \tau}$$

From $\hat{\phi} \vdash \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b :\!\approx\!\!\gg \tau$
and $|\langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi_f; \rho_f \rangle| = \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b$,
we have $\hat{\phi} \vdash |\langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi_f; \rho_f \rangle| :\!\approx\!\!\gg \tau$.
Thus, $\hat{\phi} \vdash |\langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi_f; \rho_f \rangle| :\!\approx\!\!\gg \tau$.

Theorem 21 (Preservation (Flow Soundness)).
If $\vdash \varsigma : \tau$, $\hat{\phi}; \hat{\rho} \vDash \varsigma$, *and* $\varsigma \longrightarrow \varsigma'$, *then* $\hat{\phi}; \hat{\rho} \vDash \varsigma'$.

Proof.
Proceed by cases on the derivation of $\varsigma \longrightarrow \varsigma'$.

– $\dfrac{w_r = \rho_r(x_r)}{\langle x_r; \phi_r; \rho_r; \langle z; \tau_z; e; \phi; \rho\rangle{::}\kappa\rangle \longrightarrow \langle e; \phi; \rho, z \mapsto w_r; \kappa\rangle}$:

By inversion of $\hat{\phi}; \hat{\rho} \vDash \langle x_r; \phi_r; \rho_r; \langle z; \tau_z; e; \phi; \rho\rangle{::}\kappa\rangle$,
we have the derivation:

$$\dfrac{\dfrac{\dfrac{}{\hat{\phi}; \hat{\rho} \vDash x_r} \qquad \hat{\phi} \vDash \phi_r \qquad \hat{\phi}; \hat{\rho} \vDash \rho_r}{\{\hat{w}_r \in \hat{\rho}(\lfloor x_r \rfloor) \mid \vdash \hat{\phi} \Rrightarrow \hat{w}_r :\approx\!\!> \tau_z\} \subseteq \hat{\rho}(z)} \qquad \hat{\phi}; \hat{\rho} \vDash e \qquad \hat{\phi} \vDash \phi \qquad \hat{\phi}; \hat{\rho} \vDash \rho \qquad \hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor e \rfloor) \rightsquigarrow \kappa}{\hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor x_r \rfloor) \rightsquigarrow \langle z; \tau_z; e; \phi; \rho\rangle{::}\kappa}}{\hat{\phi}; \hat{\rho} \vDash \langle x_r; \phi_r; \rho_r; \langle z; \tau_z; e; \phi; \rho\rangle{::}\kappa\rangle}$$

By Lemma 18 with $\hat{\phi}; \hat{\rho} \vDash \rho_r$ and $\rho(x_r) = w_r$,
we have $\hat{\phi}; \hat{\rho} \vDash w_r$ and $|w_r| \in \hat{\rho}(x_r)$.
By inversion of $\vdash \langle x_r; \phi_r; \rho_r; \langle z; \tau_z; e; \phi; \rho\rangle{::}\kappa\rangle : \tau$,
we have the derivation:

$$\dfrac{\dfrac{\vdash \phi_r : \theta_r \qquad \vdash \rho_r : \Gamma_r}{\bullet; \Gamma_r \vdash \theta_r(x_r) : \tau_r} \qquad \dfrac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet \vdash \theta(\tau_z) \qquad \theta(\tau_z) = \tau_r \qquad \bullet; \Gamma, z{:}\theta(\tau_z) \vdash \theta(e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau}{\vdash \tau_r \rightsquigarrow \langle z; \tau_z; e; \phi; \rho\rangle{::}\kappa : \tau}}{\vdash \langle x_r; \phi_r; \rho_r; \langle z; \tau_z; e; \phi; \rho\rangle{::}\kappa\rangle : \tau}$$

From $\bullet; \Gamma_r \vdash \theta_r(x_r) : \tau_r$ and $\theta_r(x_r) = x_r$,
we have $\bullet; \Gamma_r \vdash x_r : \tau_r$.
By inversion of $\bullet; \Gamma_r \vdash x_r : \tau_r$, we have the derivation:

$$\dfrac{\bullet \vdash \Gamma_r \qquad \Gamma_r(x_r) = \tau_r}{\bullet; \Gamma_r \vdash x_r : \tau_r}$$

By Lemma 7 with $\vdash \rho_r : \Gamma_r$, $\Gamma_r(x_r) = \tau_r$, and $\rho_r(x_r) = w_r$
we have $\vdash w_r : \tau_r$.
From $\vdash w_r : \tau_r$ and $\theta(\tau_z) = \tau_r$,
we have $\vdash w_r : \theta(\tau_z)$.
By Lemma 20 with $\vdash \phi : \theta$, $\hat{\phi} \vDash \phi$, $\vdash w_r : \theta(\tau_z)$, and $\hat{\phi}; \hat{\rho} \vDash w_r$,
we have $\hat{\phi} \vdash |w_r| :\approx\!\!> \tau_z$.
From $\{\hat{w}_r \in \hat{\rho}(\lfloor x_r \rfloor) \mid \hat{\phi} \vdash \hat{w}_r :\approx\!\!> \tau_z\} \in \hat{\rho}(z)$ and $\lfloor x_r \rfloor = x_r$,
we have $\{\hat{w}_r \in \hat{\rho}(x_r) \mid \hat{\phi} \vdash \hat{w}_r :\approx\!\!> \tau_z\} \in \hat{\rho}(z)$.

From $\{\hat{w}_r \in \hat{\rho}(x) \mid \hat{\phi} \vdash \hat{w}_r :\approx\!\!\!\!\approx \tau_z\} \in \hat{\rho}(z)$, $|w_r| \in \hat{\rho}(x_r)$,
and $\hat{\phi} \vdash |w_r| :\approx\!\!\!\!\approx \tau_z$,
we have $|w_r| \in \hat{\rho}(z)$.
We can construct the derivation:

$$\dfrac{\hat{\phi};\hat{\rho} \vDash e \qquad \hat{\phi} \vDash \phi \qquad \dfrac{\hat{\phi};\hat{\rho} \vDash \rho \qquad \hat{\phi};\hat{\rho} \vDash w_r \qquad |w_r| \in \hat{\rho}(z)}{\hat{\phi};\hat{\rho} \vDash \rho, z \mapsto w_r} \qquad \dfrac{\hat{\phi};\hat{\rho} \vDash \hat{\rho}(\lfloor e \rfloor) \rightsquigarrow \kappa}{\hat{\phi};\hat{\rho} \vDash \langle e;\phi;\rho, z \mapsto w_r;\kappa\rangle}$$

Thus, $\hat{\phi};\hat{\rho} \vDash \langle e;\phi;\rho, z \mapsto w_r;\kappa\rangle$.

– $\dfrac{w_r = \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi; \rho\rangle}{\langle \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\ \texttt{in}\ e; \phi; \rho; \kappa\rangle \longrightarrow \langle e; \phi; \rho, x \mapsto w_r; \kappa\rangle}$:

By inversion of $\hat{\phi}; \hat{\rho} \vDash \langle \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\ \texttt{in}\ e\ \texttt{in}\ e; \phi; \rho; \kappa\rangle$,
we have the derivation:

$$\frac{\hat{\phi}; \hat{\rho} \vDash \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b \rightsquigarrow \hat{W} \qquad \{\hat{w} \in \hat{W} \mid \hat{\phi} \vdash \hat{w} :\approx\!\approx \tau_x\} \subseteq \hat{\rho}(x) \qquad \hat{\phi}; \hat{\rho} \vDash e}{\hat{\phi}; \hat{\rho} \vDash \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\ \texttt{in}\ e}$$

$$\frac{\hat{\phi} \vDash \phi \qquad \hat{\phi}; \hat{\rho} \vDash \rho \qquad \hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\ \texttt{in}\ e\rfloor) \rightsquigarrow \kappa}{\hat{\phi}; \hat{\rho} \vDash \langle \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\ \texttt{in}\ e; \phi; \rho; \kappa\rangle}$$

By inversion of $\hat{\phi}; \hat{\rho} \vDash \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b \rightsquigarrow \hat{W}$,
we have the derivation:

$$\frac{\{\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\} \subseteq \hat{\rho}(f) \qquad \hat{\phi}; \hat{\rho} \vDash e_b \qquad \{\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\} \subseteq \hat{W}}{\hat{\phi}; \hat{\rho} \vDash \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b \rightsquigarrow \hat{W}}$$

From $\{\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\} \subseteq \hat{W}$, $|\langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi; \rho\rangle| = \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b$,
and $w_r = \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi; \rho\rangle$,
we have $\{|w_r|\} \subseteq \hat{W}$.
We can construct the derivation:

$$\frac{\{\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\} \in \hat{\rho}(f) \qquad \hat{\phi}; \hat{\rho} \vDash e_b \qquad \hat{\phi} \vDash \phi \qquad \hat{\phi}; \hat{\rho} \vDash \rho}{\hat{\phi}; \hat{\rho} \vDash \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi; \rho\rangle}$$

From $\hat{\phi}; \hat{\rho} \vDash \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi; \rho\rangle$ and $w_r = \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi; \rho\rangle$,
we have $\hat{\phi}; \hat{\rho} \vDash w_f$.
From $\hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\ \texttt{in}\ e\rfloor) \rightsquigarrow \kappa$
and $\lfloor \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\ \texttt{in}\ e\rfloor = \lfloor e\rfloor$,
we have $\hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor e\rfloor) \rightsquigarrow \kappa$.
By inversion of $\vdash \langle \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\ \texttt{in}\ e; \phi; \rho; \kappa\rangle : \tau$,
we have the derivation:

$$\frac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet; \Gamma \vdash \theta(\texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\ \texttt{in}\ e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau}{\vdash \langle \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\ \texttt{in}\ e; \phi; \rho; \kappa\rangle : \tau}$$

From $\bullet; \Gamma \vdash \theta(\texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\ \texttt{in}\ e) : \tau_e$
and $\theta(\texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\ \texttt{in}\ e) = \texttt{let}\ x\!:\!\theta(\tau_x)\ \texttt{=}\ \theta(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b)\ \texttt{in}\ \theta(e)$,
we have $\bullet; \Gamma \vdash \texttt{let}\ x\!:\!\theta(\tau_x)\ \texttt{=}\ \theta(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b)\ \texttt{in}\ \theta(e) : \tau_e$.
By inversion of $\bullet; \Gamma \vdash \texttt{let}\ x\!:\!\theta(\tau_x)\ \texttt{=}\ \theta(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b)\ \texttt{in}\ \theta(e) : \tau_e$,

we have the derivation:

$$\dfrac{\begin{array}{c}\bullet; \Gamma \vdash \theta(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b) : \tau_s \\ \bullet \vdash \theta(\tau_x) \qquad \theta(\tau_x) = \tau_s \qquad \bullet; \Gamma, x{:}\theta(\tau_x) \vdash \theta(e) : \tau_e\end{array}}{\bullet; \Gamma \vdash \mathtt{let}\ x{:}\theta(\tau_x) = \theta(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b)\ \mathtt{in}\ \theta(e) : \tau_e}$$

From $\bullet; \Gamma \vdash \theta(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b) : \tau_s$ and $\theta(\tau_x) = \tau_s$,
we have $\bullet; \Gamma \vdash \theta(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b) : \theta(\tau_x)$.
We can construct the derivation:

$$\dfrac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet; \Gamma \vdash \theta(\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b) : \theta(\tau_x)}{\vdash \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi; \rho\rangle : \theta(\tau_x)}$$

From $\vdash \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi; \rho\rangle : \theta(\tau_x)$ and $w_r = \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi; \rho\rangle$,
we have $\vdash w_r : \theta(\tau_x)$.
From Lemma 20 with $\vdash \phi : \theta$, $\hat{\phi} \vDash \phi$, $\vdash w_r : \theta(\tau_x)$, and $\hat{\phi}; \hat{\rho} \vDash w_r$,
we have $\hat{\phi} \vdash |w_r| :\!\approx\!\!\approx \tau_x$.
From $\{\hat{w} \in \hat{W} \mid \hat{\phi} \vdash \hat{w} :\!\approx\!\!\approx \tau_x\} \subseteq \hat{\rho}(x)$, $\{|w_r|\} \subseteq \hat{W}$, and $\hat{\phi} \vdash |w_r| :\!\approx\!\!\approx \tau_x$,
we have $\{|w_r|\} \subseteq \hat{\rho}(x)$.
We can construct the derivation:

$$\dfrac{\hat{\phi}; \hat{\rho} \vDash e \qquad \hat{\phi} \vDash \phi \qquad \dfrac{\hat{\phi}; \hat{\rho} \vDash \rho \qquad \hat{\phi}; \hat{\rho} \vDash w_r \qquad \{|w_r|\} \in \hat{\rho}(x)}{\hat{\phi}; \hat{\rho} \vDash \rho, x \mapsto w_r} \qquad \hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor e \rfloor) \rightsquigarrow \kappa}{\hat{\phi}; \hat{\rho} \vDash \langle e; \phi; \rho, x \mapsto w_r; \kappa\rangle}$$

Thus, $\hat{\phi}; \hat{\rho} \vDash \langle e; \phi; \rho, x \mapsto w_r; \kappa\rangle$.

– $\dfrac{w_r = \langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi; \rho\rangle}{\langle \texttt{let}\ x\!:\!\tau_x = \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e; \phi; \rho; \kappa\rangle \longrightarrow \langle e; \phi; \rho, x \mapsto w_r; \kappa\rangle}$:

By inversion of $\hat{\phi}; \hat{\rho} \vDash \langle \texttt{let}\ x\!:\!\tau_x = \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e\ \texttt{in}\ e; \phi; \rho; \kappa\rangle$,
we have the derivation:

$$\dfrac{\hat{\phi}; \hat{\rho} \vDash \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b \rightsquigarrow \hat{W} \qquad \{\hat{w} \in \hat{W} \mid \hat{\phi} \vdash \hat{w} :\approx\!\!\approx \tau_x\} \subseteq \hat{\rho}(x) \qquad \hat{\phi}; \hat{\rho} \vDash e}{\hat{\phi}; \hat{\rho} \vDash \texttt{let}\ x\!:\!\tau_x = \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e}$$

$$\dfrac{\hat{\phi} \vDash \phi \qquad \hat{\phi}; \hat{\rho} \vDash \rho \qquad \hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor \texttt{let}\ x\!:\!\tau_x = \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e\rfloor) \rightsquigarrow \kappa}{\hat{\phi}; \hat{\rho} \vDash \langle \texttt{let}\ x\!:\!\tau_x = \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e; \phi; \rho; \kappa\rangle}$$

By inversion of $\hat{\phi}; \hat{\rho} \vDash \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b \rightsquigarrow \hat{W}$,
we have the derivation:

$$\dfrac{\{\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\} \subseteq \hat{\rho}(f) \qquad \hat{\phi}; \hat{\rho} \vDash e_b \qquad \{\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\} \subseteq \hat{W}}{\hat{\phi}; \hat{\rho} \vDash \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b \rightsquigarrow \hat{W}}$$

From $\{\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\} \subseteq \hat{W}$, $|\langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi; \rho\rangle| = \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b$,
and $w_r = \langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi; \rho\rangle$,
we have $\{|w_r|\} \subseteq \hat{W}$.
We can construct the derivation:

$$\dfrac{\{\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\} \in \hat{\rho}(f) \qquad \hat{\phi}; \hat{\rho} \vDash e_b \qquad \hat{\phi} \vDash \phi \qquad \hat{\phi}; \hat{\rho} \vDash \rho}{\hat{\phi}; \hat{\rho} \vDash \langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi; \rho\rangle}$$

From $\hat{\phi}; \hat{\rho} \vDash \langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi; \rho\rangle$ and $w_r = \langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi; \rho\rangle$,
we have $\hat{\phi}; \hat{\rho} \vDash w_f$.
From $\hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor \texttt{let}\ x\!:\!\tau_x = \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e\rfloor) \rightsquigarrow \kappa$
and $\lfloor \texttt{let}\ x\!:\!\tau_x = \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e\rfloor = \lfloor e\rfloor$,
we have $\hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor e\rfloor) \rightsquigarrow \kappa$.
By inversion of $\vdash \langle \texttt{let}\ x\!:\!\tau_x = \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e; \phi; \rho; \kappa\rangle : \tau$,
we have the derivation:

$$\dfrac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet; \Gamma \vdash \theta(\texttt{let}\ x\!:\!\tau_x = \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau}{\vdash \langle \texttt{let}\ x\!:\!\tau_x = \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e; \phi; \rho; \kappa\rangle : \tau}$$

From $\bullet; \Gamma \vdash \theta(\texttt{let}\ x\!:\!\tau_x = \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e) : \tau_e$
and $\theta(\texttt{let}\ x\!:\!\tau_x = \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\ \texttt{in}\ e) = \texttt{let}\ x\!:\!\theta(\tau_x) = \theta(\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b)\ \texttt{in}\ \theta(e)$,
we have $\bullet; \Gamma \vdash \texttt{let}\ x\!:\!\theta(\tau_x) = \theta(\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b)\ \texttt{in}\ \theta(e) : \tau_e$.
By inversion of $\bullet; \Gamma \vdash \texttt{let}\ x\!:\!\theta(\tau_x) = \theta(\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b)\ \texttt{in}\ \theta(e) : \tau_e$,

we have the derivation:

$$\dfrac{\begin{array}{c}\bullet;\Gamma \vdash \theta(\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b) : \tau_s \\ \bullet \vdash \theta(\tau_x) \qquad \theta(\tau_x) = \tau_s \qquad \bullet;\Gamma, x{:}\theta(\tau_x) \vdash \theta(e) : \tau_e\end{array}}{\bullet;\Gamma \vdash \texttt{let}\ x{:}\theta(\tau_x)\ \texttt{=}\ \theta(\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b)\ \texttt{in}\ \theta(e) : \tau_e}$$

From $\bullet;\Gamma \vdash \theta(\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b) : \tau_s$ and $\theta(\tau_x) = \tau_s$,
we have $\bullet;\Gamma \vdash \theta(\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b) : \theta(\tau_x)$.
We can construct the derivation:

$$\dfrac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet;\Gamma \vdash \theta(\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b) : \theta(\tau_x)}{\vdash \langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi; \rho\rangle : \theta(\tau_x)}$$

From $\vdash \langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi; \rho\rangle : \theta(\tau_x)$ and $w_r = \langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi; \rho\rangle$,
we have $\vdash w_r : \theta(\tau_x)$.
From Lemma 20 with $\vdash \phi : \theta$, $\hat{\phi} \vDash \phi$, $\vdash w_r : \theta(\tau_x)$, and $\hat{\phi}; \hat{\rho} \vDash w_r$,
we have $\hat{\phi} \vdash |w_r| :\!\approx\!\!\approx \tau_x$.
From $\{\hat{w} \in \hat{W} \mid \hat{\phi} \vdash \hat{w} :\!\approx\!\!\approx \tau_x\} \subseteq \hat{\rho}(x)$, $\{|w_r|\} \subseteq \hat{W}$, and $\hat{\phi} \vdash |w_r| :\!\approx\!\!\approx \tau_x$,
we have $\{|w_r|\} \subseteq \hat{\rho}(x)$.
We can construct the derivation:

$$\dfrac{\hat{\phi};\hat{\rho} \vDash e \qquad \hat{\phi} \vDash \phi \qquad \dfrac{\hat{\phi};\hat{\rho} \vDash \rho \qquad \hat{\phi};\hat{\rho} \vDash w_r \qquad \{|w_r|\} \in \hat{\rho}(x)}{\hat{\phi};\hat{\rho} \vDash \rho, x \mapsto w_r} \qquad \hat{\phi};\hat{\rho} \vDash \hat{\rho}(\lfloor e \rfloor) \rightsquigarrow \kappa}{\hat{\phi};\hat{\rho} \vDash \langle e; \phi; \rho, x \mapsto w_r; \kappa\rangle}$$

Thus, $\hat{\phi};\hat{\rho} \vDash \langle e; \phi; \rho, x \mapsto w_r; \kappa\rangle$.

– $\dfrac{\rho(x_f) = w_f \qquad w_f = \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi_f; \rho_f\rangle \qquad \rho(x_a) = w_a}{\langle \texttt{let}\ x\!:\!\tau_x = x_f\ x_a\ \texttt{in}\ e; \phi; \rho; \kappa\rangle \longrightarrow \langle e_b; \phi_f; \rho_f, f \mapsto w_f, z \mapsto w_a; \langle x; \tau_x; e; \phi; \rho\rangle{::}\kappa\rangle}$:

By inversion of $\hat{\phi}; \hat{\rho} \vDash \langle \texttt{let}\ x\!:\!\tau_x = x_f\ x_a\ \texttt{in}\ e; \phi; \rho; \kappa\rangle$,
we have the derivation

$$\frac{\hat{\phi}; \hat{\rho} \vDash x_f\ x_a \rightsquigarrow \hat{W} \qquad \{\hat{w} \in \hat{W} \mid \hat{\phi} \vdash \hat{w} :\approx \tau_x\} \subseteq \hat{\rho}(x) \qquad \hat{\phi}; \hat{\rho} \vDash e}{\hat{\phi}; \hat{\rho} \vDash \texttt{let}\ x\!:\!\tau_x = x_f\ x_a\ \texttt{in}\ e}$$

$$\frac{\hat{\phi} \vDash \phi \qquad \hat{\phi}; \hat{\rho} \vDash \rho \qquad \hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor \texttt{let}\ x\!:\!\tau_x = x_f\ x_a\ \texttt{in}\ e \rfloor) \rightsquigarrow \kappa}{\hat{\phi}; \hat{\rho} \vDash \langle \texttt{let}\ x\!:\!\tau_x = x_f\ x_a\ \texttt{in}\ e; \phi; \rho; \kappa\rangle}$$

By inversion of $\hat{\phi}; \hat{\rho} \vDash x_f\ x_a \rightsquigarrow \hat{W}$, we have the derivation:

$$\frac{\Psi}{\hat{\phi}; \hat{\rho} \vDash x_f\ x_a \rightsquigarrow \hat{W}}$$

where

$$\Psi = \bigwedge_{\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b \in \hat{\rho}(x_f)} \left(\{\hat{w}_a \in \hat{\rho}(x_a) \mid \hat{\phi} \vdash \hat{w}_a :\approx \tau_z\} \subseteq \hat{\rho}(z) \wedge \hat{\rho}(\lfloor e_b \rfloor) \subseteq \hat{W} \right)$$

By Lemma 18 with $\hat{\phi}; \hat{\rho} \vDash \rho$ and $\rho(x_f) = w_f$,
we have $\hat{\phi}; \hat{\rho} \vDash w_f$ and $\{|w_f|\} \subseteq \hat{\rho}(x)$.
From $\{|w_f|\} \subseteq \hat{\rho}(x_f)$
and $|w_f| = |\langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi_f; \rho_f\rangle| = \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b$,
we have $\{\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\} \subseteq \hat{\rho}(x_f)$.
From Ψ and $\{\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\} \subseteq \hat{\rho}(x_f)$,
we have $\{\hat{w}_a \in \hat{\rho}(x_a) \mid \hat{\phi} \vdash \hat{w}_a :\approx \tau_z\} \subseteq \hat{\rho}(z)$ and $\hat{\rho}(\lfloor e_b \rfloor) \subseteq \hat{W}$.
From $\{\hat{w} \in \hat{W} \mid \hat{\phi} \vdash \hat{w} :\approx \tau_x\} \subseteq \hat{\rho}(x)$ and $\hat{\rho}(\lfloor e_b \rfloor) \subseteq \hat{W}$,
we have $\{\hat{w} \in \hat{\rho}(\lfloor e_b \rfloor) \mid \hat{\phi} \vdash \hat{w} :\approx \tau_x\} \subseteq \hat{\rho}(x)$.
From $\hat{\phi}; \hat{\rho} \vDash w_f$ and $w_f = \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi_f; \rho_f\rangle$,
we have $\hat{\phi}; \hat{\rho} \vDash \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi_f; \rho_f\rangle$.
By inversion of $\hat{\phi}; \hat{\rho} \vDash \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi_f; \rho_f\rangle$,
we have the derivation:

$$\frac{\dfrac{\{\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\} \subseteq \hat{\rho}(f) \qquad \hat{\phi}; \hat{\rho} \vDash e_b}{\hat{\phi}; \hat{\rho} \vDash \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b} \qquad \hat{\phi} \vDash \phi_f \qquad \hat{\phi}; \hat{\rho} \vDash \rho_f}{\hat{\phi}; \hat{\rho} \vDash \langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi_f; \rho_f\rangle}$$

From $\hat{\rho}(f) \supseteq \{\mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b\}$
and $|w_f| = |\langle \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b; \phi_f; \rho_f\rangle| = \mu f\!:\!\tau_f\,.\,\lambda z\!:\!\tau_z\,.\,e_b$,

we have $\{|w_f|\} \subseteq \hat{\rho}(f)$.
By Lemma 18 with $\hat{\phi}; \hat{\rho} \vDash \rho$ and $\rho(x_a) = w_a$,
we have $\hat{\phi}; \hat{\rho} \vDash w_a$ and $\{|w_a|\} \subseteq \hat{\rho}(x)$.
By inversion of $\vdash \langle \mathtt{let}\ x{:}\tau_x\ \mathtt{=}\ x_f\ x_a\ \mathtt{in}\ e; \phi; \rho; \kappa\rangle : \tau$,
we have the derivation:

$$\frac{\vdash \phi : \theta \qquad \vdash \rho : \Gamma \qquad \bullet; \Gamma \vdash \theta(\mathtt{let}\ x{:}\tau_x\ \mathtt{=}\ x_f\ x_a\ \mathtt{in}\ e) : \tau_e \qquad \vdash \tau_e \rightsquigarrow \kappa : \tau}{\vdash \langle \mathtt{let}\ x{:}\tau_x\ \mathtt{=}\ x_f\ x_a\ \mathtt{in}\ e; \phi; \rho; \kappa\rangle : \tau}$$

From $\bullet; \Gamma \vdash \theta(\mathtt{let}\ x{:}\tau_x\ \mathtt{=}\ x_f\ x_a\ \mathtt{in}\ e) : \tau_e$
and $\theta(\mathtt{let}\ x{:}\tau_x\ \mathtt{=}\ x_f\ x_a\ \mathtt{in}\ e) = \mathtt{let}\ x{:}\theta(\tau_x)\ \mathtt{=}\ x_f\ x_a\ \mathtt{in}\ \theta(e)$,
we have $\bullet; \Gamma \vdash \mathtt{let}\ x{:}\theta(\tau_x)\ \mathtt{=}\ x_f\ x_a\ \mathtt{in}\ \theta(e) : \tau_e$.
By inversion of $\bullet; \Gamma \vdash \mathtt{let}\ x{:}\theta(\tau_x)\ \mathtt{=}\ x_f\ x_a\ \mathtt{in}\ \theta(e) : \tau_e$,
we have the derivation:

$$\frac{\bullet \vdash \Gamma \qquad \Gamma(x_f) = \tau_a \to \tau_b \qquad \Gamma(x_a) = \tau_a}{\bullet; \Gamma \vdash x_f\ x_a : \tau_b}$$

$$\frac{\bullet \vdash \theta(\tau_x) \qquad \theta(\tau_x) = \tau_b \qquad \bullet; \Gamma, x{:}\theta(\tau_x) \vdash \theta(e) : \tau_e}{\bullet; \Gamma \vdash \mathtt{let}\ x{:}\theta(\tau_x)\ \mathtt{=}\ x_f\ x_a\ \mathtt{in}\ \theta(e) : \tau_e}$$

By Lemma 7 with $\vdash \rho : \Gamma$, $\Gamma(x_f) = \tau_a \to \tau_b$, $\rho(x_f) = w_f$,
we have $\vdash w_f : \tau_a \to \tau_b$.
From $\vdash w_f : \tau_a \to \tau_b$ and $w_f = \langle \mu f{:}\tau_f\,.\,\lambda z{:}\tau_z\,.\,e_b; \phi_f; \rho_f\rangle$,
we have $\vdash \langle \mu f{:}\tau_f\,.\,\lambda z{:}\tau_z\,.\,e_b; \phi_f; \rho_f\rangle : \tau_a \to \tau_b$.
By inversion of $\vdash \langle \mu f{:}\tau_f\,.\,\lambda z{:}\tau_z\,.\,e_b; \phi_f; \rho_f\rangle : \tau_a \to \tau_b$,
we have the derivation:

$$\frac{\vdash \phi_f : \theta_f \qquad \vdash \rho_f : \Gamma_f \qquad \bullet; \Gamma_f \vdash \theta_f(\mu f{:}\tau_f\,.\,\lambda z{:}\tau_z\,.\,e_b) : \tau_a \to \tau_b}{\vdash \langle \mu f{:}\tau_f\,.\,\lambda z{:}\tau_z\,.\,e_b; \phi_f; \rho_f\rangle : \tau_a \to \tau_b}$$

From $\bullet; \Gamma_f \vdash \theta_f(\mu f{:}\tau_f\,.\,\lambda z{:}\tau_z\,.\,e_b) : \tau_a \to \tau_b$
and $\theta_f(\mu f{:}\tau_f\,.\,\lambda z{:}\tau_z\,.\,e_b) = \mu f{:}\theta_f(\tau_f)\,.\,\lambda z{:}\theta_f(\tau_z)\,.\,\theta_f(e_b)$,
we have $\bullet; \Gamma_f \vdash \mu f{:}\theta_f(\tau_f)\,.\,\lambda z{:}\theta_f(\tau_z)\,.\,\theta_f(e_b) : \tau_a \to \tau_b$.
By inversion of $\bullet; \Gamma_f \vdash \mu f{:}\theta_f(\tau_f)\,.\,\lambda z{:}\theta_f(\tau_z)\,.\,\theta_f(e_b) : \tau_a \to \tau_b$,
we have the derivation:

$$\frac{\begin{array}{c}\bullet \vdash \Gamma_f \qquad \bullet \vdash \theta_f(\tau_f) \qquad \bullet \vdash \theta_f(\tau_z) \\ \bullet; \Gamma_f, f{:}\theta_f(\tau_f), z{:}\theta_f(\tau_z) \vdash \theta_f(e_b) : \tau_b \\ \theta_f(\tau_f) = \tau_a \to \tau_b \qquad \theta_f(\tau_z) = \tau_a\end{array}}{\bullet; \Gamma_f \vdash \mu f{:}\theta_f(\tau_f)\,.\,\lambda z{:}\theta_f(\tau_z)\,.\,\theta_f(e_b) : \tau_a \to \tau_b}$$

From $\vdash w_f : \tau_a \to \tau_b$ and $\theta_f(\tau_f) = \tau_a \to \tau_b$,
we have $\vdash w_f : \theta_f(\tau_f)$.
By Lemma 7 with $\vdash \rho : \Gamma$, $\Gamma(x_a) = \tau_a$, and $\rho(x_a) = w_a$,
we have $\vdash w_a : \tau_a$.

From $\vdash w_a : \tau_a$ and $\theta_f(\tau_z) = \tau_a$,
we have $\vdash w_a : \theta_f(\tau_z)$.
By Lemma 20 with $\vdash \phi_f : \theta_f$, $\hat{\phi} \vDash \phi_f$, $\vdash w_a : \theta_f(\tau_z)$, and $\hat{\phi}; \hat{\rho} \vDash w_a$,
we have $\hat{\phi} \vdash |w_a| :\approx\!\gg \tau_z$.
From $\{\hat{w}_a \in \hat{\rho}(x_a) \mid \hat{\phi} \vdash \hat{w}_a :\approx\!\gg \tau_z\} \subseteq \hat{\rho}(z)$, $\{|w_a|\} \subseteq \hat{\rho}(x_a)$,
and $\hat{\phi} \vdash |w_a| :\approx\!\gg \tau_z$,
we have $\{|w_a|\} \subseteq \hat{\rho}(z)$.
From $\hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor \texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e \rfloor) \rightsquigarrow \kappa$
and $\lfloor \texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e \rfloor = \lfloor e \rfloor$,
we have $\hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor e \rfloor) \rightsquigarrow \kappa$.
We can construct the derivation:

$$\hat{\phi}; \hat{\rho} \vDash e_b \qquad \hat{\phi} \vDash \phi_f$$

$$\dfrac{\dfrac{\hat{\phi}; \hat{\rho} \vDash \rho_f \qquad \hat{\phi}; \hat{\rho} \vDash w_f \qquad \{|w_f|\} \subseteq \hat{\rho}(f)}{\hat{\phi}; \hat{\rho} \vDash \rho_f, f \mapsto w_f} \qquad \hat{\phi}; \hat{\rho} \vDash w_a \qquad \{|w_a|\} \subseteq \hat{\rho}(z)}{\hat{\phi}; \hat{\rho} \vDash \rho_f, f \mapsto w_f, z \mapsto w_a}$$

$$\dfrac{\dfrac{\{\hat{w} \in \hat{\rho}(\lfloor e_b \rfloor) \mid \hat{\phi} \vdash \hat{w} :\approx\!\gg \tau_x\} \subseteq \hat{\rho}(x)}{\hat{\phi}; \hat{\rho} \vDash e \qquad \hat{\phi} \vDash \phi \qquad \hat{\phi}; \hat{\rho} \vDash \rho \qquad \hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor e \rfloor) \rightsquigarrow \kappa}}{\hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor e_b \rfloor) \rightsquigarrow \langle x; \tau_x; e; \phi; \rho \rangle :: \kappa}}{\hat{\phi}; \hat{\rho} \vDash \langle e_b; \phi_f; \rho_f, f \mapsto w_f, z \mapsto w_a; \langle x; \tau_x; e; \phi; \rho \rangle :: \kappa \rangle}$$

Thus, $\hat{\phi}; \hat{\rho} \vDash \langle e_b; \phi_f; \rho_f, f \mapsto w_f, z \mapsto w_a; \langle x; \tau_x; e; \phi; \rho \rangle :: \kappa \rangle$.

– $\dfrac{\rho(x_f) = w_f \qquad w_f = \langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi_f; \rho_f\rangle \qquad \pi_a = \langle \tau_a; \phi\rangle}{\begin{array}{l}\langle \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ x_f\ \texttt{[}\tau_a\texttt{]}\ \texttt{in}\ e; \phi; \rho; \kappa\rangle \\ \qquad \longrightarrow \langle e_b; \phi_f, \beta \mapsto \pi_a; \rho_f, f \mapsto w_f; \langle x; \tau_x; e; \phi; \rho\rangle :: \kappa\rangle\end{array}}$:

By inversion of $\hat{\phi}; \hat{\rho} \vDash \langle \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ x_f\ \texttt{[}\tau_a\texttt{]}\ \texttt{in}\ e; \phi; \rho; \kappa\rangle$,
we have the derivation

$$\dfrac{\hat{\phi}; \hat{\rho} \vDash x_f\ \texttt{[}\tau_a\texttt{]} \rightsquigarrow \hat{W} \qquad \{\hat{w} \in \hat{W} \mid \hat{\phi} \vdash \hat{w} :\!\approx\!\!> \tau_x\} \subseteq \hat{\rho}(x) \qquad \hat{\phi}; \hat{\rho} \vDash e}{\hat{\phi}; \hat{\rho} \vDash \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e}$$

$$\dfrac{\hat{\phi} \vDash \phi \qquad \hat{\phi}; \hat{\rho} \vDash \rho \qquad \hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ x_f\ \texttt{[}\tau_a\texttt{]}\ \texttt{in}\ e\rfloor) \rightsquigarrow \kappa}{\hat{\phi}; \hat{\rho} \vDash \langle \texttt{let}\ x\!:\!\tau_x\ \texttt{=}\ x_f\ \texttt{[}\tau_a\texttt{]}\ \texttt{in}\ e; \phi; \rho; \kappa\rangle}$$

By inversion of $\hat{\phi}; \hat{\rho} \vDash x_f\ \texttt{[}\tau_a\texttt{]} \rightsquigarrow \hat{W}$, we have the derivation:

$$\dfrac{\Psi}{\hat{\phi}; \hat{\rho} \vDash x_f\ \texttt{[}\tau_a\texttt{]} \rightsquigarrow \hat{W}}$$

where

$$\Psi = \bigwedge_{\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b \in \hat{\rho}(x_f)} \left(\{\tau_a\} \subseteq \hat{\phi}(\beta) \wedge \hat{\rho}(\lfloor e_b \rfloor) \subseteq \hat{W}\right)$$

By Lemma 18 with $\hat{\phi}; \hat{\rho} \vDash \rho$ and $\rho(x_f) = w_f$,
we have $\hat{\phi}; \hat{\rho} \vDash w_f$ and $\{|w_f|\} \subseteq \hat{\rho}(x_f)$.
From $\{|w_f|\} \subseteq \hat{\rho}(x_f)$
and $|w_f| = |\langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi_f; \rho_f\rangle| = \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b$,
we have $\{\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\} \subseteq \hat{\rho}(x_f)$.
From Ψ and $\{\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\} \subseteq \hat{\rho}(x_f)$,
we have $\{\tau_a\} \subseteq \hat{\phi}(\beta)$ and $\hat{\rho}(\lfloor e_b \rfloor) \subseteq \hat{W}$.
From $\{\hat{w} \in \hat{W} \mid \hat{\phi} \vdash \hat{w} :\!\approx\!\!> \tau_x\} \subseteq \hat{\rho}(x)$ and $\hat{\rho}(\lfloor e_b \rfloor) \subseteq \hat{W}$,
we have $\{\hat{w} \in \hat{\rho}(\lfloor e_b \rfloor) \mid \hat{\phi} \vdash \hat{w} :\!\approx\!\!> \tau_x\} \subseteq \hat{\rho}(x)$.
From $\hat{\phi}; \hat{\rho} \vDash w_f$ and $w_f = \langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi_f; \rho_f\rangle$,
we have $\hat{\phi}; \hat{\rho} \vDash \langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi_f; \rho_f\rangle$.
By inversion of $\hat{\phi}; \hat{\rho} \vDash \langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi_f; \rho_f\rangle$,
we have the derivation:

$$\dfrac{\dfrac{\{\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\} \subseteq \hat{\rho}(f) \qquad \hat{\phi}; \hat{\rho} \vDash e_b}{\hat{\phi}; \hat{\rho} \vDash \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b} \qquad \hat{\phi} \vDash \phi_f \qquad \hat{\phi}; \hat{\rho} \vDash \rho_f}{\hat{\phi}; \hat{\rho} \vDash \langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi_f; \rho_f\rangle}$$

From $\{\mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b\} \subseteq \hat{\rho}(f)$
and $|w_f| = |\langle \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b; \phi_f; \rho_f\rangle| = \mu f\!:\!\tau_f\,.\,\Lambda\beta\,.\,e_b$,

we have $\{|w_f|\} \subseteq \hat{\rho}(f)$.
We can construct the derivation:

$$\frac{\hat{\phi} \vDash \phi}{\hat{\phi} \vDash \langle \tau_a; \phi \rangle}$$

From $\hat{\phi} \vDash \langle \tau_a; \phi \rangle$ and $\pi_a = \langle \tau_a; \phi \rangle$,
we have $\hat{\phi} \vDash \pi_a$. From $\{\tau_a\} \subseteq \hat{\phi}(\beta)$ and $|\pi_a| = |\langle \tau_a; \phi \rangle| = \tau_a$,
we have $\{|\pi_a|\} \subseteq \hat{\phi}(\beta)$.
From $\hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor \texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ \texttt{[}\tau_a\texttt{]}\ \texttt{in}\ e \rfloor) \rightsquigarrow \kappa$
and $\lfloor \texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ \texttt{[}\tau_a\texttt{]}\ \texttt{in}\ e \rfloor = \lfloor e \rfloor$,
we have $\hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor e \rfloor) \rightsquigarrow \kappa$.
We can construct the derivation

$$\hat{\phi}; \hat{\phi}; \hat{\rho} \vDash e_b$$

$$\frac{\hat{\phi} \vDash \phi_f \qquad \hat{\phi} \vDash \pi_a \qquad \{|\pi_a|\} \subseteq \hat{\phi}(\beta)}{\hat{\phi} \vDash \phi_f, \beta \mapsto \langle \tau_a; \phi \rangle}$$

$$\frac{\hat{\phi}; \hat{\rho} \vDash \rho_f \qquad \hat{\phi}; \hat{\rho} \vDash w_f \qquad \{|w_f\} \subseteq \hat{\rho}(f)}{\hat{\phi}; \hat{\rho} \vDash \rho_f, f \mapsto w_f}$$

$$\frac{\dfrac{\{\hat{w} \in \hat{\rho}(\lfloor e_b \rfloor) \mid \hat{\phi} \vdash \hat{w} :\approx\!\!> \tau_x\} \subseteq \hat{\rho}(x) \qquad \hat{\phi}; \hat{\rho} \vDash e \qquad \hat{\phi} \vDash \phi \qquad \hat{\phi}; \hat{\rho} \vDash \rho \qquad \hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor e \rfloor) \rightsquigarrow \kappa}{\hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor e_b \rfloor) \rightsquigarrow \langle x; \tau_x; e; \phi; \rho \rangle :: \kappa}}{\hat{\phi}; \hat{\rho} \vDash \langle e_b; \phi_f, \beta \mapsto \pi_a; \rho_f, f \mapsto w_f; \langle x; \tau_x; e; \phi; \rho \rangle :: \kappa \rangle}$$

Thus, $\hat{\phi}; \hat{\rho} \vDash \langle e_b; \phi_f, \beta \mapsto \pi_a; \rho_f, f \mapsto w_f; \langle x; \tau_x; e; \phi; \rho \rangle :: \kappa \rangle$.

Theorem 22 (Preservation$*$ (Flow Soundness)).
If $\vdash \varsigma : \tau$, $\hat{\rho} \vDash \varsigma$, *and* $\varsigma \longrightarrow^* \varsigma'$, *then* $\hat{\rho} \vDash \varsigma'$.

Proof.
By induction on the derivation of $\varsigma \longrightarrow^* \varsigma'$, using Theorems 15 and 21.

Theorem 2 (Flow Soundness).
If $\bullet; \bullet \vdash e : \tau$, $\hat{\phi}; \hat{\rho} \vDash e$, *and* $\langle e; \bullet; \bullet; \circ \rangle \longrightarrow^* \langle e'; \phi'; \rho'; \kappa' \rangle$, *then* $\langle |\phi'|, |\rho'| \rangle \sqsubseteq \langle \hat{\phi}, \hat{\rho} \rangle$.

Proof.
By Theorem 22 with $\bullet; \bullet \vdash e : \tau$, $\hat{\phi}; \hat{\rho} \vDash \langle e; \bullet; \bullet; \circ \rangle$,
and $\langle e; \bullet; \bullet; \circ \rangle \longrightarrow^* \langle e'; \phi'; \rho'; \kappa' \rangle$,
we have $\hat{\phi}; \hat{\rho} \vDash \langle e'; \phi'; \rho'; \kappa' \rangle$.
By inversion of $\hat{\phi}; \hat{\rho} \vDash \langle e'; \phi'; \rho'; \kappa' \rangle$, we have the derivation:

$$\frac{\hat{\phi}; \hat{\rho} \vDash e' \qquad \hat{\phi} \vDash \phi' \qquad \hat{\phi}; \hat{\rho} \vDash \rho' \qquad \hat{\phi}; \hat{\rho} \vDash \hat{\rho}(\lfloor e' \rfloor) \rightsquigarrow \kappa}{\hat{\phi}; \hat{\rho} \vDash \langle e'; \phi'; \rho'; \kappa' \rangle}$$

By Lemma 16 with $\hat{\phi} \vDash \phi'$, we have $|\phi'| \sqsubseteq \hat{\phi}$.
By Lemma 17 with $\hat{\phi}; \hat{\rho} \vDash \rho'$, we have $|\rho'| \sqsubseteq \hat{\rho}$.
Thus, $|\phi'| \sqsubseteq \hat{\phi}$ and $|\rho'| \sqsubseteq \hat{\rho}$.