Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science
Bachelor's Program in Computer Science

Bachelor's Thesis

A Semantics for Lazy Types

submitted by
Georg Neis
on September 30, 2006

Supervisor
Prof. Dr. Gert Smolka

Advisor
Dipl.-Inform. Andreas Rossberg

Reviewers
Prof. Dr. Gert Smolka
Prof. Dr. Andreas Zeller

Statement

Hereby I confirm that this thesis is my own work and that I have documented all sources used.

Saarbrücken, September 30, 2006

Acknowledgment

I would like to thank Andreas Rossberg and Prof. Gert Smolka for offering me this topic. I am particularly grateful for the countless discussions with Andreas—I will miss them. Further thanks go to Andi Scharfstein for proof-reading the thesis.

Abstract

A central requirement for open and distributed programming is the ability to compose code at run time. In such programs type safety can no longer be guaranteed by static checks. Additional type checking at link time is necessary to ensure the integrity of the run-time system. Apart from that, it is desirable to perform linking as late as possible: this decreases the startup time of programs, keeps their working set small, and avoids loading unneeded code.

In this thesis, we present a system that is based on F_ω and models a language with support for such type-safe, late dynamic linking in the presence of higher-order polymorphism. Its key ingredients are intensional type analysis and lazy evaluation. Interesting implications arise from this combination. Type analysis needs to compare dynamic types, which have to be computed from (probably higher-order) type expressions that may contain types from other modules. If lazy linking has delayed the loading of these modules so far, then the corresponding types are represented by free type variables. For the type-level computation in our model this means that it may encounter free type variables whose denotation depends on yet unevaluated term-level expressions. To proceed, it inevitably has to eliminate these variables by triggering the necessary evaluations on term-level. In other words: type-level computation has to account for lazy types. We give two strategies for this and show soundness properties for the resulting calculi. The second strategy is more complicated than the first but provides a higher degree of laziness.

Contents

Chapter 1

Introduction

1.1 Motivation

A central requirement for open and distributed programming is the ability to compose code at run time. In such programs type safety can no longer be guaranteed by static checks. Additional type checking at link time is necessary to ensure the integrity of the run-time system. Apart from that, it is desirable to perform linking as late as possible. This decreases the startup time of programs, keeps their working set small, and avoids loading unneeded code.

Creating a formal model of a language with support for such type-safe, late dynamic linking requires two key ingredients:

- *intensional type analysis*, the ability to inspect the dynamic 'value' of type variables,
- *laziness*, the ability to delay a computation until its result is needed, and to avoid later recomputation by caching this result once it is available.

The integration of both techniques into a single system has an interesting impact on its dynamic semantics. Modeling dynamic type checking, type analysis has to compare run-time types in the form of arbitrary complex expressions, which demands for some form of normalization. Dynamic loading of code is a term-level operation but may introduce new types. If linking is lazy, however, then these new types have to be represented by free variables as long as the loading is deferred. Therefore, the normalization of type expressions may encounter free type variables that denote yet unknown types. To continue, these variables have to be eliminated by triggering the linking of the associated code. This necessitates a new concept: *lazy types.*

1.2 Alice ML

Alice ML [3, 22] is a programming language designed for *typed open programming.* It is based on Standard ML [15] and hence comes with a strong static type system. Alice ML provides support for concurrent, distributed and constraint programming. That in particular includes the ability to load arbitrary code at run time by importing so-called *components* [22, 20], which can be understood as containers for modules. To ensure type safety, Alice ML performs dynamic type checks in addition to the usual compile-time checking. This requires the availability of run-time type information.

Another central concept of Alice ML are *futures* [22, 17]. They come in different flavours, of which only *lazy futures* are of interest for us. Lazy futures add lazy evaluation to the language: using special syntax it is possible to suspend the evaluation of an expression until its value is actually needed.

```
(* A *)
type t = int
val x : t = 42

(* B *)
import type t; val x : t from "A"
type u = t
val y : u × u = (x,x)

(* C *)
import type u = int; val y : u × u from "B"
type v = int
val z = #1 y
```

Figure 1.1: An example from Alice ML

The mechanism for importing components implicitely makes use of the laziness, i.e., components are loaded at the latest possible point in time. This is called *lazy linking* and provides the aforementionend benefits such as decreasing the startup time of programs.

Dynamic type checking is carried out when the import of a component is triggered and works by comparing the run-time signature of the contained module with a statically specified one. The type expressions that are to be compared may contain free type variables representing types from modules, which—due to the lazy linking—have not been loaded yet. Hence, in order to obtain the denotation of these types the import of the respective components has to be triggered first. Figure 1.1 shows a short example consisting of three components, each of which may reside in a separate file. What happens when component C is evaluated? Since imports are lazy, the first declaration at this point only introduces the new names u and y as lazy futures. The definition of the type alias v has no operational effect either. In the binding of z, the first constituent of y is requested. This triggers the import from component B. There, the first two statements again only introduce new names. The tuple (x,x) can be constructed without knowing the proper value of x, so the binding of y does not cause any evaluation. However, the import of B is not finished yet. What remains is the dynamic type check: in this case, to make sure that type u really is equal to type int. Due to the previous examination of B it is already known that u is a lazy reference to type t, which is imported from component A. For this reason, in order for the type check to proceed, A has to be loaded as well at this point—even though it is not directly accessed in C.

1.3 Outline

In this thesis, we develop a calculus that is based on F_ω and models the integration of dynamic type checking with laziness. Our main focus lies on the type-level reduction. We present two deterministic strategies, which differ significantly in their degree of laziness, and prove soundness for them.

Chapter 2

The basic calculus

2.1 System F_ω

We build our calculus on top of F_ω, the higher-order polymorphic lambda calculus [11, 18]. This gives us sufficient expressive power to model the ML core language[1]. Let us examine a few examples. The identity function, `fn x ⇒ x`, can be translated into F_ω as $\lambda\alpha{:}\Omega.\lambda x{:}\alpha.x$. In contrast to SML, we have to explicitly specify the type parameters of a polymorphic function. The declaration `type (α, β) state = α → α × β` introduces a type constructor that corresponds to the type-level function $\lambda\alpha{:}\Omega.\lambda\beta{:}\Omega.\alpha \to \alpha \times \beta$. The application of type constructors is backwards in SML: the F_ω equivalent of the type `(int,int) state` is $(\lambda\alpha{:}\Omega.\lambda\beta{:}\Omega.\alpha \to \alpha \times \beta)$ `int int`. Some other typical functions are:

fun flip f x y = f y x
  $\rightsquigarrow$ $\mathit{flip} = \lambda X{:}\Omega.\lambda Y{:}\Omega.\lambda Z{:}\Omega.\lambda f{:}Y \to X \to Z.\lambda x{:}X.\lambda y{:}Y.f\ y\ x$
fun compose f g = fn x ⇒ f (g x)
  $\rightsquigarrow$ $\mathit{compose} = \lambda X{:}\Omega.\lambda Y{:}\Omega.\lambda Z{:}\Omega.\lambda f{:}Y \to Z.\lambda g{:}X \to Y.\lambda x{:}X.f\ (g\ x)$
fun compose' f g = flip compose f g
  $\rightsquigarrow$ $\mathit{compose'} = \lambda X{:}\Omega.\lambda Y{:}\Omega.\lambda Z{:}\Omega.\mathit{flip}\ (Y \to Z)\ (X \to Y)\ (X \to Z)\ (\mathit{compose}\ X\ Y\ Z)$

Figure 2.1 shows the syntax of F_ω. The operational (small-step) semantics is formalized with the help of evaluation contexts [10]. Figure 2.2 defines a common left-to-right call-by-value reduction relation. We write $a[b := c]$ for the capture-free substitution of b by c in a.

term variables	$x \in \mathit{Var}$
type variables	$\alpha \in \mathit{TVar}$
terms	$e ::= x \mid \lambda x{:}\tau.e \mid e_1\ e_2 \mid \lambda\alpha{:}\kappa.e \mid e\ \tau$
types	$\tau ::= \alpha \mid \tau_1 \to \tau_2 \mid \forall\alpha{:}\kappa.\tau \mid \lambda\alpha{:}\kappa.\tau \mid \tau_1\ \tau_2$
kinds	$\kappa ::= \Omega \mid \kappa_1 \to \kappa_2$

Figure 2.1: Syntax of F_ω

The static semantics is defined by the three judgements $\Gamma \vdash \Box$ (well-formedness of environments), $\Gamma \vdash \tau : \kappa$ (well-formedness of types), and $\Gamma \vdash e : \tau$ (well-formedness of terms). Figure 2.3 lists the corresponding inference rules. The rules for environments (prefixed with E-) guarantee that a well-formed environment can be interpreted as a partial function from term and type

[1]Mainly with regard to the type system. While F_ω does not allow the definition of recursive functions, this will become possible as a side effect of adding a typecase operator.

values	$v ::= \lambda x{:}\tau.e \mid \lambda\alpha{:}\kappa.e$
evaluation contexts	$E ::= \_ \mid E\ e \mid (\lambda x{:}\tau.e)\ E \mid E\ \tau$

Reduction $\boxed{e \longrightarrow e'}$

R-App	$E[(\lambda x{:}\tau.e)\ v] \longrightarrow E[e[x := v]]$
R-Inst	$E[(\lambda\alpha{:}\kappa.e)\ \tau] \longrightarrow E[e[\alpha := \tau]]$

Figure 2.2: Operational semantics of F_ω

Well-formedness of environments $\boxed{\Gamma \vdash \Box}$

$$(\text{E-Empty})\ \frac{}{\cdot \vdash \Box} \qquad (\text{E-Type})\ \frac{\Gamma \vdash \Box \qquad \alpha \notin dom(\Gamma)}{\Gamma, \alpha{:}\kappa \vdash \Box}$$

$$(\text{E-Term})\ \frac{\Gamma \vdash \tau : \Omega \qquad x \notin dom(\Gamma)}{\Gamma, x{:}\tau \vdash \Box}$$

Well-formedness of types $\boxed{\Gamma \vdash \tau : \kappa}$

$$(\text{K-Var})\ \frac{\Gamma \vdash \Box}{\Gamma \vdash \alpha : \Gamma(\alpha)} \qquad (\text{K-Arrow})\ \frac{\Gamma \vdash \tau_1 : \Omega \qquad \Gamma \vdash \tau_2 : \Omega}{\Gamma \vdash \tau_1 \to \tau_2 : \Omega}$$

$$(\text{K-Univ})\ \frac{\Gamma, \alpha{:}\kappa \vdash \tau : \Omega}{\Gamma \vdash \forall\alpha{:}\kappa.\tau : \Omega} \qquad (\text{K-Abs})\ \frac{\Gamma, \alpha{:}\kappa \vdash \tau : \kappa'}{\Gamma \vdash \lambda\alpha{:}\kappa.\tau : \kappa \to \kappa'}$$

$$(\text{K-App})\ \frac{\Gamma \vdash \tau_1 : \kappa_2 \to \kappa \qquad \Gamma \vdash \tau_2 : \kappa_2}{\Gamma \vdash \tau_1\ \tau_2 : \kappa}$$

Well-formedness of terms $\boxed{\Gamma \vdash e : \tau}$

$$(\text{T-Equiv})\ \frac{\Gamma \vdash e : \tau' \qquad \tau' \equiv \tau \qquad \Gamma \vdash \tau : \Omega}{\Gamma \vdash e : \tau} \qquad (\text{T-Var})\ \frac{\Gamma \vdash \Box}{\Gamma \vdash x : \Gamma(x)}$$

$$(\text{T-Abs})\ \frac{\Gamma, x{:}\tau \vdash e : \tau'}{\Gamma \vdash \lambda x{:}\tau.e : \tau \to \tau'} \qquad (\text{T-App})\ \frac{\Gamma \vdash e_1 : \tau_2 \to \tau \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\ e_2 : \tau}$$

$$(\text{T-Gen})\ \frac{\Gamma, \alpha{:}\kappa \vdash e : \tau}{\Gamma \vdash \lambda\alpha{:}\kappa.e : \forall\alpha{:}\kappa.\tau} \qquad (\text{T-Inst})\ \frac{\Gamma \vdash e : \forall\alpha{:}\kappa.\tau' \qquad \Gamma \vdash \tau : \kappa}{\Gamma \vdash e\ \tau : \tau'[\alpha := \tau]}$$

Figure 2.3: Static semantics of F_ω

Equivalence of types $\boxed{\tau \equiv \tau'}$

$$(\text{Q-R{\small EFL}})\ \frac{}{\tau \equiv \tau} \qquad (\text{Q-S{\small YMM}})\ \frac{\tau_2 \equiv \tau_1}{\tau_1 \equiv \tau_2}$$

$$(\text{Q-T{\small RANS}})\ \frac{\tau_1 \equiv \tau_2 \qquad \tau_2 \equiv \tau_3}{\tau_1 \equiv \tau_3} \qquad (\text{Q-A{\small RROW}})\ \frac{\tau_1 \equiv \tau_1' \qquad \tau_2 \equiv \tau_2'}{\tau_1 \to \tau_2 \equiv \tau_1' \to \tau_2'}$$

$$(\text{Q-U{\small NIV}})\ \frac{\tau_1 \equiv \tau_2}{\forall\alpha{:}\kappa.\tau_1 \equiv \forall\alpha{:}\kappa.\tau_2} \qquad (\text{Q-A{\small BS}})\ \frac{\tau_1 \equiv \tau_2}{\lambda\alpha{:}\kappa.\tau_1 \equiv \lambda\alpha{:}\kappa.\tau_2}$$

$$(\text{Q-A{\small PP}})\ \frac{\tau_1 \equiv \tau_1' \qquad \tau_2 \equiv \tau_2'}{\tau_1\ \tau_2 \equiv \tau_1'\ \tau_2'} \qquad (\text{Q-B{\small ETA}})\ \frac{}{(\lambda\alpha{:}\kappa.\tau_1)\ \tau_2 \equiv \tau_1[\alpha := \tau_2]}$$

Figure 2.4: Type equivalence of F_ω

variables to types and kinds, respectively, by asserting that no variable is bound twice in the same environment. We write $dom(\Gamma)$ to denote the domain of such a function.

The rule T-Equiv accounts for the fact that, due to the existence of type constructors in F_ω, there are type expressions that are syntactically different but semantically equal. The corresponding type equivalence is defined in figure 2.4.

2.2 Pairs

Even though pairs can be encoded in F_ω, we explicitely add them to the language on both term and type level. By nesting pairs we are then able to represent tuples of arbitrary many terms (or types):

$$\langle e_1, \langle e_2, \langle e_3, \ldots \rangle\rangle\rangle$$

Figure 2.5 shows the extensions necessary for adding pairs of terms to our calculus. Instead of introducing two projection operations—one for extracting the first component of a pair and one for extracting the second—we use a single construct reminiscent of pattern matching[2]: let $\langle x_1, x_2 \rangle$ = e_1 in e_2. x_1 is bound to the first constituent of the pair e_1 and x_2 to the second. Their scope is e_2. The corresponding reduction rule, R-Proj, is straightforward using substitution.

The extensions for pairs of types are shown in figure 2.6. Here we decided to use the ordinary projections $\tau.1$ and $\tau.2$ (where τ must be of type $\langle \tau_1, \tau_2 \rangle$). If κ_1 is the kind of τ_1 and κ_2 the one of τ_2, then the kind of $\langle \tau_1, \tau_2 \rangle$ is $\kappa_1 \times \kappa_2$—analogous to the types of term pairs.

2.3 Existential types

We will now introduce *existential types* [18, 16]. Intuitively, an element of an existential type is a pair consisting of a type and a term. This motivates the syntactic form $\langle \tau, e \rangle$ for constructing such terms (we call them *packages*). Their type, $\exists\alpha{:}\kappa.\tau'$, can be read as "there exists a type, which we call α, such that the term component has type τ'" (where α usually occurs freely in τ'). Of course, the existentially quantified variable α stands for the type component of the package and hence the term component has type $\tau'[\alpha := \tau]$.

[2]The reason for the choice of the non-standard projection is its similarity to the construct for opening packages that we will introduce next.

terms	$e ::= \cdots \mid \langle e_1, e_2 \rangle \mid \mathsf{let}\ \langle x_1, x_2 \rangle = e_1\ \mathsf{in}\ e_2$
types	$\tau ::= \cdots \mid \tau_1 \times \tau_2$
values	$v ::= \cdots \mid \langle v_1, v_2 \rangle$
evaluation contexts	$E ::= \cdots \mid \langle E, e \rangle \mid \langle v, E \rangle \mid \mathsf{let}\ \langle x_1, x_2 \rangle = E\ \mathsf{in}\ e$

Reduction $\boxed{e \longrightarrow e'}$

...

R-Proj $\quad E[\mathsf{let}\ \langle x_1, x_2 \rangle = \langle v_1, v_2 \rangle\ \mathsf{in}\ e] \longrightarrow E[e[x_1 := v_1][x_2 := v_2]]$

Well-formedness of types $\boxed{\Gamma \vdash \tau : \kappa}$

$$\ldots \qquad (\text{K-Times})\ \frac{\Gamma \vdash \tau_1 : \Omega \qquad \Gamma \vdash \tau_2 : \Omega}{\Gamma \vdash \tau_1 \times \tau_2 : \Omega}$$

Well-formedness of terms $\boxed{\Gamma \vdash e : \tau}$

$$\ldots \qquad (\text{T-Pair})\ \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2}$$

$$(\text{T-Proj})\ \frac{\Gamma \vdash e_1 : \tau_1 \times \tau_2 \qquad \Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_2 : \tau}{\Gamma \vdash \mathsf{let}\ \langle x_1, x_2 \rangle = e_1\ \mathsf{in}\ e_2 : \tau}$$

Equivalence of types $\boxed{\tau \equiv \tau'}$

$$\ldots \qquad (\text{Q-Times})\ \frac{\tau_1 \equiv \tau_1' \qquad \tau_2 \equiv \tau_2'}{\tau_1 \times \tau_2 \equiv \tau_1' \times \tau_2'}$$

Figure 2.5: Pairs on term level

$$\begin{array}{lll} \text{types} & \tau ::= & \cdots \mid \langle \tau_1, \tau_2 \rangle \mid \tau.1 \mid \tau.2 \\ \text{kinds} & \kappa ::= & \cdots \mid \kappa_1 \times \kappa_2 \end{array}$$

Well-formedness of types $\boxed{\Gamma \vdash \tau : \kappa}$

$$\ldots \qquad (\textsc{K-Pair})\ \frac{\Gamma \vdash \tau_1 : \kappa_1 \qquad \Gamma \vdash \tau_2 : \kappa_2}{\Gamma \vdash \langle \tau_1, \tau_2 \rangle : \kappa_1 \times \kappa_2}$$

$$(\textsc{K-Proj1})\ \frac{\Gamma \vdash \tau : \kappa_1 \times \kappa_2}{\Gamma \vdash \tau.1 : \kappa_1} \qquad (\textsc{K-Proj2})\ \frac{\Gamma \vdash \tau : \kappa_1 \times \kappa_2}{\Gamma \vdash \tau.2 : \kappa_2}$$

Equivalence of types $\boxed{\tau \equiv \tau'}$

$$\ldots \qquad (\textsc{Q-Pair})\ \frac{\tau_1 \equiv \tau_1' \qquad \tau_2 \equiv \tau_2'}{\langle \tau_1, \tau_2 \rangle \equiv \langle \tau_1', \tau_2' \rangle}$$

$$(\textsc{Q-Proj1a})\ \frac{\tau \equiv \tau'}{\tau.1 \equiv \tau'.1} \qquad (\textsc{Q-Proj1b})\ \frac{}{\langle \tau_1, \tau_2 \rangle.1 \equiv \tau_1}$$

$$(\textsc{Q-Proj2a})\ \frac{\tau \equiv \tau'}{\tau.2 \equiv \tau'.2} \qquad (\textsc{Q-Proj2b})\ \frac{}{\langle \tau_1, \tau_2 \rangle.2 \equiv \tau_2}$$

Figure 2.6: Pairs on type level

However, the type and term component alone do not uniquely fix the existential type of a package. For instance[3], both $\exists\alpha{:}\Omega.\mathsf{int}$ and $\exists\alpha{:}\Omega.\alpha$ are valid types for $\langle \mathsf{int}, 42\rangle$. For this reason we explicitely annotate a package with the intended existential type and hence write e.g. $\langle \mathsf{int}, 42\rangle{:}\exists\alpha{:}\Omega.\alpha$.

Figure 2.7 shows the required additions to the syntax and semantics definitions. The premise $\Gamma \vdash \exists\alpha{:}\kappa.\tau' : \Omega$ of the typing rule T-Close is not strictly necessary. We use it nevertheless to simplify a later proof (proposition 9). Packages are *opened* using the $\mathsf{let}\ \langle \alpha, x\rangle = e_1\ \mathsf{in}\ e_2$ construct. Here e_1 must evaluate to a package. α is then bound to its type component and x to its term component and both can be used within e_2. The corresponding rule T-Open requires a side condition, $\alpha \notin \mathit{ftv}(\tau)$, to ensure that α does not leave its scope and appears freely in the type of the whole expression. $\mathit{ftv}(\tau)$ denotes the set of type variables that occur freely in τ. Later we also use $\mathit{fv}(X)$ as well as $\mathit{btv}(X)$ and $\mathit{bv}(X)$ to refer to the set of free term variables, bound type variables, and bound term variables, respectively, of some syntactic form X. For instance, if $\tau = \forall\alpha{:}\Omega.\alpha$ and $E = \_\ x$, then $\mathit{btv}(\tau) = \{\alpha\}$ and $\mathit{fv}(E) = \{x\}$.

Existential types are mainly used for modeling abstract data types [7, 16]. The type component of a package is then understood as the encapsulated representation type and the term component as the implementation that provides the operations on values of the abstract type. Since the language offers no way of inspecting the encapsulated type it is impossible to circumvent the abstraction. However, with the extension we undertake in the next section, this property does no longer hold for our calculus. Measures of repair are known (see chapter 4) but they fall outside the scope of this thesis. We therefore use existential types to model simple SML modules[4] without caring about information hiding.

[3]To make examples more intuitive we often assume that our system has been extended with integers.

[4]In fact, F_ω is expressive enough to encode almost the complete SML module system. The only exception are

terms	$e ::=$	$\cdots \mid \langle\tau_1, e\rangle{:}\tau_2 \mid \mathsf{let}\ \langle\alpha, x\rangle = e_1\ \mathsf{in}\ e_2$
types	$\tau ::=$	$\cdots \mid \exists\alpha{:}\kappa.\tau$
values	$v ::=$	$\cdots \mid \langle\tau_1, v\rangle{:}\tau_2$
evaluation contexts	$E ::=$	$\cdots \mid \langle\tau_1, E\rangle{:}\tau_2 \mid \mathsf{let}\ \langle\alpha, x\rangle = E\ \mathsf{in}\ e$

Reduction $\boxed{e \longrightarrow e'}$

...

R-Open $E[\mathsf{let}\ \langle\alpha, x\rangle = \langle\tau, v\rangle{:}\tau'\ \mathsf{in}\ e] \longrightarrow E[e[\alpha := \tau][x := v]]$

Well-formedness of types $\boxed{\Gamma \vdash \tau : \kappa}$

$$\ldots \qquad (\text{K-Exist})\ \frac{\Gamma, \alpha{:}\kappa \vdash \tau : \Omega}{\Gamma \vdash \exists\alpha{:}\kappa.\tau : \Omega}$$

Well-formedness of terms $\boxed{\Gamma \vdash e : \tau}$

$$\ldots \qquad (\text{T-Close})\ \frac{\Gamma \vdash \tau : \kappa \qquad \Gamma \vdash e : \tau'[\alpha := \tau] \qquad \Gamma \vdash \exists\alpha{:}\kappa.\tau' : \Omega}{\Gamma \vdash \langle\tau, e\rangle{:}\exists\alpha{:}\kappa.\tau' : \exists\alpha{:}\kappa.\tau'}$$

$$(\text{T-Open})\ \frac{\Gamma \vdash e_1 : \exists\alpha{:}\kappa.\tau' \qquad \Gamma, \alpha{:}\kappa, x{:}\tau' \vdash e_2 : \tau \qquad \alpha \notin ftv(\tau)}{\Gamma \vdash \mathsf{let}\ \langle\alpha, x\rangle = e_1\ \mathsf{in}\ e_2 : \tau}$$

Equivalence of types $\boxed{\tau \equiv \tau'}$

$$\ldots \qquad (\text{Q-Exist})\ \frac{\tau_1 \equiv \tau_2}{\exists\alpha{:}\kappa.\tau_1 \equiv \exists\alpha{:}\kappa.\tau_2}$$

Figure 2.7: Existential types

The following SML code shows a module implementing *options*:

```
structure S = struct
  datatype α option = None | Some of α
  fun ocase None z f = z
    | ocase (Some x) z f = f x
end
```

It can be encoded in our calculus as

$$S = \langle option, \langle None, \langle Some, ocase\rangle\rangle\rangle{:}\exists O{:}\Omega \to \Omega.\langle None\text{-}Type, \langle Some\text{-}Type, ocase\text{-}Type\rangle\rangle$$

where the occuring abbreviations are defined as follows:

$$\begin{array}{lcl} option & = & \lambda\alpha{:}\Omega.\forall\beta{:}\Omega.\beta \to (\alpha \to \beta) \to \beta \\ None & = & \lambda\alpha{:}\Omega.\lambda\beta{:}\Omega.\lambda z{:}\beta.\lambda f{:}\alpha \to \beta.z \\ Some & = & \lambda\alpha{:}\Omega.\lambda x{:}\alpha.\lambda\beta{:}\Omega.\lambda z{:}\beta.\lambda f{:}\alpha \to \beta.f\ x \\ ocase & = & \lambda\alpha{:}\Omega.\lambda o{:}(\forall\beta{:}\Omega.\beta \to (\alpha \to \beta) \to \beta).o \\ \\ None\text{-}Type & = & \forall\alpha{:}\Omega.O\ \alpha \\ Some\text{-}Type & = & \forall\alpha{:}\Omega.\alpha \to O\ \alpha \\ ocase\text{-}Type & = & \forall\alpha{:}\Omega.O\ \alpha \to \forall\beta{:}\Omega.\beta \to (\alpha \to \beta) \to \beta \end{array}$$

It may then be used by simply unpacking it corresponding to its signature (the existential type), e.g.:

$$\mathsf{let}\ \langle O, x\rangle = S\ \mathsf{in}\ \mathsf{let}\ \langle None, x'\rangle = x\ \mathsf{in}\ \mathsf{let}\ \langle Some, ocase\rangle = x'\ \mathsf{in}\ \ldots$$

If a module exports more than one type, then the type component of the package is a tuple—like the term part in the example above.

A completely different application area for existential types is the representation of *dynamics* [1, 2, 13]. A dynamic basically is a pair consisting of a term and its type and has the fixed type Dynamic. The corresponding existential type therefore is $\exists\alpha{:}\Omega.\alpha$.

Alice ML's components are best understood as dynamics that contain a single module together with its signature.

2.4 Typecase

Dynamics per se are useless without the ability to inspect the run-time type they carry. This analysis is exactly the purpose of the *typecase* operator [1]: its behaviour depends on whether the dynamic type matches a specified one. More concretely, it takes a dynamic, a type, and two terms. If the embedded type of the dynamic equals the specified type, then the first term is evaluated, otherwise the second. A typical, illustrating example is a function that returns a string representation of a dynamic value (written in an imaginary extension of SML):

```
fun show_dyn d =
  typecase d of v:int then Int.toString v
  else typecase d of v:real then Real.toString v
  else typecase d of v:string then "\"" ^ v ^ "\""
  else typecase d of v:bool then (if v then "true" else "false")
  else "<unknown>"
```

transparent signatures, which can be represented with the help of *singleton kinds* [24].

terms $\quad e ::= \cdots \mid \mathsf{tcase}\ e{:}\tau_1\ \mathsf{of}\ x{:}\tau_2\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$

evaluation contexts $\quad E ::= \cdots \mid \mathsf{tcase}\ E{:}\tau_1\ \mathsf{of}\ x{:}\tau_2\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$

Well-formedness of terms $\boxed{\Gamma \vdash e : \tau}$

$$\ldots \qquad (\text{T-Case})\ \frac{\Gamma \vdash e : \tau_1 \qquad \Gamma, x{:}\tau_2 \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathsf{tcase}\ e{:}\tau_1\ \mathsf{of}\ x{:}\tau_2\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau}$$

Figure 2.8: Typecase

We augment our calculus with a typecase construct that is slightly more general insofar that it is independent from dynamics, and hence fits better into our system. Figure 2.8 shows the additional syntax and typing rule. The expression e_0 in $\mathsf{tcase}\ e{:}\tau\ \mathsf{of}\ x{:}\tau'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$ must be of type τ—the type we want to analyse (we could infer it but that would force us to make the dynamic semantics dependent on the static semantics). If τ equals τ', then the whole expression reduces to $e_1[x := v]$ (where v is the value of e), otherwise to e_2. The exact rules are given in chapter 3. Obviously, our strategy must not allow reduction below binders—otherwise a typecase expression within a polymorphic function may compare the parameter variable instead of the substituted run-time type.

A shortened version of the print_dyn function then looks like:

$$\lambda x{:}\exists\alpha{:}\Omega.\alpha.\mathsf{tcase}\ x{:}\alpha\ \mathsf{of}\ x'{:}\mathsf{int}\ \mathsf{then}\ \mathsf{Int.toString}\ x'\ \mathsf{else}\ \text{"<unknown>"}$$

This example also makes clear why the tcase has to introduce a new variable: even in the case of α being dynamically equal to int, Int.toString x is ill-typed because x is statically of type α, while Int.toString expects an argument of type int.

It is important to note that, due to the introduction of tcase, our calculus is no longer terminating. It is now possible to express a fixpoint operator[5], which allows the construction of diverging recursive functions.

2.5 Lazy evaluation

The last missing ingredient for our model is lazy evaluation. There are two approaches for modeling lazy evaluation in a lambda calculus. The first one is to make it implicit by building the laziness directly into the reduction rule(s). The result is a pure call-by-need calculus like [4]. We choose the second way: our reduction strategy itself stays call-by-value, but we introduce a lazy construct that allows us to *selectively* defer the evaluation of expressions as long as possible. This resembles Alice ML, which is an *eager* (or *strict*) language like Standard ML but—thanks to its features for concurrency programming—provides explicit lazy evaluation in the form of lazy futures [17]. For example, lazy 3+5 immediately evaluates to such a future, which acts as a placeholder for the actual value of the sum and can be treated like any other term. When it is later used in such a way that the result of the addition is required, then the computation is triggered and its result substitutes the future.

[5]The idea is similar to constructing a fixpoint operator with the help of recursive types. Details can be found in [1] (using dynamics) and [6] (using universal types).

The natural extension for providing explicit laziness in a lambda calculus on term level is adding a $\mathsf{lazy}\ x = e_1\ \mathsf{in}\ e_2$ construct and treating term variables as values. The Alice ML phrase lazy 3+5 might then be encoded as $\mathsf{lazy}\ x = 3 + 5\ \mathsf{in}\ x$ (assuming that we add integers to the calculus). The idea is that in order to suspend the evaluation of e_1 (here the sum) we bind it to x and then use this variable as a placeholder. Consider the following example:

$$\mathsf{lazy}\ x = 4 \cdot 7\ \mathsf{in}\ (\lambda x'{:}\mathsf{int}.x' + 1)\ x$$

Here the lazy-bound variable x occurs as the right-hand side of an application, the left-hand side of which is an abstraction. Because x is considered a value, β-reduction can be performed. That way we successfully avoid to trigger the evaluation of $4 \cdot 7$ at this point.

In our system, where the tcase inspects dynamic types, we want to have lazy types as well. We therefore abandon the simple lazy expression in favour of a lazy variant for opening packages: $\mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ e_2$. (The different syntax for type variables, ζ, is necessary because we must be able to distinguish them from normal type variables later.) The idea of achieving laziness is basically the same as before: the evaluation of e_1, which must have an existential type, is suspended by using ζ and x as deputies for the package's contents—until their actual value is required (what exactly that means will be made precise soon). Furthermore, the simple lazy construct can be emulated by creating a dummy package that contains the term of interest and some *don't care* type.

It is clear that our evaluation contexts must not allow the reduction of e_1 in $\mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ e_2$. On the other hand, it must be possible to reduce e_2. We achieve this by introducing a set of *lazy contexts*, syntactic shortcuts for chains of lazy bindings, and slightly modifying the existing reduction rules. For example, the rule for reducing applications will then read

$$LE[(\lambda x{:}\tau.e)\ v] \quad \longrightarrow LE[e[x := v]]$$

where L is defined as

$$L ::= \ \_ \mid \mathsf{lazy}\ \langle\zeta, x\rangle = e\ \mathsf{in}\ L$$

and E remains unchanged.

But what happens if e.g. the whole term $\mathsf{lazy}\ x = e_1\ \mathsf{in}\ x$ is the right-hand side of an application? Consider $(\lambda x{:}\mathsf{int}.x \cdot x)\ (\mathsf{lazy}\ x = 4 + 9\ \mathsf{in}\ x)$. If we treat lazy expressions as values too, then β reduction can be performed, yielding $(\mathsf{lazy}\ x = 4 + 9\ \mathsf{in}\ x) \cdot (\mathsf{lazy}\ x = 4 + 9\ \mathsf{in}\ x)$. Unfortunately, no *sharing* takes place here: in the end, $4 + 9$ will be calculated twice! For this reason we do not treat lazy expressions as values but add another reduction rule, R-Suspend:

$$LE[\mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ e_2] \quad \longrightarrow L[\mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ E[e_2]]$$

The intuition is the following: whenever reduction comes across a lazy, the binding is lifted above the surrounding evaluation context while the body is not moved. This is called *scope extrusion* and causes the lazy bindings to accumulate at the front of the term as reduction proceeds. Two side conditions must be taken into accout, though. First of all, E must not be the trivial (empty) context—otherwise the term will always reduce to itself. Furthermore, neither ζ nor x may appear in E to avoid capturing. Reducing the previous example by R-Suspend now has the effect of dragging the abstraction into the body of the lazy expression, leading to $\mathsf{lazy}\ x = 4 + 9\ \mathsf{in}\ (\lambda x{:}\mathsf{int}.x \cdot x)\ x$. This term can then be further simplified using R-App to $\mathsf{lazy}\ x = 4+9\ \mathsf{in}\ x \cdot x$, thus sharing the computation of $4+9$. Figure 2.9 shows a sample sequence of R-Suspend reductions.

The remaining question is when and how the evaluation of a lazy package is triggered. The answer to the first part is: it is triggered when reduction hits upon an occurrence of the corresponding lazy type or term variable in a *strict position.* Lazy type variables are covered in the

$$\begin{array}{ll} & \langle \mathsf{lazy}\ x_1 = 8/4\ \mathsf{in}\ x_1, (\lambda x_2{:}\mathsf{int}.5 + x_2)\ (\mathsf{lazy}\ x_3 = 2 \cdot 3\ \mathsf{in}\ ((\mathsf{lazy}\ x_4 = 7 - 4\ \mathsf{in}\ x_4) - x_3))\rangle \\ \longrightarrow & \mathsf{lazy}\ x_1 = 8/4\ \mathsf{in}\ \langle x_1, (\lambda x_2{:}\mathsf{int}.5 + x_2)\ (\mathsf{lazy}\ x_3 = 2 \cdot 3\ \mathsf{in}\ ((\mathsf{lazy}\ x_4 = 7 - 4\ \mathsf{in}\ x_4) - x_3))\rangle \\ \longrightarrow & \mathsf{lazy}\ x_1 = 8/4\ \mathsf{in}\ \mathsf{lazy}\ x_3 = 2 \cdot 3\ \mathsf{in}\ \langle x_1, (\lambda x_2{:}\mathsf{int}.5 + x_2)\ ((\mathsf{lazy}\ x_4 = 7 - 4\ \mathsf{in}\ x_4) - x_3)\rangle \\ \longrightarrow & \mathsf{lazy}\ x_1 = 8/4\ \mathsf{in}\ \mathsf{lazy}\ x_3 = 2 \cdot 3\ \mathsf{in}\ \mathsf{lazy}\ x_4 = 7 - 4\ \mathsf{in}\ \langle x_1, (\lambda x_2{:}\mathsf{int}.5 + x_2)\ (x_4 - x_3)\rangle \end{array}$$

Figure 2.9: Suspension

$$\begin{array}{ll} & \mathsf{lazy}\ x_1 = 8/4\ \mathsf{in}\ \mathsf{lazy}\ x_3 = 2 \cdot 3\ \mathsf{in}\ \mathsf{lazy}\ x_4 = 7 - 4\ \mathsf{in}\ \langle x_1, (\lambda x_2{:}\mathsf{int}.5 + x_2)\ (x_4 - x_3)\rangle \\ \longrightarrow & \mathsf{lazy}\ x_1 = 8/4\ \mathsf{in}\ \mathsf{lazy}\ x_3 = 2 \cdot 3\ \mathsf{in}\ \mathsf{let}\ x_4 = 7 - 4\ \mathsf{in}\ \langle x_1, (\lambda x_2{:}\mathsf{int}.5 + x_2)\ (x_4 - x_3)\rangle \\ \longrightarrow & \mathsf{lazy}\ x_1 = 8/4\ \mathsf{in}\ \mathsf{lazy}\ x_3 = 2 \cdot 3\ \mathsf{in}\ \mathsf{let}\ x_4 = 3\ \mathsf{in}\ \langle x_1, (\lambda x_2{:}\mathsf{int}.5 + x_2)\ (x_4 - x_3)\rangle \\ \longrightarrow & \mathsf{lazy}\ x_1 = 8/4\ \mathsf{in}\ \mathsf{lazy}\ x_3 = 2 \cdot 3\ \mathsf{in}\ \langle x_1, (\lambda x_2{:}\mathsf{int}.5 + x_2)\ (x_3 - x_3)\rangle \\ \longrightarrow & \mathsf{lazy}\ x_1 = 8/4\ \mathsf{in}\ \mathsf{let}\ x_3 = 2 \cdot 3\ \mathsf{in}\ \langle x_1, (\lambda x_2{:}\mathsf{int}.5 + x_2)\ (x_3 - x_3)\rangle \\ \longrightarrow & \mathsf{lazy}\ x_1 = 8/4\ \mathsf{in}\ \mathsf{let}\ x_3 = 6\ \mathsf{in}\ \langle x_1, (\lambda x_2{:}\mathsf{int}.5 + x_2)\ (x_3 - x_3)\rangle \\ \longrightarrow & \mathsf{lazy}\ x_1 = 8/4\ \mathsf{in}\ \langle x_1, (\lambda x_2{:}\mathsf{int}.5 + x_2)\ (6 - 6)\rangle \\ \longrightarrow^* & \mathsf{lazy}\ x_1 = 8/4\ \mathsf{in}\ \langle x_1, 5\rangle \end{array}$$

Figure 2.10: Triggering

next chapter, here we consider only term variables. The strict positions for them are defined with the help of yet another set of contexts:

$$S ::= \ \_\, e \mid \_\, \tau \mid \mathsf{let}\ \langle x_1, x_2\rangle = \_\ \mathsf{in}\ e \mid \mathsf{let}\ \langle \alpha, x\rangle = \_\ \mathsf{in}\ e$$

The answer to the second part of the question, *how* we trigger the evaluation, is easy: we use the fact that we already have a strict way of opening packages and hence just turn a lazy into a let. This makes up for the reduction rule R-Trigger:

$$L_1[\mathsf{lazy}\ \langle \zeta, x\rangle = e\ \mathsf{in}\ L_2ES[x]] \quad \longrightarrow L_1[\mathsf{let}\ \langle \alpha, x\rangle = e\ \mathsf{in}\ (L_2ES[x])[\zeta := \alpha]]$$

This time the rule's side condition is that x must not be bound by L_2, which guarantees that we transform the appropriate binding. Additionally, our distinction between normal and lazy type variables requires us to rename ζ at this point.

Figure 2.10 continues the previous example (assuming an analogous trigger-rule for the simple lazy construct). The first reduction step is performed because x_4 occurs in a strict position[6] for $E = \langle x_1, (\lambda x_2{:}\mathsf{int}.5 + x_2)\ \_\rangle$ and $S = \_ - x_3$. Therefore the lazy that binds x_4 becomes a let, which causes $7 - 4$ to be computed. The result is then substituted for x_4 and the same game starts with x_3. This example also demonstrates that the results of computations no longer always are values. In fact, they now have the more general form $L[v]$.

Figures 2.11, 2.12, and 2.13 show our calculus including all the extensions we have made so far. This is the base for the next chapter. We use ξ to denote both lazy and regular type variables. $btv(L)$ is defined below. The only missing pieces now are the reduction rules for the tcase construct and the trigger-rule for lazy types. This is no coincidence, as they are interdependent.

[6]We assume that strict contexts are extended with $\_ \circ e$ and $v \circ \_$ for each integer operator $\circ$.

term variables	$x \in \mathit{Var}$
regular type variables	$\alpha \in \mathit{RTVar}$
lazy type variables	$\zeta \in \mathit{LTVar}$
terms	$e ::= x \mid \lambda x{:}\tau.e \mid e_1\ e_2 \mid \lambda\alpha{:}\kappa.e \mid e\ \tau \mid \langle e_1, e_2\rangle \mid \mathsf{let}\ \langle x_1, x_2\rangle = e_1\ \mathsf{in}\ e_2 \mid$
	$\langle \tau_1, e\rangle{:}\tau_2 \mid \mathsf{let}\ \langle \alpha, x\rangle = e_1\ \mathsf{in}\ e_2 \mid \mathsf{lazy}\ \langle \zeta, x\rangle = e_1\ \mathsf{in}\ e_2$
type variables	$\xi ::= \alpha \mid \zeta$
types	$\tau ::= \xi \mid \tau_1 \to \tau_2 \mid \tau_1 \times \tau_2 \mid \forall\alpha{:}\kappa.\tau \mid \exists\alpha{:}\kappa.\tau \mid$
	$\lambda\alpha{:}\kappa.\tau \mid \tau_1\ \tau_2 \mid \langle \tau_1, \tau_2\rangle \mid \tau.1 \mid \tau.2$
kinds	$\kappa ::= \Omega \mid \kappa_1 \to \kappa_2 \mid \kappa_1 \times \kappa_2$
values	$v ::= x \mid \lambda x{:}\tau.e \mid \lambda\alpha{:}\kappa.e \mid \langle v_1, v_2\rangle \mid \langle \tau_1, v\rangle{:}\tau_2$
evaluation contexts	$E ::= \_ \mid E\ e \mid (\lambda x{:}\tau.e)\ E \mid E\ \tau \mid \langle E, e\rangle \mid \langle v, E\rangle \mid \mathsf{let}\ \langle x_1, x_2\rangle = E\ \mathsf{in}\ e \mid$
	$\langle \tau, E\rangle{:}\tau \mid \mathsf{let}\ \langle \alpha, x\rangle = E\ \mathsf{in}\ e$
lazy contexts	$L ::= \_ \mid \mathsf{lazy}\ \langle \zeta, x\rangle = e\ \mathsf{in}\ L$
strict contexts	$S ::= \_\ e \mid \_\ \tau \mid \mathsf{let}\ \langle x_1, x_2\rangle = \_\ \mathsf{in}\ e \mid \mathsf{let}\ \langle \alpha, x\rangle = \_\ \mathsf{in}\ e$

Reduction $\boxed{e \longrightarrow e'}$

$$\begin{array}{lrcl}
\text{R-App} & LE[(\lambda x{:}\tau.e)\ v] & \longrightarrow & LE[e[x := v]] \\
\text{R-Inst} & LE[(\lambda\alpha{:}\kappa.e)\ \tau] & \longrightarrow & LE[e[\alpha := \tau]] \\
\text{R-Proj} & LE[\mathsf{let}\ \langle x_1, x_2\rangle = \langle v_1, v_2\rangle\ \mathsf{in}\ e] & \longrightarrow & LE[e[x_1 := v_1][x_2 := v_2]] \\
\text{R-Open} & LE[\mathsf{let}\ \langle \alpha, x\rangle = \langle \tau, v\rangle{:}\tau'\ \mathsf{in}\ e] & \longrightarrow & LE[e[\alpha := \tau][x := v]] \\
\text{R-Suspend} & LE[\mathsf{lazy}\ \langle \zeta, x\rangle = e_1\ \mathsf{in}\ e_2] & \longrightarrow & L[\mathsf{lazy}\ \langle \zeta, x\rangle = e_1\ \mathsf{in}\ E[e_2]] \\
 & & & (E \neq \_ \wedge \zeta \notin \mathit{ftv}(E)\ \wedge\ x \notin \mathit{fv}(E)) \\
\text{R-Trigger} & L_1[\mathsf{lazy}\ \langle \zeta, x\rangle = e\ \mathsf{in}\ L_2ES[x]] & \longrightarrow & L_1[\mathsf{let}\ \langle \alpha, x\rangle = e\ \mathsf{in}\ (L_2ES[x])[\zeta := \alpha]] \\
 & & & (x \notin \mathit{btv}(L_2))
\end{array}$$

Figure 2.11: Extended calculus (part 1 of 3)

Definition 1 (Variables Bound By Lazy Contexts)**.**

$$\begin{aligned}
\mathit{bv}(\_) &\overset{\mathit{def}}{=} \emptyset \\
\mathit{bv}(\mathsf{lazy}\ \langle \zeta, x\rangle = e\ \mathsf{in}\ L) &\overset{\mathit{def}}{=} \{x\} \cup \mathit{bv}(L)
\end{aligned}$$

$$\begin{aligned}
\mathit{btv}(\_) &\overset{\mathit{def}}{=} \emptyset \\
\mathit{btv}(\mathsf{lazy}\ \langle \zeta, x\rangle = e\ \mathsf{in}\ L) &\overset{\mathit{def}}{=} \{\zeta\} \cup \mathit{btv}(L)
\end{aligned}$$

Now we have the necessary machinery to model the combination of lazy linking and dynamic type checking. We demonstrate this by translating the Alice ML code from figure 1.1 into our calculus.

In Alice ML, a component in fact is a higher-order function that takes a ‘link’ procedure and returns a dynamic that carries a structure. The link function does all the work of acquiring a component located at a given URL, such as resolving the URL, establishing a network connection or reading a file, and unpickling some bytecode. Each component inherits it from its parent and hands it down again to the components it does import itself. We abstract away from this

Well-formedness of environments $\boxed{\Gamma \vdash \Box}$

$$\text{(E-Empty)}\ \frac{}{\cdot \vdash \Box} \qquad \text{(E-Type)}\ \frac{\Gamma \vdash \Box \qquad \alpha \notin dom(\Gamma)}{\Gamma, \alpha{:}\kappa \vdash \Box}$$

$$\text{(E-Term)}\ \frac{\Gamma \vdash \tau : \Omega \qquad x \notin dom(\Gamma)}{\Gamma, x{:}\tau \vdash \Box}$$

Well-formedness of types $\boxed{\Gamma \vdash \tau : \kappa}$

$$\text{(K-Var)}\ \frac{\Gamma \vdash \Box}{\Gamma \vdash \alpha : \Gamma(\alpha)} \qquad \text{(K-Arrow)}\ \frac{\Gamma \vdash \tau_1 : \Omega \qquad \Gamma \vdash \tau_2 : \Omega}{\Gamma \vdash \tau_1 \to \tau_2 : \Omega}$$

$$\text{(K-Times)}\ \frac{\Gamma \vdash \tau_1 : \Omega \qquad \Gamma \vdash \tau_2 : \Omega}{\Gamma \vdash \tau_1 \times \tau_2 : \Omega} \qquad \text{(K-Univ)}\ \frac{\Gamma, \alpha{:}\kappa \vdash \tau : \Omega}{\Gamma \vdash \forall \alpha{:}\kappa.\tau : \Omega}$$

$$\text{(K-Exist)}\ \frac{\Gamma, \alpha{:}\kappa \vdash \tau : \Omega}{\Gamma \vdash \exists \alpha{:}\kappa.\tau : \Omega} \qquad \text{(K-Abs)}\ \frac{\Gamma, \alpha{:}\kappa \vdash \tau : \kappa'}{\Gamma \vdash \lambda \alpha{:}\kappa.\tau : \kappa \to \kappa'}$$

$$\text{(K-App)}\ \frac{\Gamma \vdash \tau_1 : \kappa_2 \to \kappa \qquad \Gamma \vdash \tau_2 : \kappa_2}{\Gamma \vdash \tau_1\ \tau_2 : \kappa} \qquad \text{(K-Pair)}\ \frac{\Gamma \vdash \tau_1 : \kappa_1 \qquad \Gamma \vdash \tau_2 : \kappa_2}{\Gamma \vdash \langle \tau_1, \tau_2 \rangle : \kappa_1 \times \kappa_2}$$

$$\text{(K-Proj1)}\ \frac{\Gamma \vdash \tau : \kappa_1 \times \kappa_2}{\Gamma \vdash \tau.1 : \kappa_1} \qquad \text{(K-Proj2)}\ \frac{\Gamma \vdash \tau : \kappa_1 \times \kappa_2}{\Gamma \vdash \tau.2 : \kappa_2}$$

Well-formedness of terms $\boxed{\Gamma \vdash e : \tau}$

$$\text{(T-Equiv)}\ \frac{\Gamma \vdash e : \tau' \qquad \tau' \equiv \tau \qquad \Gamma \vdash \tau : \Omega}{\Gamma \vdash e : \tau} \qquad \text{(T-Var)}\ \frac{\Gamma \vdash \Box}{\Gamma \vdash x : \Gamma(x)}$$

$$\text{(T-Abs)}\ \frac{\Gamma, x{:}\tau \vdash e : \tau'}{\Gamma \vdash \lambda x{:}\tau.e : \tau \to \tau'} \qquad \text{(T-App)}\ \frac{\Gamma \vdash e_1 : \tau_2 \to \tau \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\ e_2 : \tau}$$

$$\text{(T-Gen)}\ \frac{\Gamma, \alpha{:}\kappa \vdash e : \tau}{\Gamma \vdash \lambda \alpha{:}\kappa.e : \forall \alpha{:}\kappa.\tau} \qquad \text{(T-Inst)}\ \frac{\Gamma \vdash e : \forall \alpha{:}\kappa.\tau' \qquad \Gamma \vdash \tau : \kappa}{\Gamma \vdash e\ \tau : \tau'[\alpha := \tau]}$$

$$\text{(T-Pair)}\ \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2} \qquad \text{(T-Proj)}\ \frac{\Gamma \vdash e_1 : \tau_1 \times \tau_2 \qquad \Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_2 : \tau}{\Gamma \vdash \mathsf{let}\ \langle x_1, x_2 \rangle = e_1\ \mathsf{in}\ e_2 : \tau}$$

$$\text{(T-Close)}\ \frac{\Gamma \vdash \tau : \kappa \qquad \Gamma \vdash e : \tau'[\alpha := \tau] \qquad \Gamma \vdash \exists \alpha{:}\kappa.\tau' : \Omega}{\Gamma \vdash \langle \tau, e \rangle{:}\exists \alpha{:}\kappa.\tau' : \exists \alpha{:}\kappa.\tau'}$$

$$\text{(T-Open)}\ \frac{\Gamma \vdash e_1 : \exists \alpha{:}\kappa.\tau' \qquad \Gamma, \alpha{:}\kappa, x{:}\tau' \vdash e_2 : \tau \qquad \alpha \notin ftv(\tau)}{\Gamma \vdash \mathsf{let}\ \langle \alpha, x \rangle = e_1\ \mathsf{in}\ e_2 : \tau}$$

$$\text{(T-Lazy)}\ \frac{\Gamma \vdash e_1 : \exists \alpha{:}\kappa.\tau' \qquad \Gamma, \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta] \vdash e_2 : \tau \qquad \zeta \notin ftv(\tau)}{\Gamma \vdash \mathsf{lazy}\ \langle \zeta, x \rangle = e_1\ \mathsf{in}\ e_2 : \tau}$$

Figure 2.12: Extended calculus (part 2 of 3)

Equivalence of types $\boxed{\tau \equiv \tau'}$

$$(\text{Q-R}{\small\text{EFL}})\ \frac{}{\tau \equiv \tau} \qquad (\text{Q-S}{\small\text{YMM}})\ \frac{\tau_2 \equiv \tau_1}{\tau_1 \equiv \tau_2} \qquad (\text{Q-T}{\small\text{RANS}})\ \frac{\tau_1 \equiv \tau_2 \qquad \tau_2 \equiv \tau_3}{\tau_1 \equiv \tau_3}$$

$$(\text{Q-A}{\small\text{RROW}})\ \frac{\tau_1 \equiv \tau_1' \qquad \tau_2 \equiv \tau_2'}{\tau_1 \to \tau_2 \equiv \tau_1' \to \tau_2'} \qquad (\text{Q-T}{\small\text{IMES}})\ \frac{\tau_1 \equiv \tau_1' \qquad \tau_2 \equiv \tau_2'}{\tau_1 \times \tau_2 \equiv \tau_1' \times \tau_2'}$$

$$(\text{Q-U}{\small\text{NIV}})\ \frac{\tau_1 \equiv \tau_2}{\forall\alpha{:}\kappa.\tau_1 \equiv \forall\alpha{:}\kappa.\tau_2} \qquad (\text{Q-E}{\small\text{XIST}})\ \frac{\tau_1 \equiv \tau_2}{\exists\alpha{:}\kappa.\tau_1 \equiv \exists\alpha{:}\kappa.\tau_2}$$

$$(\text{Q-A}{\small\text{BS}})\ \frac{\tau_1 \equiv \tau_2}{\lambda\alpha{:}\kappa.\tau_1 \equiv \lambda\alpha{:}\kappa.\tau_2} \qquad (\text{Q-A}{\small\text{PP}})\ \frac{\tau_1 \equiv \tau_2 \qquad \tau_1' \equiv \tau_2'}{\tau_1\ \tau_2 \equiv \tau_1'\ \tau_2'}$$

$$(\text{Q-B}{\small\text{ETA}})\ \frac{}{(\lambda\alpha{:}\kappa.\tau_1)\ \tau_2 \equiv \tau_1[\alpha := \tau_2]} \qquad (\text{Q-P}{\small\text{AIR}})\ \frac{\tau_1 \equiv \tau_1' \qquad \tau_2 \equiv \tau_2'}{\langle\tau_1, \tau_2\rangle \equiv \langle\tau_1', \tau_2'\rangle}$$

$$(\text{Q-P}{\small\text{ROJ1A}})\ \frac{\tau \equiv \tau'}{\tau.1 \equiv \tau'.1} \qquad (\text{Q-P}{\small\text{ROJ1B}})\ \frac{}{\langle\tau_1, \tau_2\rangle.1 \equiv \tau_1}$$

$$(\text{Q-P}{\small\text{ROJ2A}})\ \frac{\tau \equiv \tau'}{\tau.2 \equiv \tau'.2} \qquad (\text{Q-P}{\small\text{ROJ2B}})\ \frac{}{\langle\tau_1, \tau_2\rangle.2 \equiv \tau_2}$$

Figure 2.13: Extended calculus (part 3 of 3)

and simply model components as lazy dynamics. The actual loading—the process that we want to happen at the latest possible time—then corresponds to the opening of the corresponding package. Let us summarize our model:

concept	**representation**
dynamics	packages of type $\exists\alpha{:}\Omega.\alpha$
dynamic type checking	typecase
modules	nested pairs within a package whose existential type reflects the module's signature
components	lazy dynamics carrying a module

The translation is shown in figure 2.14. In order to improve readability, we take the freedom to omit the annotated existential type of modules since it already appears as the type component of the surrounding dynamic. Furthermore we leave out kinds because we are not dealing with any type constructors here. The three lazy bindings at the top level roughly resemble the table of Alice ML's component manager that keeps track of imported components [22, 20].

2.6 Properties

In this section we present several basic properties of the calculus shown in figures 2.11, 2.12, and 2.13. None of them refers to the operational semantics so it does no harm that there are still some reduction rules missing. We only show a few selected parts of the more interesting proofs here. The complete proofs can be found in the appendix.

Proposition 1 (Substitution). *If $\xi_1 \neq \xi_2$ and $\xi_1 \notin ftv(\tau_2)$, then, for all τ and τ_1, $\tau[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$.*

Proof. By induction on the structure of τ. □

```
lazy $\langle \alpha_A, x_A \rangle$ = $\langle \exists t.t, \langle \mathsf{int}, 42 \rangle \rangle$ in
lazy $\langle \alpha_B, x_B \rangle$ =
  lazy $\langle t, x \rangle$ = tcase $x_A$:$\alpha_A$ of $x'_A$:$\exists t.t$ then $x'_A$ else $\bot$ in
  $\langle \exists u.u \times u, \langle t, \langle x, x \rangle \rangle \rangle$ in
lazy $\langle \alpha_C, x_C \rangle$ =
  lazy $\langle u, y \rangle$ =
    tcase $x_B$:$\alpha_B$ of $x'_B$:$\exists u.u \times u$ then
      let $\langle u, y \rangle$ = $x'_B$ in tcase $y$:$u \times u$ of $y'$:int $\times$ int then $\langle u, y \rangle$:$\exists u.$int $\times$ int else $\bot$
    else $\bot$ in
  $\langle \exists v.$int$, \langle$int$,$ let $\langle z, \_ \rangle$ = $y'$ in $z \rangle \rangle$ in
```

Figure 2.14: Translation of the Alice ML code from figure 1.1

Proposition 2 (Environment Validity)**.**

1. *If* $\Gamma \vdash \tau : \kappa$*, then* $\Gamma \vdash \Box$*.*
2. *If* $\Gamma \vdash e : \tau$*, then* $\Gamma \vdash \Box$*.*

Proof. By induction on the derivation. □

Proposition 3 (Subenvironment Validity)**.** *If* $\Gamma_1, \Gamma_2 \vdash \Box$*, then* $\Gamma_1 \vdash \Box$*.*

Proof. By induction on the structure of Γ_2. □

Definition 2 (Environment Inclusion)**.**

$$\Gamma \subseteq \Gamma' \overset{def}{\iff} dom(\Gamma) \subseteq dom(\Gamma') \wedge \forall x \in dom(\Gamma) : \Gamma(x) \equiv \Gamma'(x) \wedge \forall \xi \in dom(\Gamma) : \Gamma(\xi) = \Gamma'(\xi)$$

Proposition 4 (Variable Containment)**.**

1. *If* $\Gamma \vdash \tau : \kappa$*, then* $ftv(\tau) \subseteq dom(\Gamma)$*.*
2. *If* $\Gamma \vdash e : \tau$*, then* $fv(e) \cup ftv(\tau) \subseteq dom(\Gamma)$*.*

Proof. By induction on the derivation.

2. Let $\Gamma \vdash e : \tau$.

   - Case T-Case: $e = \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau'_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$
     (a) By inversion:
        i. $\Gamma \vdash e_0 : \tau_0$
        ii. $\Gamma, x{:}\tau'_0 \vdash e_1 : \tau$
        iii. $\Gamma \vdash e_2 : \tau$
     (b) By (a) and induction:
        i. $fv(e_0) \subseteq dom(\Gamma)$
        ii. $fv(e_1) \subseteq dom(\Gamma, x{:}\tau'_0)$
        iii. $fv(e_2) \cup ftv(\tau) \subseteq dom(\Gamma)$
     (c) By (a-ii) and Environment Validity: $\Gamma, x{:}\tau'_0 \vdash \Box$
     (d) By (c) and inversion of E-Term: $x \notin dom(\Gamma)$
     (e) By (b-ii) and (d): $fv(e_1) - \{x\} \subseteq dom(\Gamma)$
     (f) By (b-i), (e), and (b-iii): $fv(e_0) \cup (fv(e_1) - \{x\}) \cup fv(e_2) \subseteq dom(\Gamma)$

(g) By definition: $fv(\textsf{tcase } e_0{:}\tau_0 \textsf{ of } x{:}\tau_0' \textsf{ then } e_1 \textsf{ else } e_2) = fv(e_0) \cup (fv(e_1) - \{x\}) \cup fv(e_2)$

(h) By (g) and (f): $fv(\textsf{tcase } e_0{:}\tau_0 \textsf{ of } x{:}\tau_0' \textsf{ then } e_1 \textsf{ else } e_2) \subset dom(\Gamma)$

(i) By (h) and (b-iii): $fv(\textsf{tcase } e_0{:}\tau_0 \textsf{ of } x{:}\tau_0' \textsf{ then } e_1 \textsf{ else } e_2) \cup ftv(\tau) \subset dom(\Gamma)$

□

Proposition 5 (Strengthening)**.**

1. *If* $\Gamma \vdash \tau : \kappa$ *and* $\Gamma' \subseteq \Gamma$ *with* $\Gamma' \vdash \Box$ *and* $ftv(\tau) \subseteq dom(\Gamma')$*, then* $\Gamma' \vdash \tau : \kappa$*.*

2. *If* $\Gamma_1, x{:}\tau, \Gamma_2 \vdash \Box$*, then* $\Gamma_1, \Gamma_2 \vdash \Box$*.*

Proof. (1) by induction on the derivation of $\Gamma \vdash \tau : \kappa$. (2) by induction on the structure of Γ_2. □

Proposition 6 (Weakening)**.**

1. *If* $\Gamma \vdash \tau : \kappa$ *and* $\Gamma \subseteq \Gamma'$ *where* $\Gamma' \vdash \Box$*, then* $\Gamma' \vdash \tau : \kappa$*.*

2. *If* $\Gamma \vdash e : \tau$ *and* $\Gamma \subseteq \Gamma'$ *where* $\Gamma' \vdash \Box$*, then* $\Gamma' \vdash e : \tau$*.*

Proof. (1) by induction on the derivation of $\Gamma \vdash \tau : \kappa$. (2) by induction on the derivation of $\Gamma \vdash e : \tau$. □

Proposition 7 (Type Substitution)**.**

1. *If* $\Gamma_1 \vdash \tau' : \kappa'$ *and* $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \Box$*, then* $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \Box$*.*

2. *If* $\Gamma_1 \vdash \tau' : \kappa'$ *and* $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \tau : \kappa$*, then* $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau[\xi := \tau'] : \kappa$*.*

3. *If* $\Gamma_1 \vdash \tau' : \kappa'$ *and* $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash e : \tau$*, then* $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e[\xi := \tau'] : \tau[\xi := \tau']$*.*

4. *If* $\tau \equiv \tau''$*, then* $\tau[\xi := \tau'] \equiv \tau''[\xi := \tau']$*.*

Proof. (1), (2), and (3) by simultaneous induction on the kinding derivations. (4) by induction on the derivation. □

Proposition 8 (Term Substitution)**.** *If* $\Gamma \vdash e' : \tau'$ *and* $\Gamma, x{:}\tau', \Gamma' \vdash e : \tau$*, then* $\Gamma, \Gamma' \vdash e[x := e'] : \tau$*.*

Proof. By induction on the derivation of $\Gamma_1, x{:}\tau', \Gamma_2 \vdash e : \tau$. □

Proposition 9 (Validity)**.** *If* $\Gamma \vdash e : \tau$*, then* $\Gamma \vdash \tau : \Omega$*.*

Proof. By induction on the derivation. In the case T-Close it becomes apparent why we are using the additional premise $\Gamma \vdash \exists\alpha{:}\kappa.\tau' : \Omega$ in this rule. Otherwise we would have to show that $\Gamma \vdash \tau[\alpha := \tau'] : \Omega$ with $\Gamma \vdash \tau' : \kappa$ implies $\Gamma, \alpha{:}\kappa \vdash \tau : \Omega$, i.e., parts (1) and (2) of proposition 7 (Type Substitution) would have to be equivalences.

- Case T-Close: $e = \langle \tau_1, e' \rangle{:}\tau$ where $\tau = \exists\alpha{:}\kappa.\tau_2$

  1. By inversion: $\Gamma \vdash \exists\alpha{:}\kappa.\tau_2 : \Omega$

- Case T-Open: $e = \textsf{let } \langle \alpha, x \rangle = e_1 \textsf{ in } e_2$

  1. By inversion: $\Gamma, \alpha{:}\kappa, x{:}\tau' \vdash e_2 : \tau$
  2. By (1) and induction: $\Gamma, \alpha{:}\kappa, x{:}\tau' \vdash \tau : \Omega$
  3. By assumption and Environment Validity: $\Gamma \vdash \Box$

4. By assumption and Variable Containment: $ftv(\tau) \subseteq dom(\Gamma)$
5. By (2), (3), (4), and Strengthening: $\Gamma \vdash \tau : \Omega$

□

Given some typing derivation $\Gamma \vdash e : \tau$ we are not able to tell which rule has been applied in the last step of that derivation. There are always two possibilities: the canonical one (e.g., T-Abs if e is an abstraction) and the equivalence rule T-Equiv. However, we do know that the derivation tree must end with a use of the canonical rule followed by arbitrary many uses of T-Equiv. We therefore formulate an inversion lemma that abstracts away the uses of the equivalence rule and allows us to make use of the premise of the respective canonical rule. Other judgements, like that defined by the kinding relation, do not require such a lemma since there is only one rule per syntactic form.

Lemma 1 (Typing Inversion)**.**

- *If* $\Gamma \vdash x : \tau$*, then* $\tau \equiv \Gamma(x)$ *and* $\Gamma \vdash \Box$*.*
- *If* $\Gamma \vdash \lambda x{:}\tau.e : \tau'$ *and* $x \notin dom(\Gamma)$*, then* $\Gamma, x{:}\tau \vdash e : \tau''$ *and* $\tau' \equiv \tau \to \tau''$*.*
- *If* $\Gamma \vdash e_1\ e_2 : \tau$*, then* $\Gamma \vdash e_1 : \tau' \to \tau$ *and* $\Gamma \vdash e_2 : \tau'$*.*
- *If* $\Gamma \vdash \lambda\alpha{:}\kappa.e : \tau$ *and* $\alpha \notin dom(\Gamma)$*, then* $\Gamma, \alpha{:}\kappa \vdash e : \tau'$ *and* $\tau \equiv \forall\alpha{:}\kappa.\tau'$*.*
- *If* $\Gamma \vdash e\ \tau' : \tau$*, then* $\Gamma \vdash e : \forall\alpha{:}\kappa.\tau_1$ *where* $\alpha \notin dom(\Gamma)$ *and* $\tau \equiv \tau_1[\alpha := \tau']$ *where* $\Gamma \vdash \tau' : \kappa$*.*
- *If* $\Gamma \vdash \langle e_1, e_2\rangle : \tau$*, then* $\Gamma \vdash e_1 : \tau_1$ *and* $\Gamma \vdash e_2 : \tau_2$ *where* $\tau \equiv \tau_1 \times \tau_2$*.*
- *If* $\Gamma \vdash \langle \tau_1, e\rangle{:}\tau_2 : \tau$*, then* $\Gamma \vdash \tau_1 : \kappa$ *and* $\Gamma \vdash e : \tau_3[\alpha := \tau_1]$ *where* $\tau_2 = \exists\alpha{:}\kappa.\tau_3$ *and* $\tau \equiv \tau_2$ *and* $\Gamma \vdash \tau_2 : \Omega$*.*
- *If* $\Gamma \vdash \mathsf{let}\ \langle x_1, x_2\rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ e_2 : \tau$ *and* $\{x_1, x_2\} \cap dom(\Gamma) = \emptyset$*, then* $\Gamma \vdash e_1 : \tau_1 \times \tau_2$ *and* $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_2 : \tau$*.*
- *If* $\Gamma \vdash \mathsf{let}\ \langle \alpha, x\rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ e_2 : \tau$ *and* $\{\alpha, x\} \cap dom(\Gamma) = \emptyset$*, then* $\Gamma \vdash e_1 : \exists\alpha{:}\kappa.\tau'$ *and* $\Gamma, \alpha{:}\kappa, x{:}\tau' \vdash e_2 : \tau''$ *where* $\alpha \notin ftv(\tau'')$ *and* $\tau'' \equiv \tau$*.*
- *If* $\Gamma \vdash \mathsf{lazy}\ \langle \zeta, x\rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ e_2 : \tau$ *and* $\{\zeta, x\} \cap dom(\Gamma) = \emptyset$*, then* $\Gamma \vdash e_1 : \exists\alpha{:}\kappa.\tau'$ *where* $\alpha \notin dom(\Gamma)$ *and* $\Gamma, \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta] \vdash e_2 : \tau''$ *where* $\zeta \notin ftv(\tau'')$ *and* $\tau'' \equiv \tau$*.*
- *If* $\Gamma \vdash \mathsf{tcase}\ e{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau$ *and* $x \notin dom(\Gamma)$*, then* $\Gamma \vdash e : \tau_0$ *and* $\Gamma, x{:}\tau_0' \vdash e_1 : \tau$ *and* $\Gamma \vdash e_2 : \tau$*.*

Proof. By induction on the corresponding derivation. □

Proving lemma 4 (Type Equivalence Inversion) like the Typing Inversion lemma directly by induction on the derivation is not possible due to the transitivity rule. For this reason, we base the proof on the *parallel reduction* relation $\Rrightarrow$ as defined in [18]. We assume that this directed version of the type equivalence relation is trivially extended to match our system.

Lemma 2 (Parallel Reduction)**.**

- $\tau_1 \equiv \tau_2$ *iff* $\tau_1 \Lleftarrow\!\!\Rrightarrow^* \tau_2$
- *If* $\tau_1 \equiv \tau_2$*, then there is some* τ *such that* $\tau_1 \Rrightarrow^* \tau$ *and* $\tau_2 \Rrightarrow^* \tau$*.*

Proof. See [18]. □

Lemma 3 (Preservation of Shapes Under Reduction)**.**

1. *If* $\xi \Rrightarrow^* \tau'$*, then* $\tau' = \xi$.
2. *If* $\tau_1 \to \tau_2 \Rrightarrow^* \tau'$*, then* $\tau' = \tau_1' \to \tau_2'$ *with* $\tau_1 \Rrightarrow^* \tau_1'$ *and* $\tau_2 \Rrightarrow^* \tau_2'$.
3. *If* $\tau_1 \times \tau_2 \Rrightarrow^* \tau'$*, then* $\tau' = \tau_1' \times \tau_2'$ *with* $\tau_1 \Rrightarrow^* \tau_1'$ *and* $\tau_2 \Rrightarrow^* \tau_2'$.
4. *If* $\langle \tau_1, \tau_2 \rangle \Rrightarrow^* \tau'$*, then* $\tau' = \langle \tau_1', \tau_2' \rangle$ *with* $\tau_1 \Rrightarrow^* \tau_1'$ *and* $\tau_2 \Rrightarrow^* \tau_2'$.
5. *If* $\forall \alpha{:}\kappa.\tau_1 \Rrightarrow^* \tau'$*, then* $\tau' = \forall \alpha{:}\kappa.\tau_1'$ *with* $\tau_1 \Rrightarrow^* \tau_1'$.
6. *If* $\exists \alpha{:}\kappa.\tau_1 \Rrightarrow^* \tau'$*, then* $\tau' = \exists \alpha{:}\kappa.\tau_1'$ *with* $\tau_1 \Rrightarrow^* \tau_1'$.
7. *If* $\lambda \alpha{:}\kappa.\tau_1 \Rrightarrow^* \tau'$*, then* $\tau' = \lambda \alpha{:}\kappa.\tau_1'$ *with* $\tau_1 \Rrightarrow^* \tau_1'$.

Proof. See [18]. □

Lemma 4 (Type Equivalence Inversion)**.**

- *If* $\exists \alpha{:}\kappa_1.\tau_1 \equiv \exists \alpha{:}\kappa_2.\tau_2$*, then* $\kappa_1 = \kappa_2$ *and* $\tau_1 \equiv \tau_2$.
- *If* $\tau_1 \to \tau_2 \equiv \tau_1' \to \tau_2'$*, then* $\tau_1 \equiv \tau_1'$ *and* $\tau_2 \equiv \tau_2'$.
- *If* $\forall \alpha{:}\kappa_1.\tau_1 \equiv \forall \alpha{:}\kappa_2.\tau_2$*, then* $\kappa_1 = \kappa_2$ *and* $\tau_1 \equiv \tau_2$.
- *If* $\tau_1 \times \tau_2 \equiv \tau_1' \times \tau_2'$*, then* $\tau_1 \equiv \tau_1'$ *and* $\tau_2 \equiv \tau_2'$.
- *If* $\langle \tau_1, \tau_2 \rangle \equiv \langle \tau_1', \tau_2' \rangle$*, then* $\tau_1 \equiv \tau_1'$ *and* $\tau_2 \equiv \tau_2'$.

Proof. Follows from lemma 3 (Preservation of Shapes Under Reduction). □

Lemma 5 (Shape Consistency)**.**

- *If* $\tau \equiv \tau_1 \to \tau_2$*, then* $\tau = \tau_1' \to \tau_2'$ *or* $\tau = \tau_1'\ \tau_2'$ *or* $\tau = \tau'.1$ *or* $\tau = \tau'.2$.
- *If* $\tau \equiv \tau_1 \times \tau_2$*, then* $\tau = \tau_1' \times \tau_2'$ *or* $\tau = \tau_1'\ \tau_2'$ *or* $\tau = \tau'.1$ *or* $\tau = \tau'.2$.
- *If* $\tau \equiv \forall \alpha{:}\kappa.\tau_1$*, then* $\tau = \forall \alpha{:}\kappa.\tau_1'$ *or* $\tau = \tau_1'\ \tau_2'$ *or* $\tau = \tau'.1$ *or* $\tau = \tau'.2$.
- *If* $\tau \equiv \exists \alpha{:}\kappa.\tau_1$*, then* $\tau = \exists \alpha{:}\kappa.\tau_1'$ *or* $\tau = \tau_1'\ \tau_2'$ *or* $\tau = \tau'.1$ *or* $\tau = \tau'.2$.

Proof. By contradiction, using lemma 2 (Parallel Reduction) and 3 (Preservation of Shapes Under Reduction). □

Proposition 10 (Canonical Values)**.** *Let* $\Gamma \vdash v : \tau$ *where* v *is not a variable.*

- *If* $\tau \equiv \tau_1 \to \tau_2$*, then* $v = \lambda x{:}\tau_1'.e$.
- *If* $\tau \equiv \tau_1 \times \tau_2$*, then* $v = \langle v_1, v_2 \rangle$.
- *If* $\tau \equiv \forall \alpha{:}\kappa.\tau_1$*, then* $v = \lambda \alpha{:}\kappa.e$.
- *If* $\tau \equiv \exists \alpha{:}\kappa.\tau_1$*, then* $v = \langle \tau_2, v' \rangle{:}\tau'$.

Proof. By induction on the typing derivation. □

Proposition 11 (Uniqueness of Kinds)**.** *If* $\Gamma \vdash \tau : \kappa$ *and* $\Gamma \vdash \tau : \kappa'$*, then* $\kappa = \kappa'$.

Proof. By induction on the structure of τ. □

Definition 3 (Equivalence of Environments)**.**

$$\Gamma \equiv \Gamma' \overset{def}{\iff} dom(\Gamma) = dom(\Gamma') \wedge \forall x \in dom(\Gamma) : \Gamma(x) \equiv \Gamma'(x) \wedge \forall \xi \in dom(\Gamma) : \Gamma(\xi) = \Gamma'(\xi)$$

Proposition 12 (Uniqueness of Types)**.** *If* $\Gamma \vdash e : \tau$ *and* $\Gamma' \vdash e : \tau'$ *with* $\Gamma \equiv \Gamma'$*, then* $\tau \equiv \tau'$*.*

Proof. By induction on the structure of e. ☐

Lemma 6 (Equivalent Environments)**.**

1. *If* $\Gamma \vdash \tau : \kappa$ *and* $\Gamma \equiv \Gamma'$ *where* $\Gamma' \vdash \Box$*, then* $\Gamma' \vdash \tau : \kappa$*.*
2. *If* $\Gamma \vdash e : \tau$ *and* $\Gamma \equiv \Gamma'$ *where* $\Gamma' \vdash \Box$*, then* $\Gamma' \vdash e : \tau$*.*

Proof. (1) by induction on the kinding derivation. (2) by induction on the typing derivation. ☐

We now introduce another judgement. Informally, $\Gamma \vdash L : \Gamma'$ holds if Γ' is the environment that is generated by the lazy bindings in L (under the assumptions in Γ).

Definition 4 (Typing Judgement for Lazy Contexts)**.** $\boxed{\Gamma \vdash L : \Gamma'}$

$$(\text{L-E{\scriptsize MPTY}})\ \frac{\Gamma \vdash \Box}{\Gamma \vdash \_ : \cdot}$$

$$(\text{L-L{\scriptsize AZY}})\ \frac{\Gamma \vdash e : \exists\alpha{:}\kappa.\tau \qquad \Gamma' = \zeta{:}\kappa, x{:}\tau[\alpha := \zeta], \Gamma'' \qquad \Gamma, \zeta{:}\kappa, x{:}\tau[\alpha := \zeta] \vdash L' : \Gamma''}{\Gamma \vdash \mathsf{lazy}\ \langle \zeta, x \rangle\ \mathsf{=}\ e\ \mathsf{in}\ L' : \Gamma'}$$

Lemma 7 (Context Elimination)**.**

1. *If* $\Gamma \vdash E[e] : \tau$*, then* $\Gamma \vdash e : \tau'$*.*
2. *If* $\Gamma \vdash LE[e] : \tau$*, then* $\Gamma \vdash L[e] : \tau'$*.*
3. *If* $\Gamma \vdash LE[e] : \tau$*, then* $\Gamma, \Gamma' \vdash e : \tau'$ *with* $\Gamma \vdash L : \Gamma'$*.*

Proof. (1) by structural induction on E. (2) and (3) by structural induction on L.

3. Let $\Gamma \vdash LE[e] : \tau$.
   - Case $L = \_$:
     (a) By assumption: $\Gamma \vdash E[e] : \tau$
     (b) By (a) and (1): $\Gamma \vdash e : \tau'$
     (c) By assumption and Environment Validity: $\Gamma \vdash \Box$
     (d) Let $\Gamma' = \cdot$.
     (e) By (c), (d), and L-Empty: $\Gamma, \Gamma' = \Gamma$ and $\Gamma \vdash L : \Gamma'$
     (f) By (b) and (e): $\Gamma, \Gamma' \vdash e : \tau'$ and $\Gamma \vdash L : \Gamma'$
   - Case $L = \mathsf{lazy}\ \langle \zeta, x \rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ L'$:
     (a) By assumption: $\Gamma \vdash \mathsf{lazy}\ \langle \zeta, x \rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ L'E[e] : \tau$
     (b) By (a) and Typing Inversion: $\Gamma \vdash e_1 : \exists\alpha{:}\kappa.\tau_1$ and $\Gamma, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash L'E[e] : \tau''$ where $\tau'' \equiv \tau$
     (c) By (b) and induction: $\Gamma, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta], \Gamma'' \vdash e : \tau'$ where $\Gamma, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash L' : \Gamma''$
     (d) By (b), (c), and L-Lazy: $\Gamma, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta], \Gamma'' \vdash e : \tau'$ and $\Gamma \vdash L : \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta], \Gamma''$

☐

Lemma 8 (Exchange)**.**

1. *If* $\Gamma \vdash E[e] : \tau$ *and* $\Gamma \vdash e : \tau'$ *and* $\Gamma, \Gamma' \vdash e' : \tau'$*, then* $\Gamma, \Gamma' \vdash E[e'] : \tau$*.*

2. *If* $\Gamma \vdash LE[e] : \tau$ *and* $\Gamma, \Gamma' \vdash e : \tau'$ *as well as* $\Gamma, \Gamma' \vdash e' : \tau'$ *and* $\Gamma \vdash L : \Gamma'$, *then* $\Gamma \vdash LE[e'] : \tau$.

Proof. (1) by structural induction on E, using proposition 12 (Uniqueness of Types). (2) by structural induction on L, using (1) and lemma 6 (Equivalent Environments).

1. Let $\Gamma \vdash E[e] : \tau$, $\Gamma \vdash e : \tau'$, and $\Gamma, \Gamma' \vdash e' : \tau'$.
   - Case $E = \mathsf{let}\ \langle x_1, x_2 \rangle = E'\ \mathsf{in}\ e''$ (w.l.o.g. $\{x_1, x_2\} \cap dom(\Gamma, \Gamma') = \emptyset$):
     (a) By assumption: $\Gamma \vdash \mathsf{let}\ \langle x_1, x_2 \rangle = E'[e]\ \mathsf{in}\ e'' : \tau$
     (b) By (a) and Typing Inversion: $\Gamma \vdash E'[e] : \tau_1 \times \tau_2$ and $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e'' : \tau$
     (c) By (b), assumption, and induction: $\Gamma, \Gamma' \vdash E'[e'] : \tau_1 \times \tau_2$
     (d) By (c) and Validity: $\Gamma, \Gamma' \vdash \tau_1 \times \tau_2 : \Omega$
     (e) By (d) and inversion of K-Times: $\Gamma, \Gamma' \vdash \tau_1 : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2 : \Omega$
     (f) By (e) and E-Term: $\Gamma, \Gamma', x_1{:}\tau_1 \vdash \Box$
     (g) By (e), (f), and Weakening: $\Gamma, \Gamma', x_1{:}\tau_1 \vdash \tau_2 : \Omega$
     (h) By (g) and E-Term: $\Gamma, \Gamma', x_1{:}\tau_1, x_2{:}\tau_2 \vdash \Box$
     (i) By (b), (h), and Weakening: $\Gamma, \Gamma', x_1{:}\tau_1, x_2{:}\tau_2 \vdash e'' : \tau$
     (j) By (c), (i), and T-Proj: $\Gamma, \Gamma' \vdash \mathsf{let}\ \langle x_1, x_2 \rangle = E'[e']\ \mathsf{in}\ e'' : \tau$
     (k) By (j): $\Gamma, \Gamma' \vdash E[e'] : \tau$

□

Lemma 9 (Lazy Term Variables)**.** *If* $\Gamma \vdash LE[x] : \tau$ *and* $x \notin dom(\Gamma)$, *then* $L = L_1[\mathsf{lazy}\ \langle \zeta, x \rangle = e\ \mathsf{in}\ L_2]$ *where* $x \notin bv(L_2)$.

Proof. By structural induction on L. □

Chapter 3

Reduction strategies for the type level

A straightforward approach is to associate two reduction rules with tcase: one for the case of the two types that are compared being equal according to the equivalence relation, and one for the case of them being different.

$$\begin{array}{lll} \text{R-C}\textsc{ase}1 & LE[\mathsf{tcase}\ v{:}\tau\ \mathsf{of}\ x{:}\tau'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2] \longrightarrow LE[e_1[x := v]] & (\tau \equiv \tau') \\ \text{R-C}\textsc{ase}2 & LE[\mathsf{tcase}\ v{:}\tau\ \mathsf{of}\ x{:}\tau'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2] \longrightarrow LE[e_2] & (\tau \not\equiv \tau') \end{array}$$

This works fine as long as no laziness is involved:

$$\mathsf{let}\ \langle \alpha, x\rangle\ \mathsf{=}\ \langle \mathsf{int}, 5\rangle{:}\exists\alpha{:}\Omega.\alpha\ \mathsf{in}\ \mathsf{tcase}\ x{:}\alpha\ \mathsf{of}\ x'{:}\mathsf{int}\ \mathsf{then}\ 1\ \mathsf{else}\ 0$$

The first reduction rule that can be applied is R-Open, with the effect that α is replaced with int. So, at the time the reduction encounters the tcase expression, int is compared with int—hence R-Case1 can be applied. But now consider the lazy version of this example:

$$\mathsf{lazy}\ \langle \zeta, x\rangle\ \mathsf{=}\ \langle \mathsf{int}, 5\rangle{:}\exists\alpha{:}\Omega.\alpha\ \mathsf{in}\ \mathsf{tcase}\ x{:}\zeta\ \mathsf{of}\ x'{:}\mathsf{int}\ \mathsf{then}\ 1\ \mathsf{else}\ 0$$

This time no reduction rule can be applied before the tcase is evaluated. This means that ζ and int have to be compared. Because they are different according to the definitional equivalence relation, this leads to the application of R-Case2—certainly not what we want.

What we need is a trigger-rule for lazy type variables, analogous to R-Trigger. Since the tcase is the only construct in our language whose semantics depends on the denotation of types, it is also the only place where a lazy type variable may occur in a strict position. A naive approach is therefore to say that a strict position is anywhere within one of the two type expressions of the tcase construct. This can be formalized by a context that determines the order in which the types are traversed looking for lazy type variables (it can be considered as the strict context for lazy types)[1].

However, we want to make the equivalence checking explicit. That is, instead of using the definitional type equivalence relation, we want to give an algorithm that can be implemented. The standard approach for this purpose is known as *normalize-and-compare* [8]: the types are reduced to some normal form and then compared (by checking for syntactic rather than semantic equality). We still need a second trigger-rule, since these normal forms obviously must not contain any lazy variables (see above). Therefore, the type-level reduction has to take care of their elimination and consequently, the degree of laziness depends on the chosen strategy.

[1]That would also require additional syntax to denote types that do not contain any lazy variables.

type reduction contexts	$T ::=$	$\_ \mid T \to \tau \mid \nu \to T \mid T \times \tau \mid \nu \times T \mid \forall\alpha{:}\kappa.T \mid \exists\alpha{:}\kappa.T \mid \lambda\alpha{:}\kappa.T \mid$ $T\ \tau \mid \nu\ T \mid \langle T, \tau\rangle \mid \langle \nu, T\rangle \mid T.1 \mid T.2$
tcase contexts	$C ::=$	$\mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\tau\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mid \mathsf{tcase}\ v{:}\nu\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$

RT-App	$LECT[(\lambda\alpha{:}\kappa.\nu)\ \nu'] \longrightarrow LECT[\nu[\alpha := \nu']]$
RT-Proj1	$LECT[\langle\nu_1, \nu_2\rangle.1] \longrightarrow LECT[\nu_1]$
RT-Proj2	$LECT[\langle\nu_1, \nu_2\rangle.2] \longrightarrow LECT[\nu_2]$
RT-Trigger	$L_1[\mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ L_2ECT[\zeta]] \longrightarrow L_1[\mathsf{let}\ \langle\alpha, x\rangle = e_1\ \mathsf{in}\ (L_2ECT[\zeta])[\zeta := \alpha]]$ $(\zeta \notin btv(L_2))$

Figure 3.1: Applicative order reduction to normal form

In the remainder of this chapter we present two such reductions strategies for type expressions of the tcase construct. Section 3.1 is concerned with a naive strategy that eventually eliminates all occuring lazy type variables by triggering the corresponding packages. We give a complete proof of safety properties. Section 3.2 presents a more sophisticated call-by-name strategy that involves weak head reduction and performs R-Case2 as soon as it becomes clear that the compared types are of different shape and hence cannot be equal. Again, safety proofs are included.

3.1 Applicative order reduction to normal form

In order to achieve the desired property that two types in normal form are equal if and only if they are syntactically equal, we have to reduce below binders. That means that e.g. an abstraction only is in normal form if its body is so. As a consequence of this we have to treat regular variables as normal forms. On the other hand, lazy type variables must of course not be considered normal forms (this is the reason why we did introduce the different syntactical forms for type variables). We therefore define normal forms as follows:

normal forms	$\nu ::= p \mid \nu_1 \to \nu_2 \mid \nu_1 \times \nu_2 \mid \forall\alpha{:}\kappa.\nu \mid \exists\alpha{:}\kappa.\nu \mid \lambda\alpha{:}\kappa.\nu \mid \langle\nu_1, \nu_2\rangle$
normal paths	$p ::= \alpha \mid p\ \nu \mid p.1 \mid p.2$

Consequently, the main tcase reduction rules are:

R-Case1	$LE[\mathsf{tcase}\ v{:}\nu\ \mathsf{of}\ x{:}\nu\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2] \longrightarrow LE[e_1[x := v]]$	
R-Case2	$LE[\mathsf{tcase}\ v{:}\nu\ \mathsf{of}\ x{:}\nu'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2] \longrightarrow LE[e_2]$	$(\nu \neq \nu')$

The normal forms are computed via an applicative order reduction, i.e., via a call-by-value strategy that reduces below binders (the leftmost innermost redex is reduced first) [23]. It is shown in figure 3.1. Except for RT-Trigger, all rules are derived directly from the type equivalence relation.

An example in the form of a sequence of reduction steps is given in figure 3.2. It is interesting and typical of our system how type- and term-level reduction interleave. First, type-level reduction encounters the lazy variable and hence triggers the evaluation of the corresponding package (RT-Trigger for $T = \_\ \mathsf{int}$). After it has been opened using R-Open, the type reduction can continue by applying RT-App. Since this turns the left type into the same normal form that the right type already is in, it's then again the turn of term-level reduction (R-Case1).

$$
\begin{array}{ll}
 & \mathsf{lazy}\ \langle \zeta, x\rangle = \langle \lambda\alpha'{:}\Omega.\alpha' \times \alpha', \langle 2+5, 7-4\rangle\rangle{:}\exists\alpha{:}\Omega \to \Omega.\alpha\ \mathsf{int}\ \mathsf{in} \\
 & \mathsf{tcase}\ x{:}\zeta\ \mathsf{int}\ \mathsf{of}\ x'{:}\mathsf{int} \times \mathsf{int}\ \mathsf{then}\ 1\ \mathsf{else}\ 0 \\
\longrightarrow & \mathsf{let}\ \langle \alpha, x\rangle = \langle \lambda\alpha'{:}\Omega.\alpha' \times \alpha', \langle 2+5, 7-4\rangle\rangle{:}\exists\alpha{:}\Omega \to \Omega.\alpha\ \mathsf{int}\ \mathsf{in} \\
 & \mathsf{tcase}\ x{:}\alpha\ \mathsf{int}\ \mathsf{of}\ x'{:}\mathsf{int} \times \mathsf{int}\ \mathsf{then}\ 1\ \mathsf{else}\ 0 \\
\longrightarrow^* & \mathsf{let}\ \langle \alpha, x\rangle = \langle \lambda\alpha'{:}\Omega.\alpha' \times \alpha', \langle 7, 3\rangle\rangle{:}\exists\alpha{:}\Omega \to \Omega.\alpha\ \mathsf{int}\ \mathsf{in} \\
 & \mathsf{tcase}\ x{:}\alpha\ \mathsf{int}\ \mathsf{of}\ x'{:}\mathsf{int} \times \mathsf{int}\ \mathsf{then}\ 1\ \mathsf{else}\ 0 \\
\longrightarrow & \mathsf{tcase}\ \langle 7,3\rangle{:}(\lambda\alpha'{:}\Omega.\alpha' \times \alpha')\ \mathsf{int}\ \mathsf{of}\ x'{:}\mathsf{int} \times \mathsf{int}\ \mathsf{then}\ 1\ \mathsf{else}\ 0 \\
\longrightarrow & \mathsf{tcase}\ \langle 7,3\rangle{:}\mathsf{int} \times \mathsf{int}\ \mathsf{of}\ x'{:}\mathsf{int} \times \mathsf{int}\ \mathsf{then}\ 1\ \mathsf{else}\ 0 \\
\longrightarrow & 1
\end{array}
$$

Figure 3.2: Sample reduction sequence

3.1.1 Properties

We define another judgment, analogous to $\Gamma \vdash L : \Gamma'$, that is needed to set up suitable context lemmata for the Preservation proof. The context lemmata for term evaluation contexts, lemma 7 (Context Elimination) and lemma 8 (Exchange), do not require such a judgement, because E—in contrast to T—does not allow reduction below binders.

Definition 5 (Typing Judgement for Type Reduction Contexts). $\boxed{\Gamma \vdash T : \Gamma'}$

$$
(\text{TC-Empty})\ \frac{}{\Gamma \vdash \_ : \cdot} \qquad (\text{TC-Arrow1})\ \frac{\Gamma \vdash T : \Gamma'}{\Gamma \vdash T \to \tau : \Gamma'} \qquad (\text{TC-Arrow2})\ \frac{\Gamma \vdash T : \Gamma'}{\Gamma \vdash \nu \to T : \Gamma'}
$$

$$
(\text{TC-Times1})\ \frac{\Gamma \vdash T : \Gamma'}{\Gamma \vdash T \times \tau : \Gamma'} \qquad (\text{TC-Times2})\ \frac{\Gamma \vdash T : \Gamma'}{\Gamma \vdash \nu \times T : \Gamma'} \qquad (\text{TC-Univ})\ \frac{\Gamma, \alpha{:}\kappa \vdash T : \Gamma'}{\Gamma \vdash \forall\alpha{:}\kappa.T : \alpha{:}\kappa, \Gamma'}
$$

$$
(\text{TC-Exist})\ \frac{\Gamma, \alpha{:}\kappa \vdash T : \Gamma'}{\Gamma \vdash \exists\alpha{:}\kappa.T : \alpha{:}\kappa, \Gamma'} \qquad (\text{TC-Abs})\ \frac{\Gamma, \alpha{:}\kappa \vdash T : \Gamma'}{\Gamma \vdash \lambda\alpha{:}\kappa.T : \alpha{:}\kappa, \Gamma'}
$$

$$
(\text{TC-App1})\ \frac{\Gamma \vdash T : \Gamma'}{\Gamma \vdash T\ \tau : \Gamma'} \qquad (\text{TC-App2})\ \frac{\Gamma \vdash T : \Gamma'}{\Gamma \vdash \nu\ T : \Gamma'} \qquad (\text{TC-Pair1})\ \frac{\Gamma \vdash T : \Gamma'}{\Gamma \vdash \langle T, \tau\rangle : \Gamma'}
$$

$$
(\text{TC-Pair2})\ \frac{\Gamma \vdash T : \Gamma'}{\Gamma \vdash \langle \nu, T\rangle : \Gamma'} \qquad (\text{TC-Proj1})\ \frac{\Gamma \vdash T : \Gamma'}{\Gamma \vdash T.1 : \Gamma'} \qquad (\text{TC-Proj2})\ \frac{\Gamma \vdash T : \Gamma'}{\Gamma \vdash T.2 : \Gamma'}
$$

As in the previous chapter, we only show a few selected parts of the proofs (if at all) and refer to the appendix for the rest.

Lemma 10 (Type Context Elimination).

1. *If* $\Gamma \vdash T[\tau] : \kappa$*, then* $\Gamma, \Gamma' \vdash \tau : \kappa'$ *where* $\Gamma \vdash T : \Gamma'$.
2. *If* $\Gamma \vdash T : \Gamma'$*, then* $\mathit{dom}(\Gamma') \cap \mathit{LTVar} = \emptyset$.
3. *If* $\Gamma \vdash C[\tau] : \tau'$*, then* $\Gamma \vdash \tau : \Omega$.

Proof. (1) by induction on the structure of T. (2) follows immediately from the definition of the $\Gamma \vdash T : \Gamma'$ judgement. (3) by case analysis on C.

1. Let $\Gamma \vdash T[\tau] : \kappa$
   - Case $T = \forall\alpha{:}\kappa.T'$:
     (a) By assumption: $\Gamma \vdash \forall\alpha{:}\kappa.T'[\tau] : \kappa$
     (b) By (a) and inversion of K-Univ: $\Gamma, \alpha{:}\kappa \vdash T'[\tau] : \Omega$

(c) By (b) and induction: $\Gamma, \alpha{:}\kappa, \Gamma'' \vdash \tau : \kappa'$ where $\Gamma \vdash T' : \Gamma''$

(d) Let $\Gamma' = \alpha{:}\kappa, \Gamma''$.

(e) By (b), (c), (d), and TC-Univ: $\Gamma, \Gamma' \vdash \tau : \kappa'$ where $\Gamma \vdash \forall\alpha{:}\kappa.T' : \Gamma'$

□

Lemma 11 (Wrapping)**.** *If $\tau \equiv \tau'$, then, for all T, $T[\tau] \equiv T[\tau']$.*

Proof. By structural induction on T. □

Lemma 12 (Type Exchange)**.**

1. *If $\Gamma \vdash T[\tau] : \kappa$ and $\Gamma, \Gamma' \vdash \tau : \kappa$ and $\Gamma, \Gamma' \vdash \tau' : \kappa$ where $\Gamma \vdash T : \Gamma'$, then $\Gamma \vdash T[\tau'] : \kappa$.*

2. *If $\Gamma \vdash CT[\tau] : \tau''$ and $\Gamma, \Gamma' \vdash \tau : \kappa$ and $\Gamma, \Gamma' \vdash \tau' : \kappa$ and $\Gamma \vdash T : \Gamma'$ and $\tau \equiv \tau'$, then $\Gamma \vdash CT[\tau'] : \tau''$.*

Proof. (1) by structural induction on T. (2) by case analysis on C, using (1), lemma 6 (Equivalent Environments), and lemma 11 (Wrapping).

2. Let $\Gamma \vdash CT[\tau] : \tau''$ and $\Gamma, \Gamma' \vdash \tau : \kappa$ and $\Gamma, \Gamma' \vdash \tau' : \kappa$ and $\Gamma \vdash T : \Gamma'$ and $\tau \equiv \tau'$.

   - Case $C = \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$:

     (a) By assumption: $\Gamma \vdash \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}T[\tau]\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau''$

     (b) By (a) and Typing Inversion: $\Gamma \vdash e_0 : \tau_0$ and $\Gamma, x{:}T[\tau] \vdash e_1 : \tau''$ and $\Gamma \vdash e_2 : \tau''$

     (c) By (b) and Environment Validity: $\Gamma, x{:}T[\tau] \vdash \Box$

     (d) By (c) and inversion of E-Term: $\Gamma \vdash T[\tau] : \Omega$ and $x \notin dom(\Gamma)$

     (e) By (d), assumption, and (1): $\Gamma \vdash T[\tau'] : \Omega$

     (f) By (e), (d), and E-Term: $\Gamma, x{:}T[\tau'] \vdash \Box$

     (g) By assumption and Wrapping: $T[\tau] \equiv T[\tau']$

     (h) By (b), (f), (g), and Equivalent Environments: $\Gamma, x{:}T[\tau'] \vdash e_1 : \tau''$

     (i) By (h), (b), and T-Case: $\Gamma \vdash \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}T[\tau']\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau''$

□

Theorem 1 (Preservation)**.** *If $\Gamma \vdash e : \tau$ and $e \longrightarrow e'$, then $\Gamma \vdash e' : \tau$.*

Proof. By case analysis on the applied reduction rule. The basic idea is the same for all cases where the reduction rule looks like $LE[e_1] \longrightarrow LE[e_2]$: first, we use Context Elimination to show that $\Gamma, \Gamma' \vdash e_1 : \tau'$ with $\Gamma \vdash L : \Gamma'$. We then show $\Gamma, \Gamma' \vdash e_2 : \tau'$ and finally use Exchange to get $\Gamma \vdash LE[e_2] : \tau$. While it works similar for R-Suspend, the cases R-Trigger, and RT-Trigger are much more involved.

- Case R-Open: $e = LE[\mathsf{let}\ \langle\alpha, x\rangle = \langle\tau_1, v\rangle{:}\tau_1'\ \mathsf{in}\ e_1]$ and $e' = LE[e_1[\alpha := \tau_1][x := v]]$:

  1. By Context Elimination: $\Gamma, \Gamma' \vdash \mathsf{let}\ \langle\alpha, x\rangle = \langle\tau_1, v\rangle{:}\tau_1'\ \mathsf{in}\ e_1 : \tau'$ where $\Gamma \vdash L : \Gamma'$
  2. We show: $\Gamma, \Gamma' \vdash e_1[x_1 := v_1][x_2 := v_2] : \tau'$

     (a) By (1) and Typing Inversion: $\Gamma, \Gamma' \vdash \langle\tau_1, v\rangle{:}\tau_1' : \exists\alpha{:}\kappa.\tau_2$ and $\Gamma, \Gamma', \alpha{:}\kappa, x{:}\tau_2 \vdash e_1 : \tau''$ where $\alpha \notin ftv(\tau'')$ and $\tau'' \equiv \tau'$

     (b) By (a) and Typing Inversion: $\Gamma, \Gamma' \vdash \tau_1 : \kappa'$ and $\Gamma, \Gamma' \vdash v : \tau_2[\alpha := \tau_1]$ where $\tau_1' = \exists\alpha{:}\kappa'.\tau_2'$ and $\exists\alpha{:}\kappa.\tau_2 \equiv \exists\alpha{:}\kappa'.\tau_2'$ and $\Gamma \vdash \tau_1' : \Omega$

     (c) By (b) and Type Equivalence Inversion: $\kappa = \kappa'$ and $\tau_2 \equiv \tau_2'$

     (d) By (a), (b), (c), and Type Substitution: $\Gamma, \Gamma', x{:}\tau_2[\alpha := \tau_1] \vdash e_1[\alpha := \tau_1] : \tau''[\alpha := \tau_1]$

   (e) By (d), (b), and Term Substitution: $\Gamma, \Gamma' \vdash e_1[\alpha := \tau_1][x := v] : \tau''[\alpha := \tau_1]$
   (f) By (a) and (e): $\Gamma, \Gamma' \vdash e_1[\alpha := \tau_1][x := v] : \tau''$
   (g) By (1) and Validity: $\Gamma, \Gamma' \vdash \tau' : \Omega$
   (h) By (a), (f), (g), and T-Equiv: $\Gamma, \Gamma' \vdash e_1[\alpha := \tau_1][x := v] : \tau'$
3. By assumption, (1), (2), and Exchange: $\Gamma \vdash LE[e_1[\alpha := \kappa][x := v]] : \tau$

- Case RT-Trigger: $e = L[\mathsf{lazy}\ \langle\zeta, x\rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ L'ECT[\zeta]]$ and $e' = L[\mathsf{let}\ \langle\zeta, x\rangle\ \mathsf{=}\ e_1\ \mathsf{in}$ $(L'ECT[\zeta])[\zeta := \alpha]]$

  1. By assumption: $\Gamma \vdash L[\mathsf{lazy}\ \langle\zeta, x\rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ L'ECT[\zeta]] : \tau$
  2. By (1) and Context Elimination: $\Gamma, \Gamma' \vdash \mathsf{lazy}\ \langle\zeta, x\rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ L'ECT[\zeta] : \tau'$ where $\Gamma \vdash L : \Gamma'$
  3. By (2) and Typing Inversion: $\Gamma, \Gamma' \vdash e_1 : \exists\alpha{:}\kappa.\tau_1$ and $\Gamma, \Gamma', \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash L'ECT[\zeta] : \tau''$ where $\zeta \notin ftv(\tau'')$ and $\tau'' \equiv \tau'$
  4. We show: $\Gamma, \Gamma', \alpha{:}\kappa \vdash \alpha : \kappa$
     (a) By (3) and Validity: $\Gamma, \Gamma' \vdash \exists\alpha{:}\kappa.\tau_1 : \Omega$
     (b) By (a) and inversion of K-Exist: $\Gamma, \Gamma', \alpha{:}\kappa \vdash \tau_1 : \Omega$
     (c) By (b) and Environment Validity: $\Gamma, \Gamma', \alpha{:}\kappa \vdash \Box$
     (d) By (c): $\Gamma, \Gamma', \alpha{:}\kappa \vdash \alpha : \kappa$
  5. We show: $\Gamma, \Gamma', \alpha{:}\kappa, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash L'ECT[\zeta] : \tau''$
     (a) By (3) and Environment Validity: $\Gamma, \Gamma', \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash \Box$
     (b) By (a) and inversion of E-Term: $\Gamma, \Gamma', \zeta{:}\kappa \vdash \tau_1[\alpha := \zeta] : \Omega$ and $x \notin dom(\Gamma, \Gamma', \zeta{:}\kappa)$
     (c) By (b) and Environment Validity: $\Gamma, \Gamma', \zeta{:}\kappa \vdash \Box$
     (d) By (c) and inversion of E-Type: $\Gamma, \Gamma' \vdash \Box$ and $\zeta \notin dom(\Gamma, \Gamma')$
     (e) By (4c) and inversion of E-Type: $\alpha \notin dom(\Gamma, \Gamma')$
     (f) By (d), (e), and E-Type: $\Gamma, \Gamma', \alpha{:}\kappa \vdash \Box$
     (g) By (d): $\zeta \notin dom(\Gamma, \Gamma', \alpha{:}\kappa)$
     (h) By (f), (g), and E-Type: $\Gamma, \Gamma', \alpha{:}\kappa, \zeta{:}\kappa \vdash \Box$
     (i) By (b), (h), and Weakening: $\Gamma, \Gamma', \alpha{:}\kappa, \zeta{:}\kappa \vdash \tau_1[\alpha := \zeta] : \Omega$
     (j) By (b): $x \notin dom(\Gamma, \Gamma', \alpha{:}\kappa, \zeta{:}\kappa)$
     (k) By (i), (j), and E-Term: $\Gamma, \Gamma', \alpha{:}\kappa, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash \Box$
     (l) By (3), (k), and Weakening: $\Gamma, \Gamma', \alpha{:}\kappa, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash L'ECT[\zeta] : \tau''$
  6. By (4), (5), and Type Substitution:
     $\Gamma, \Gamma', \alpha{:}\kappa, x{:}\tau_1[\alpha := \zeta][\zeta := \alpha] \vdash (L'ECT[\zeta])[\zeta := \alpha] : \tau''[\zeta := \alpha]$
  7. We show: $\zeta \notin ftv(\tau_1)$
     (a) By (4b) and Variable Containment: $ftv(\tau_1) \subseteq dom(\Gamma, \Gamma', \alpha{:}\kappa)$
     (b) By (5d): $\zeta \notin dom(\Gamma, \Gamma', \alpha{:}\kappa)$
     (c) By (a) and (b): $\zeta \notin ftv(\tau_1)$
  8. By (6) and (7): $\Gamma, \Gamma', \alpha{:}\kappa, x{:}\tau_1 \vdash (L'ECT[\zeta])[\zeta := \alpha] : \tau''[\zeta := \alpha]$
  9. By (3) and (8): $\Gamma, \Gamma', \alpha{:}\kappa, x{:}\tau_1 \vdash (L'ECT[\zeta])[\zeta := \alpha] : \tau''$
  10. We show: $\alpha \notin ftv(\tau'')$
      (a) By (3) and Variable Containment: $ftv(\tau'') \subseteq dom(\Gamma, \Gamma', \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta])$
      (b) By (5e): $\alpha \notin dom(\Gamma, \Gamma', \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta])$
      (c) By (a) and (b): $\alpha \notin ftv(\tau'')$

11. By (3), (9), (10), and T-Open: $\Gamma, \Gamma' \vdash \mathsf{let}\ \langle \alpha, x\rangle = e_1\ \mathsf{in}\ (L'ECT[\zeta])[\zeta := \alpha] : \tau''$
12. By (2) and Validity: $\Gamma, \Gamma' \vdash \tau' : \Omega$
13. By (3), (11), (12), and T-Equiv: $\Gamma, \Gamma' \vdash \mathsf{let}\ \langle \alpha, x\rangle = e_1\ \mathsf{in}\ (L'ECT[\zeta])[\zeta := \alpha] : \tau'$
14. By (1), (2), (13), and Exchange: $\Gamma \vdash L[\mathsf{let}\ \langle \alpha, x\rangle = e_1\ \mathsf{in}\ (L'ECT[\zeta])[\zeta := \alpha]] : \tau$

□

Lemma 13 (Lazy Type Variables)**.** *If* $\Gamma \vdash LCT[\zeta] : \tau$ *and* $\zeta \notin dom(\Gamma)$*, then* $L = L_1[\mathsf{lazy}\ \langle \zeta, x\rangle = e\ \mathsf{in}\ L_2]$ *where* $\zeta \notin btv(L_2)$.

Proof. By structural induction on L. □

Proposition 13 (Canonical Normal Forms)**.** *Let* $\Gamma \vdash \nu : \kappa$ *where* ν *is not a normal path.*

- *If* $\kappa = \kappa_1 \to \kappa_2$*, then* $\nu = \lambda\alpha{:}\kappa.\nu'$.
- *If* $\kappa = \kappa_1 \times \kappa_2$*, then* $\nu = \langle \nu_1, \nu_2\rangle$.

Proof. Follows from the definition of the kinding relation. □

Proposition 14 (Type Progress)**.** *If* $\cdot \vdash LCT[\tau] : \tau'$ *and* τ *is not a normal form, then* $LCT[\tau] \longrightarrow e$.

Proof. By structural induction on τ, using lemma 13 (Lazy Type Variables).

- Case $\tau = \tau_1.1$:
  - Subcase τ_1 is not a normal form:
    1. Let $T' = T[\_.1]$. Then: $LCT[\tau] = LCT'[\tau_1]$
    2. By (1) and induction: $LCT[\tau] \longrightarrow e$
  - Subcase $\tau_1 = \nu_1$:
    * Subsubcase ν_1 is not a path:
      1. By assumption and Context Elimination: $\Gamma \vdash CT[\nu_1.1] : \tau''$
      2. By (1) and Type Context Elimination: $\Gamma' \vdash T[\nu_1.1] : \Omega$
      3. By (2) and Type Context Elimination: $\Gamma, \Gamma' \vdash \nu_1.1 : \kappa$
      4. By (3) and inversion of K-Proj1: $\Gamma, \Gamma' \vdash \nu_1 : \kappa \times \kappa_2$
      5. By (4) and Canonical Normal Forms: $\nu_1 = \langle \nu_{11}, \nu_{12}\rangle$
      6. By (5) and RT-Proj1: $LCT[\tau] = LCT[\langle \nu_{11}, \nu_{12}\rangle.1] \longrightarrow LCT[\nu_{11}]$
    * Subsubcase $\nu_1 = p$: not possible since $p.1$ is a normal form

□

Lemma 14 (Context Extension)**.** *If* $L[e] \longrightarrow e'$ *and* e *is not a* lazy *expression, then for all* E *exists an* e'' *such that* $LE[e] \longrightarrow e''$.

Proof. By case analysis on the applied reduction rule. □

The most standard formulation of Progress would look like

$$\textit{If } e \neq L[v] \textit{ and } \cdot \vdash e : \tau\textit{, then } e \to e'.$$

However, this cannot be proven directly by induction on the structure of e: if $e = \mathsf{lazy}\ \langle \zeta, x\rangle = e_1\ \mathsf{in}\ e_2$ and we assume that no trigger-rule can be applied, then we know that e_2 can be reduced. However, we cannot make use of the induction hypothesis, because e_2 may contain free variables (namely ζ and x). We therefore do not analyze the whole source term but only the part within the (maximal) surrounding lazy context. That way it is impossible that the term we are analyzing is a lazy expression.

Theorem 2 (Progress)**.** *If* $\cdot \vdash L[e] : \tau$ *where* e *is neither a value nor a* lazy *expression, then* $L[e] \longrightarrow e'$*.*

Proof. By structural induction on e, using proposition 10 (Canonical Values) and lemma 9 (Lazy Term Variables). The case where e is a tcase expression uses proposition 14 (Type Progress).

- Case $e = E[\mathsf{lazy}\ \langle\zeta, x\rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ e_2]$ (w.l.o.g. $\zeta \notin ftv(E)$ and $x \notin fv(E)$):
  1. By assumption: $E \neq \_$
  2. By (1) and R-Suspend: $L[e] \longrightarrow L[\mathsf{lazy}\ \langle\zeta, x\rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ E[e_2]]$
- Case $e \neq E[\mathsf{lazy}\ \dots] \wedge e = e_1\ e_2$:
  - Subcase e_1 is not a value:
    1. Let $E_1 = \_\ e_2$.
    2. By (1) and assumption: $\cdot \vdash LE_1[e_1] : \tau$
    3. By (2) and Context Elimination: $\cdot \vdash L[e_1] : \tau_1'$
    4. By assumption: e_1 is not a lazy expression
    5. By (3), (4), and induction: $L[e_1] \longrightarrow e_1'$
    6. By (4), (5), and Context Extension: $L[e] = LE_1[e_1] \longrightarrow e'$
  - Subcase $e_1 = v_1$:
    - Subsubcase v_1 is not a variable:
      1. By assumption and Context Elimination: $\Gamma \vdash v_1\ e_2 : \tau'$ where $\cdot \vdash L : \Gamma$
      2. By (1) and Typing Inversion: $\Gamma \vdash v_1 : \tau_2 \rightarrow \tau'$
      3. By (2) and Canonical Values: $v_1 = \lambda x{:}\tau_2'.e'$
      4. By (3) and R-App: $L[e] = L[(\lambda x{:}\tau_2.e')\ v_2] \longrightarrow L[e'[x := v_2]]$
    - Subsubcase $v_1 = x$:
      1. Let $S = \_\ e_2$ and $C = S$. Then: $e = S[x] = C[x]$
      2. By (1) and assumption: $\cdot \vdash LC[x] : \tau$
      3. By (2) and Lazy Term Variables: $L = L_1[\mathsf{lazy}\ \langle\zeta, x\rangle\ \mathsf{=}\ e'\ \mathsf{in}\ L_2]$ where $x \notin bv(L_2)$
      4. By (1), (3), and R-Trigger: $L[e] = L_1[\mathsf{lazy}\ \langle\zeta, x\rangle\ \mathsf{=}\ e'\ \mathsf{in}\ L_2S[x]] \longrightarrow L_1[\mathsf{let}\ \langle\alpha, x\rangle\ \mathsf{=}\ e'\ \mathsf{in}\ (L_2S[x])[\zeta := \alpha]]$
- Case $e \neq E[\mathsf{lazy}\ \dots] \wedge e = \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$:
  - Subcase e_0 is not a value:
    1. Let $E_0 = \mathsf{tcase}\ \_{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$.
    2. By (1) and assumption: $\cdot \vdash LE_0[e_0] : \tau$
    3. By (2) and Context Elimination: $\cdot \vdash L[e_0] : \tau_0''$
    4. By assumption: e_0 is not a lazy expression
    5. By (3), (4), and induction: $L[e_0] \longrightarrow e_0'$
    6. By (4), (5), and Context Extension: $L[e] = LE_0[e_0] \longrightarrow e'$
  - Subcase $e_0 = v$ and τ_0 is not a normal form:
    1. Let $C = \mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$. Then: $L[e] = LC[\tau_0]$
    2. By (1) and Type Progress: $L[e] \longrightarrow e'$
  - Subcase $e_0 = v$ and $\tau_0 = \nu$:
    - Subsubcase τ_0' is not a normal form:

1. Let $C = \mathsf{tcase}\ v{:}\tau_0\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$. Then: $L[e] = LC[\tau_0']$
2. By (1) and Type Progress: $L[e] \longrightarrow e'$

$*$ Subsubcase $\tau_0' = \nu$:

1. By R-Case1: $L[e] = L[\mathsf{tcase}\ v{:}\nu\ \mathsf{of}\ x{:}\nu\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2] \longrightarrow L[e_1[x := v]]$

$*$ Subsubcase $\tau_0' = \nu' \neq \nu$:

1. By R-Case2: $L[e] = L[\mathsf{tcase}\ v{:}\nu\ \mathsf{of}\ x{:}\nu'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2] \longrightarrow L[e_2]$

□

As another consequence of the nature of our reduction relation, the progress proof is not as constructive as usual. For instance, if $L[e] = L[\langle e_1, e_2\rangle]$, then we cannot conclude $e' = L[\langle e_1', e_2\rangle]$ or $e' = L[\langle e_1, e_2'\rangle]$ since the applicable rule may have been R-Suspend, R-Trigger, or RT-Trigger. However, the case analysis in the proof still shows implicitly that our reduction is fully deterministic.

3.2 Interleaved call-by-name reduction to weak head normal form

One might argue that the previous strategy is not lazy at all, because all occuring lazy type variables are eliminated (by opening the corresponding packages). Still, it is lazy in so far that RT-Trigger is only applied when reduction cannot proceed otherwise. Admittedly, the reduction is not very smart. Consider the following example:

$$\mathsf{lazy}\ \langle\zeta, x\rangle\ \texttt{=}\ e\ \mathsf{in}\ \mathsf{tcase}\ x{:}(\lambda\alpha{:}\Omega.\mathsf{int})\ \zeta\ \mathsf{of}\ x'{:}\mathsf{int}\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$$

It is obvious that the proper value of ζ is not needed here at all. Nevertheless, our reduction relation would trigger the evaluation and opening of e because it tries to reduce the argument of the application to a normal form before executing RT-App. This can be avoided by using call-by-name instead of call-by-value, i.e., by allowing β reduction to be performed even when the argument is not in normal form. (To maintain determinism the definition of type reduction contexts needs to be adapted as well.)

Looking at another example gives us insight on how to get an even lazier strategy:

$$\mathsf{lazy}\ \langle\zeta, x\rangle\ \texttt{=}\ e\ \mathsf{in}\ \mathsf{tcase}\ x{:}\zeta \rightarrow \mathsf{int}\ \mathsf{of}\ x'{:}\mathsf{int} \times \mathsf{int}\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$$

According to the reduction relation, e must be evaluated and opened to eliminate ζ, because $\zeta \rightarrow \mathsf{int}$ is not in normal form yet. It is, however, already clear that the two type expressions can never be equal: one is an arrow type and the other a product type—and this will not change, no matter what ζ is replaced with. An algorithm that is aware of such implications not only does lead to a faster termination but also avoids unnecessary triggering of lazy packages and thus provides a higher degree of laziness.

This section presents a strategy that makes use of these two techniques. In short, the idea is that we repeatedly perform a combination of weak head reduction, comparison, and descent—until we either obtain two identical normal forms or discover that the types cannot be equal. The R-Case1 rule remains unchanged.

An expression is in weak head normal form [12, 23], if and only if it has no top-level redex. In our calculus we extend this definition to exclude lazy variables, i.e., all type expressions except lazy variables, applications, and projections are always in weak head normal form. An application is in weak head normal form iff β-reduction cannot be performed because the left-hand side is a variable or itself an expression that cannot be reduced. The like holds for a projection. We make this formal with the following syntax definitions:

weak head normal forms $\omega ::= q \mid \tau_1 \to \tau_2 \mid \tau_1 \times \tau_2 \mid \forall\alpha{:}\kappa.\tau \mid \exists\alpha{:}\kappa.\tau \mid \lambda\alpha{:}\kappa.\tau \mid \langle\tau_1, \tau_2\rangle$

weak head normal paths $q ::= \alpha \mid q\ \tau \mid q.1 \mid q.2$

When we have to evaluate a tcase expression, i.e., when we have to compare two type expressions τ_1 and τ_2, then we first reduce them to weak head normal forms ω_1 and ω_2. Afterwards, we compare their heads, and, in case of existential and universal types, the kinds of the bound variables as well[2]. If they are different (written $\omega_1 \not\sim \omega_2$, see below), then, knowing that the types never can be equal, we abort using a R-Case2 rule like

$$LE[\mathsf{tcase}\ v{:}\omega\ \mathsf{of}\ x{:}\omega'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2] \longrightarrow E[e_2] \quad (\omega \not\sim \omega')$$

where the $\sim$ relation is defined as follows:

$$\begin{array}{rcl} \omega \sim \omega' & \overset{\text{def}}{\iff} & \omega = q \ \wedge\ \omega' = q' \ \wedge\ q \sim q' \\ & \vee & \omega = \tau_1 \to \tau_2 \ \wedge\ \omega' = \tau_1' \to \tau_2' \\ & \vee & \omega = \tau_1 \times \tau_2 \ \wedge\ \omega' = \tau_1' \times \tau_2' \\ & \vee & \omega = \forall\alpha{:}\kappa.\tau \ \wedge\ \omega' = \forall\alpha{:}\kappa.\tau' \\ & \vee & \omega = \exists\alpha{:}\kappa.\tau \ \wedge\ \omega' = \exists\alpha{:}\kappa.\tau' \\ & \vee & \omega = \lambda\alpha{:}\kappa.\tau \ \wedge\ \omega' = \lambda\alpha{:}\kappa.\tau' \\ & \vee & \omega = \langle\tau_1, \tau_2\rangle \ \wedge\ \omega' = \langle\tau_1', \tau_2'\rangle \end{array}$$

$$\begin{array}{rcl} q \sim q' & \overset{\text{def}}{\iff} & q = \alpha = q' \\ & \vee & q = q_1\ \tau \ \wedge\ q' = q_1'\ \tau' \\ & \vee & q = q_1.1 \ \wedge\ q' = q_1'.1 \\ & \vee & q = q_1.2 \ \wedge\ q' = q_1'.2 \end{array}$$

But how do we continue if $\omega \sim \omega'$? Let us consider an example:

$$\mathsf{lazy}\ \langle\zeta, x\rangle = e_0\ \mathsf{in}\ \mathsf{tcase}\ x{:}\exists\alpha{:}\Omega.\alpha\ \mathsf{of}\ x'{:}\exists\alpha{:}\Omega.\zeta\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$$

We would like to reduce the problem of comparing $\exists\alpha{:}\Omega.\alpha$ and $\exists\alpha{:}\Omega.\zeta$ to the problem of comparing α and ζ, because the latter are equal if and only if the former are. We therefore need a way to descend into the existential types and start the whole procedure again on their bodies until we either abort or reach two identical normal forms ν_1 and ν_2 (as defined in the previous section) in which case we use R-Case1. Obviously, we cannot simply throw away the quantifiers by reducing the term to

$$\mathsf{lazy}\ \langle\zeta, x\rangle = e_0\ \mathsf{in}\ \mathsf{tcase}\ x{:}\alpha\ \mathsf{of}\ x'{:}\zeta\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$$

because that breaks the preservation property (the term is no longer well-typed since x has type $\exists\alpha{:}\Omega.\alpha$, not type α). Instead, we once again make use of contexts and define a set of binary contexts B that allow us to descend into two weak head normal forms of the same shape. Unfortunately, the required grammar is not context-free this time. However, we still present it in the usual BNF style, because the meaning is obvious:

$$\begin{array}{rl} B ::= & \mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mid \\ & B[\_ \to \tau_1][\_ \to \tau_2] \mid B[\nu \to \_][\nu \to \_] \mid \\ & B[\_ \times \tau_1][\_ \times \tau_2] \mid B[\nu \times \_][\nu \times \_] \mid \\ & B[\forall\alpha{:}\kappa.\_][\forall\alpha{:}\kappa.\_] \mid B[\exists\alpha{:}\kappa.\_][\exists\alpha{:}\kappa.\_] \mid B[\lambda\alpha{:}\kappa.\_][\lambda\alpha{:}\kappa.\_] \mid \\ & B[\langle\_, \tau_1\rangle][\langle\_, \tau_2\rangle] \mid B[\langle\nu, \_\rangle][\langle\nu, \_\rangle] \end{array}$$

[2]This is not necessary for abstractions. We define the system such that, in a well-formed term, we are never comparing types of different kinds—aside from variables.

$$\begin{aligned}
&\mathsf{tcase}\ v{:}\exists\alpha_1{:}\Omega.\forall\alpha_2{:}\Omega.\alpha_1\ \mathsf{of}\ x{:}\exists\alpha_1{:}\Omega.\forall\alpha_2{:}\Omega.\alpha_2\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2\\
&\qquad = B_0[\exists\alpha_1{:}\Omega.\forall\alpha_2{:}\Omega.\alpha_1][\exists\alpha_1{:}\Omega.\forall\alpha_2{:}\Omega.\alpha_2]\\
&\qquad = B_1[\forall\alpha_2{:}\Omega.\alpha_1][\forall\alpha_2{:}\Omega.\alpha_2]\\
&\qquad = B_2[\alpha_1][\alpha_2]
\end{aligned}$$

$$\begin{aligned}
\text{where } B_0 &= \mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2\\
B_1 &= B_0[\exists\alpha_1{:}\Omega.\_][\exists\alpha_1{:}\Omega.\_]\\
B_2 &= B_1[\forall\alpha_2{:}\Omega.\_][\forall\alpha_2{:}\Omega.\_]
\end{aligned}$$

Figure 3.3: Sample decomposition using binary contexts

Note that e.g. in $B[\nu \to \_][\nu \to \_]$ the two ν really stand for one and the same normal form. These binary contexts (which we also call *comparison contexts*) can be defined accurately as a set of functions that take two types and yield a term:

$$\begin{aligned}
&B \in \mathit{BContext} \overset{\text{def}}{=}\\
&\qquad\{\lambda\ \tau_1,\tau_2 \in \mathit{Ty}.\ \mathsf{tcase}\ v{:}\tau_1\ \mathsf{of}\ x{:}\tau_2\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mid v \in \mathit{Val};\ x \in \mathit{Var};\ e_1,e_2 \in \mathit{Ter}\}\\
&\quad\cup \{\lambda\ \tau_1,\tau_2 \in \mathit{Ty}.\ B\ (\tau_1 \to \tau_1')\ (\tau_2 \to \tau_2') \mid \tau_1',\tau_2' \in \mathit{Ty};\ B \in \mathit{BContext}\}\\
&\quad\cup \{\lambda\ \tau_1,\tau_2 \in \mathit{Ty}.\ B\ (\nu \to \tau_1)\ (\nu \to \tau_2) \mid \nu \in \mathit{NF};\ B \in \mathit{BContext}\}\\
&\quad\cup \{\lambda\ \tau_1,\tau_2 \in \mathit{Ty}.\ B\ (\tau_1 \times \tau_1')\ (\tau_2 \times \tau_2') \mid \tau_1',\tau_2' \in \mathit{Ty};\ B \in \mathit{BContext}\}\\
&\quad\cup \{\lambda\ \tau_1,\tau_2 \in \mathit{Ty}.\ B\ (\nu \times \tau_1)\ (\nu \times \tau_2) \mid \nu \in \mathit{NF};\ B \in \mathit{BContext}\}\\
&\quad\cup \{\lambda\ \tau_1,\tau_2 \in \mathit{Ty}.\ B\ (\forall\alpha{:}\kappa.\tau_1)\ (\forall\alpha{:}\kappa.\tau_2) \mid \alpha \in \mathit{RTVar};\ \kappa \in \mathit{Ki};\ B \in \mathit{BContext}\}\\
&\quad\cup \{\lambda\ \tau_1,\tau_2 \in \mathit{Ty}.\ B\ (\exists\alpha{:}\kappa.\tau_1)\ (\exists\alpha{:}\kappa.\tau_2) \mid \alpha \in \mathit{RTVar};\ \kappa \in \mathit{Ki};\ B \in \mathit{BContext}\}\\
&\quad\cup \{\lambda\ \tau_1,\tau_2 \in \mathit{Ty}.\ B\ (\lambda\alpha{:}\kappa.\tau_1)\ (\lambda\alpha{:}\kappa.\tau_2) \mid \alpha \in \mathit{RTVar};\ \kappa \in \mathit{Ki};\ B \in \mathit{BContext}\}\\
&\quad\cup \{\lambda\ \tau_1,\tau_2 \in \mathit{Ty}.\ B\ \langle\tau_1,\tau_1'\rangle\ \langle\tau_2,\tau_2'\rangle \mid \tau_1',\tau_2' \in \mathit{Ty};\ B \in \mathit{BContext}\}\\
&\quad\cup \{\lambda\ \tau_1,\tau_2 \in \mathit{Ty}.\ B\ \langle\nu,\tau_1\rangle\ \langle\nu,\tau_2\rangle \mid \nu \in \mathit{NF};\ B \in \mathit{BContext}\}
\end{aligned}$$

(Where *Ty* denotes the set of types, *Val* the set of values, *Ter* the set of terms, *NF* the set of normal forms, and *Ki* the set of kinds.)

Figure 3.3 illustrates the decomposition of a tcase expression with the help of our new comparison contexts. Using an R-Case2 rule like

$$\begin{aligned}
&\mathit{LE}[\mathsf{tcase}\ v{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2] \longrightarrow \mathit{LE}[e_2]\\
&\ ((\mathsf{tcase}\ v{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) = B[\omega_1][\omega_2] \text{ and } \omega_1 \not\sim \omega_2)
\end{aligned}$$

$B_2[\alpha_1][\alpha_2]$ now reduces to e_2, as α_1 and α_2 are weak head normal forms with different heads. More examples follow as we proceed.

Figure 3.4 specifies the reduction that is responsible for producing weak head normal forms. It is a call-by-name reduction, i.e., the leftmost outermost redex not below a binder is reduced first [23]. As stated at the end of the previous section, call-by-name avoids unnecessary triggering if an abstraction is applied to a lazy variable. Similar, we perform projection as soon as possible, in the hope that this throws away a lazy variable (for instance, $\langle\zeta_1,\zeta_2\rangle.1$ is reduced to ζ_1). Consequently, RT-Trigger is only applied if it is indispensable in order to compute a weak head normal form. The C contexts determine that the left type of a tcase expression is reduced before the right type. We reuse the names C and T here, because this makes large parts of some proofs identical to the corresponding proofs for the applicative order reduction.

Together with our way of descending, the whole strategy can also be seen as a normal order reduction, since effectively the leftmost outermost redex is reduced first and the types end up in normal form (if they are equal).

The binary contexts are defined such that, when comparing arrow types, pair types, or type pairs, we may only descend into the right component if the left component already is in normal

tcase contexts	$C ::= \; B[\_][\tau] \mid B[\omega][\_]$
type reduction contexts	$T ::= \; \_ \mid T\ \tau \mid T.1 \mid T.2$

RT-App	$LECT[(\lambda\alpha{:}\kappa.\tau_1)\ \tau_2] \longrightarrow LECT[\tau_1[\alpha := \tau_2]]$
RT-Proj1	$LECT[\langle\tau_1, \tau_2\rangle.1] \longrightarrow LECT[\tau_1]$
RT-Proj2	$LECT[\langle\tau_1, \tau_2\rangle.2] \longrightarrow LECT[\tau_2]$
RT-Trigger	$L_1[\mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ L_2ECT[\zeta]] \longrightarrow L_1[\mathsf{let}\ \langle\alpha, x\rangle = e_1\ \mathsf{in}\ (L_2ECT[\zeta])[\zeta := \alpha]]$ $(\zeta \notin btv(L_2))$

Figure 3.4: Call-by-name reduction to weak head normal form

form. The idea is that we first process the left part and there, by recursion, either can apply R-Case2 at some point or end up with the same normal form on both sides. In the latter case we are then able to construct a B that allows us to delve into the right part. In other words, our binary contexts force a left-to-right depth-first traversal. Consider the following term:

$$\mathsf{tcase}\ v{:}(\exists\alpha{:}\Omega.(\lambda\alpha'{:}\Omega.\alpha')\ \alpha) \times \tau_1'\ \mathsf{of}\ x{:}(\exists\alpha{:}\Omega.\alpha) \times \tau_2'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$$

It is easy to see that whether the two types are equal solely depends on whether τ_1' and τ_2' are equal, hence we would like to find a binary context that lets us compare them. We start with $B_0 = \mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$. Looking at $B_0[(\exists\alpha{:}\Omega.(\lambda\alpha'{:}\Omega.\alpha')\ \alpha) \times \tau_1'][(\exists\alpha{:}\Omega.\alpha) \times \tau_2']$ now, it becomes apparent that we cannot focus on τ_1' and τ_2' already because $\exists\alpha{:}\Omega.(\lambda\alpha'{:}\Omega.\alpha')\ \alpha$ is not in normal form yet. We therefore have to choose $B_1 = B_0[\_ \times \tau_1'][\_ \times \tau_2']$ and thus to first descend into the left part of the type pair. This lets us view the term as $B_1[\exists\alpha{:}\Omega.(\lambda\alpha'{:}\Omega.\alpha')\ \alpha][\exists\alpha{:}\Omega.\alpha]$. Using $B_2 = B_1[\exists\alpha{:}\Omega.\_][\exists\alpha{:}\Omega.\_]$, we descend once more. For $T = \_$ and $C = B_2[\_][\alpha]$ we are now able to reduce $B_2[(\lambda\alpha'{:}\Omega.\alpha')\ \alpha][\alpha]$ to $B_2[\alpha][\alpha]$ by using the RT-App rule. Having arrived at two identical normal forms, we now ascend again to B_0. This time, however, we are looking at $B_0[(\exists\alpha{:}\Omega.\alpha) \times \tau_1'][(\exists\alpha{:}\Omega.\alpha) \times \tau_2']$. $\exists\alpha{:}\Omega.\alpha$ is a normal form and hence we can finally choose $B_1' = B_0[(\exists\alpha{:}\Omega.\alpha) \times \_][(\exists\alpha{:}\Omega.\alpha) \times \_]$ and thus compare τ_1' and τ_2'.

What about applications and projections in weak head normal form? So far we have ignored them. Consider the term $B[\alpha_1\ (\alpha_2\ \alpha_3)][\alpha_1\ \alpha_3]$ (for some B). It suggests itself to extend the binary contexts with $B[p\ \_][p\ \_]$ (again, the left and the right p denote the same normal path). We have to use p instead of ν here to exclude abstractions. Choosing $B' = B[\alpha_1\ \_][\alpha_1\ \_]$, we can then view the term as $B'[\alpha_2\ \alpha_3][\alpha_3]$ and reduce it to e_2 because $\alpha_2\ \alpha_3 \not\sim \alpha_3$. Now consider the term $B[(\alpha_1\ (\alpha_2\ \alpha_3))\ \alpha_2][(\alpha_1\ \alpha_3)\ \alpha_2]$. We have just seen that the left-hand sides of the outermost applications are not equal, hence these applications cannot be equal either. But how do we show it this time? At a first glance one might be tempted to add $B[\_\ \tau_1][\_\ \tau_2]$ to the definition of comparison contexts. Indeed this would allow us to descend into the left-hand sides ($B' = B[\_\ \alpha_2][\_\ \alpha_2]$) and then to continue as above. Nevertheless this extension has fatal implications! Suppose that we have to evaluate $\mathsf{tcase}\ v{:}\exists\alpha{:}(\Omega \to \Omega) \times \Omega.\alpha.1\ \alpha.2\ \mathsf{of}\ x{:}\exists\alpha{:}(\Omega \to \Omega) \times \Omega.(\lambda\alpha'{:}\Omega \to \Omega.\alpha'\ \alpha.2)\ \alpha.1\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$. Obviously the comparison should be successful, leading to the execution of R-Case1. Choosing $B = (\mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2)[\exists\alpha{:}(\Omega \to \Omega) \times \Omega.\_][\exists\alpha{:}(\Omega \to \Omega) \times \Omega.\_]$, $T = \_$, and $C = B[\alpha.1\ \alpha.2][\_]$, we can view the term as $CT[(\lambda\alpha'{:}\Omega \to \Omega.\alpha'\ \alpha.2)\ \alpha.1]$ and hence reduce it to $\mathsf{tcase}\ v{:}\exists\alpha{:}(\Omega \to \Omega) \times \Omega.\alpha.1\ \alpha.2\ \mathsf{of}\ x{:}\exists\alpha{:}(\Omega \to \Omega) \times \Omega.\alpha.1\ \alpha.2\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$. Using R-Case1, this term can be further reduced to $e_1[x := v]$ as expected. However, due to the previous extension there is a second possibility: if we choose $B' = B[\_\ \alpha.2][\_\ \alpha.1]$, then the term is equal to $B'[\alpha.1][\lambda\alpha'{:}\Omega \to \Omega.\alpha'\ \alpha.2]$. Because obviously

$\alpha.1 \not\sim \lambda\alpha'{:}\Omega \to \Omega.\alpha'\ \alpha.2$, it can therefore be reduced to e_2 by R-Case2. So the extension not only made the reduction nondeterministic but also non-confluent.

The problem is that we may only descend into the left part of an application if we know for sure that it is a path, i.e., that the leftmost innermost constituent of the type expression is a regular variable. We can express this, and thus solve the problem, by introducing a set of path comparison contexts:

$$P ::= \ B \mid P[\_\ \tau_1][\_\ \tau_2] \mid P[\_.1][\_.1] \mid P[\_.2][\_.2]$$

The previous harmful extension to the definition of regular comparison contexts is now replaced with $P[p\ \_][p\ \_]$. Since each regular comparison context also is a path comparison context, we can discard the first extension again, which was $B[p\ \_][p\ \_]$. All in all, the comparison contexts now are defined as follows:

$$\begin{aligned}
B ::= \ & \mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mid \\
& B[\_ \to \tau_1][\_ \to \tau_2] \mid B[\nu \to \_][\nu \to \_] \mid \\
& B[\_ \times \tau_1][\_ \times \tau_2] \mid B[\nu \times \_][\nu \times \_] \mid \\
& B[\forall\alpha{:}\kappa.\_][\forall\alpha{:}\kappa.\_] \mid B[\exists\alpha{:}\kappa.\_][\exists\alpha{:}\kappa.\_] \mid \\
& B[\lambda\alpha{:}\kappa.\_][\lambda\alpha{:}\kappa.\_] \mid P[p\ \_][p\ \_] \mid \\
& B[\langle\_, \tau_1\rangle][\langle\_, \tau_2\rangle] \mid B[\langle\nu, \_\rangle][\langle\nu, \_\rangle] \\
P ::= \ & B \mid P[\_\ \tau_1][\_\ \tau_2] \mid P[\_.1][\_.1] \mid P[\_.2][\_.2]
\end{aligned}$$

(The direct set definition can be trivially extended to match this updated version.)

As can be seen easily, there is now only the one, intended, reduction possible for the previous sample term. Let us look again at the example that took us here, $B[(\alpha_1\ (\alpha_2\ \alpha_3))\ \alpha_2][(\alpha_1\ \alpha_3)\ \alpha_2]$. With the help of path contexts we are now able to find a B' such that the term can be viewed as $B'[\alpha_2\ \alpha_3][\alpha_3]$, which makes R-Case2 applicable:

$$\begin{aligned}
P &= B \\
P' &= P[\_\ \alpha_2][\_\ \alpha_2] \\
B' &= P'[\alpha_1\ \_][\alpha_1\ \_]
\end{aligned}$$

There is still one case that is not covered, though. Consider an example where we are comparing two paths whose roots[3] differ: $B[\alpha.1.2][\alpha.2.2]$. There is only one possibility for descending: for $P = B$ and $P' = P[\_.2][\_.2]$ we can view the term as $P'[\alpha.1][\alpha.2]$. Clearly we would like to apply R-Case2 now, but its side condition is not fulfilled. We overcome this by modifying the rule:

$$\begin{gathered}
LE[\mathsf{tcase}\ v{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2] \longrightarrow LE[e_2] \\
((\mathsf{tcase}\ v{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) = B[\omega][\omega'] \text{ and } \omega \not\sim \omega' \\
\text{or } (\mathsf{tcase}\ v{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) = P[q][q'] \text{ and } q \not\sim q')
\end{gathered}$$

It is important that only weak head normal paths may be compared via path contexts—otherwise we encounter the same problem as before.

3.2.1 Properties

Definition 6 (Typing Judgement for comparison contexts). $\boxed{\Gamma \vdash B : \Gamma'}$

$$\text{(B-Case)}\ \frac{}{\Gamma \vdash \mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \cdot}$$

[3]We call the 'leftmost' variable in a path *root*. For instance, α is the root of both $\alpha\ \tau_1\ \tau_2$ and $\alpha.2\ \alpha'.1.1$.

$$\text{(B-Arrow1)}\ \frac{\Gamma \vdash B : \Gamma'}{\Gamma \vdash B[\_ \to \tau_1][\_ \to \tau_2] : \Gamma'} \qquad \text{(B-Arrow2)}\ \frac{\Gamma \vdash B : \Gamma'}{\Gamma \vdash B[\nu \to \_][\nu \to \_]}$$

$$\text{(B-Times1)}\ \frac{\Gamma \vdash B : \Gamma'}{\Gamma \vdash B[\_ \times \tau_1][\_ \times \tau_2] : \Gamma'} \qquad \text{(B-Times2)}\ \frac{\Gamma \vdash B : \Gamma'}{\Gamma \vdash B[\nu \times \_][\nu \times \_]}$$

$$\text{(B-Univ)}\ \frac{\Gamma \vdash B : \Gamma'}{\Gamma \vdash B[\forall\alpha{:}\kappa.\_][\forall\alpha{:}\kappa.\_] : \Gamma', \alpha{:}\kappa} \qquad \text{(B-Exist)}\ \frac{\Gamma \vdash B : \Gamma'}{\Gamma \vdash B[\exists\alpha{:}\kappa.\_][\exists\alpha{:}\kappa.\_] : \Gamma', \alpha{:}\kappa}$$

$$\text{(B-Abs)}\ \frac{\Gamma \vdash B : \Gamma'}{\Gamma \vdash B[\lambda\alpha{:}\kappa.\_][\lambda\alpha{:}\kappa.\_] : \Gamma', \alpha{:}\kappa} \qquad \text{(B-Path)}\ \frac{\Gamma \vdash P : \Gamma'}{\Gamma \vdash P[p\ \_][p\ \_] : \Gamma'}$$

$$\text{(B-Pair1)}\ \frac{\Gamma \vdash B : \Gamma'}{\Gamma \vdash B[\langle \_, \tau_1\rangle][\langle \_, \tau_2\rangle] : \Gamma'} \qquad \text{(B-Pair2)}\ \frac{\Gamma \vdash B : \Gamma'}{\Gamma \vdash B[\langle \nu, \_\rangle][\langle \nu, \_\rangle] : \Gamma'}$$

$$\boxed{\Gamma \vdash P : \Gamma'}$$

$$\text{(P-B)}\ \frac{\Gamma \vdash B : \Gamma'}{\Gamma \vdash B : \Gamma'} \qquad \text{(P-App)}\ \frac{\Gamma \vdash P : \Gamma'}{\Gamma \vdash P[\_\ \tau_1][\_\ \tau_2] : \Gamma'}$$

$$\text{(P-Proj1)}\ \frac{\Gamma \vdash P : \Gamma'}{\Gamma \vdash P[\_.1][\_.1] : \Gamma'} \qquad \text{(P-Proj2)}\ \frac{\Gamma \vdash P : \Gamma'}{\Gamma \vdash P[\_.2][\_.2] : \Gamma'}$$

Proposition 15 (Uniqueness of Environments)**.**

1. *If* $\Gamma \vdash B : \Gamma_1$ *and* $\Gamma \vdash B : \Gamma_2$*, then* $\Gamma_1 = \Gamma_2$*.*
2. *If* $\Gamma \vdash P : \Gamma_1$ *and* $\Gamma \vdash P : \Gamma_2$*, then* $\Gamma_1 = \Gamma_2$*.*

Proof. Simultaneous, by induction on the generation of B and P. □

Like before, the preservation proof requires various context lemmata.

Lemma 15 (Type Context Elimination)**.**

1. *If* $\Gamma \vdash B[\tau_1][\tau_2] : \tau$*, then* $\Gamma, \Gamma' \vdash \tau_1 : \kappa$ *and* $\Gamma, \Gamma' \vdash \tau_2 : \kappa$ *where* $\Gamma \vdash B : \Gamma'$*.*
2. *If* $\Gamma \vdash P[\tau_1][\tau_2] : \tau$*, then* $\Gamma, \Gamma' \vdash \tau_1 : \kappa_1$ *and* $\Gamma, \Gamma' \vdash \tau_2 : \kappa_2$ *where* $\Gamma \vdash P : \Gamma'$*.*
3. *If* $\Gamma \vdash B : \Gamma'$*, then* $dom(\Gamma') \cap LTVar = \emptyset$*.*
4. *If* $\Gamma \vdash C[\tau] : \tau'$*, then* $\Gamma, \Gamma' \vdash \tau : \kappa$ *and* $dom(\Gamma') \cap LTVar = \emptyset$*.*
5. *If* $\Gamma \vdash T[\tau] : \kappa$*, then* $\Gamma \vdash \tau : \kappa'$*.*

Proof. (1) and (2) simultaneous, by induction on the generation of B and P. (3) follows immediately from the definition of the $\Gamma \vdash B : \Gamma'$ judgement. (4) by case analysis on C, using (1). (5) by structural induction on T.

1. Let $\Gamma \vdash B[\tau_1][\tau_2] : \tau$.
   - Case $B = \mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$:
     (a) By Typing Inversion: $\Gamma \vdash v : \tau_1$ and $\Gamma, x{:}\tau_2 \vdash e_1 : \tau$
     (b) By (a) and Validity: $\Gamma \vdash \tau_1 : \Omega$
     (c) By (a) and Environment Validity: $\Gamma, x{:}\tau_2 \vdash \Box$
     (d) By (c) and inversion of E-Term: $\Gamma \vdash \tau_2 : \Omega$
     (e) By B-Case: $\Gamma \vdash \mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \cdot$

- Case $B = B'[\_ \to \tau_1'][\_ \to \tau_2']$:
  (a) By induction: $\Gamma, \Gamma' \vdash \tau_1 \to \tau_1' : \kappa$ and $\Gamma, \Gamma' \vdash \tau_2 \to \tau_2' : \kappa$ where $\Gamma \vdash B' : \Gamma'$
  (b) By (a) and inversion of K-Arrow: $\Gamma, \Gamma' \vdash \tau_1 : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2 : \Omega$
  (c) By (a) and B-Arrow1: $\Gamma \vdash B'[\_ \to \tau_1'][\_ \to \tau_2'] : \Gamma'$

□

Lemma 16 (Type Exchange)**.**

1. *If* $\Gamma \vdash T[\tau] : \kappa$ *and* $\Gamma \vdash \tau : \kappa'$ *as well as* $\Gamma \vdash \tau' : \kappa'$*, then* $\Gamma \vdash T[\tau'] : \kappa$*.*

2. *If* $\Gamma \vdash B[\tau_1][\tau_2] : \tau$ *and* $\Gamma, \Gamma' \vdash \tau_1 : \kappa$ *and* $\Gamma, \Gamma' \vdash \tau_1' : \kappa$ *and* $\tau_1 \equiv \tau_1'$ *and* $\Gamma, \Gamma' \vdash \tau_2 : \kappa$ *and* $\Gamma, \Gamma' \vdash \tau_2' : \kappa$ *and* $\tau_2 \equiv \tau_2'$ *and* $\Gamma \vdash B : \Gamma'$*, then* $\Gamma \vdash B[\tau_1'][\tau_2']$*.*

3. *If* $\Gamma \vdash P[\tau_1][\tau_2] : \tau$ *and* $\Gamma, \Gamma' \vdash \tau_1 : \kappa$ *and* $\Gamma, \Gamma' \vdash \tau_1' : \kappa$ *and* $\tau_1 \equiv \tau_1'$ *and* $\Gamma, \Gamma' \vdash \tau_2 : \kappa$ *and* $\Gamma, \Gamma' \vdash \tau_2' : \kappa$ *and* $\tau_2 \equiv \tau_2'$ *and* $\Gamma \vdash P : \Gamma'$*, then* $\Gamma \vdash P[\tau_1'][\tau_2']$*.*

Proof. (1) by structural induction on T. (2) and (3) simultaneous by induction on the derivation of B and P, using proposition 15 (Uniqueness of Environments) and lemma 15 (Type Context Elimination).

2. Let $\Gamma \vdash B[\tau_1][\tau_2] : \tau$ and $\Gamma, \Gamma' \vdash \tau_1 : \kappa$ and $\Gamma, \Gamma' \vdash \tau_1' : \kappa$ and $\tau_1 \equiv \tau_1'$ and $\Gamma, \Gamma' \vdash \tau_2 : \kappa$ and $\Gamma, \Gamma' \vdash \tau_2' : \kappa$ and $\tau_2 \equiv \tau_2'$ and $\Gamma \vdash B : \Gamma'$.

   - Case $B = \mathsf{tcase}\ v : \_\ \mathsf{of}\ x : \_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$ (w.l.o.g. $x \notin dom(\Gamma)$):
     (a) By Typing Inversion:
       i. $\Gamma \vdash v : \tau_1$
       ii. $\Gamma, x{:}\tau_2 \vdash e_1 : \tau$
       iii. $\Gamma \vdash e_2 : \tau$
     (b) By inversion of B-Case: $\Gamma' = \cdot$
     (c) We show: $\Gamma \vdash v : \tau_1'$
       i. By (a-i) and Validity: $\Gamma \vdash \tau_1 : \Omega$
       ii. By (i), (b), assumption, and Uniqueness of Kinds: $\kappa = \Omega$
       iii. By (ii), (b), and assumption: $\Gamma \vdash \tau_1' : \Omega$
       iv. By (a-i), (iii), assumption, and TEquiv: $\Gamma \vdash v : \tau_1'$
     (d) We show: $\Gamma, x{:}\tau_2' \vdash e_1 : \tau$
       i. By (c-ii), (b), and assumption: $\Gamma \vdash \tau_2' : \Omega$
       ii. By (i) and ETerm: $\Gamma, x{:}\tau_2' \vdash \Box$
       iii. By assumption: $\Gamma, x{:}\tau_2 \equiv \Gamma, x{:}\tau_2'$
       iv. By (a-ii), (ii), (iv), and Equivalent Environments: $\Gamma, x{:}\tau_2' \vdash e_1 : \tau$
     (e) By (c), (d), (a-iii), and TCase: $\Gamma \vdash \mathsf{tcase}\ v : \tau_1'\ \mathsf{of}\ x : \tau_2'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau$
   - Case $B = B'[\_ \to \tau_1''][\_ \to \tau_2'']$:
     (a) By inversion of B-Arrow1: $\Gamma \vdash B' : \Gamma'$
     (b) By Type Context Elimination: $\Gamma, \Gamma'' \vdash \tau_1 \to \tau_1'' : \kappa'$ and $\Gamma, \Gamma'' \vdash \tau_2 \to \tau_2'' : \kappa'$ where $\Gamma \vdash B' : \Gamma''$
     (c) By (a), (b), and Uniqueness of Environments: $\Gamma' = \Gamma''$
     (d) By (b), (c), and inversion of K-Arrow: $\kappa' = \Omega$ and $\Gamma, \Gamma' \vdash \tau_1 : \Omega$ and $\Gamma, \Gamma' \vdash \tau_1'' : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2 : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2'' : \Omega$
     (e) By (d), assumption, and Uniqueness of Kinds: $\kappa = \Omega$

(f) By (d), (e), assumption, and K-Arrow: $\Gamma, \Gamma' \vdash \tau_1' \to \tau_1'' : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2' \to \tau_2'' : \Omega$
(g) By assumption, Q-Refl, and Q-Arrow: $\tau_1 \to \tau_1'' \equiv \tau_1' \to \tau_1''$ and $\tau_2 \to \tau_2'' \equiv \tau_2' \to \tau_2''$
(h) By assumption, (a), (d), (f), (g), and induction: $\Gamma \vdash B'[\tau_1' \to \tau_1''][\tau_2' \to \tau_2''] : \tau$
(i) By (h): $\Gamma \vdash B[\tau_1'][\tau_2'] : \tau$

□

Lemma 17 (Wrapping). *If $\tau \equiv \tau'$, then, for all T, $T[\tau] \equiv T[\tau']$.*

Proof. By structural induction on T. □

Theorem 3 (Preservation). *If $\Gamma \vdash e : \tau$ and $e \longrightarrow e'$, then $\Gamma \vdash e' : \tau$.*

Proof. By case analysis on the applied reduction rule, using the context lemmata. Obviously, the proof can differ from the first preservation proof only for the tcase-related rules. Since we are reusing the names C and T, even the part for RT-Trigger can be adopted one-to-one. Thus, the only cases that need to be dealt with here are application and projection on type level—because of slightly different type context lemmata in comparison to the other strategy.

- Case RT-Proj2: $e = LECT[\langle\tau_1, \tau_2\rangle.2]$ and $e' = LECT[\tau_2]$

  1. By Context Elimination: $\Gamma, \Gamma' \vdash CT[\langle\tau_1, \tau_2\rangle.2] : \tau'$ where $\Gamma \vdash L : \Gamma'$
  2. We show: $\Gamma, \Gamma' \vdash CT[\tau_2] : \tau'$
     - Subcase $C = B[\_][\tau_3]$:
       (a) By (1): $\Gamma, \Gamma' \vdash B[T[\langle\tau_1, \tau_2\rangle.2]][\tau_3] : \tau'$
       (b) By (a) and Type Context Elimination: $\Gamma, \Gamma', \Gamma'' \vdash T[\langle\tau_1, \tau_2\rangle.2] : \kappa'$ and $\Gamma, \Gamma', \Gamma'' \vdash \tau_3 : \kappa'$ where $\Gamma, \Gamma' \vdash B : \Gamma''$
       (c) By (b) and Type Context Elimination: $\Gamma, \Gamma', \Gamma'' \vdash \langle\tau_1, \tau_2\rangle.2 : \kappa''$
       (d) By (c) and inversion of K-Proj2: $\Gamma, \Gamma', \Gamma'' \vdash \langle\tau_1, \tau_2\rangle : \kappa''' \times \kappa''$
       (e) By (d) and inversion of K-Pair: $\Gamma, \Gamma', \Gamma'' \vdash \tau_2 : \kappa''$
       (f) By (b), (c), (e), and Type Exchange: $\Gamma, \Gamma', \Gamma'' \vdash T[\tau_2] : \kappa'$
       (g) By Q-Proj1b: $\langle\tau_1, \tau_2\rangle.2 \equiv \tau_2$
       (h) By (g) and Wrapping: $T[\langle\tau_1, \tau_2\rangle.2] \equiv T[\tau_2]$
       (i) By (a), (b), (f), (h), and Type Exchange: $\Gamma, \Gamma' \vdash B[T[\tau_2]][\tau_3] : \tau'$
       (j) By (i): $\Gamma, \Gamma' \vdash CT[\tau_2] : \tau'$
     - Subcase $C = B[\omega][\_]$: analogous
  3. By assumption, (1), (2), and Exchange: $\Gamma \vdash LECT[\tau_2] : \tau$

□

Lemma 18 (Lazy Type Variables). *If $\Gamma \vdash LCT[\zeta] : \tau$ and $\zeta \notin dom(\Gamma)$, then $L = L_1[$*lazy $\langle\zeta, x\rangle$ = e in $L_2]$ *where $\zeta \notin btv(L_2)$.*

Proof. By structural induction on L. □

For the progress proof we need a lemma (let us call it lemma 1 for now) stating that, if $LB[\tau_1][\tau_2]$ is a closed and well-formed term, then it can be reduced. Unfortunately, this cannot be proven by induction on the size or structure of τ_1 or τ_2. The problem[4] arises if the analysed type is a variable and thus a normal form. What can we say about a term like $LB[\nu][\nu]$? Another

[4]We simplify matters a bit here.

lemma (lemma 2 for now) is required that states the following. One possibility is that no type-level reduction can be performed since the types are already in normal form. If these normal forms are different, then we can find a comparison context such that R-Case2 is applicable. Otherwise the term can be viewed as tcase $v{:}\nu$ of $x{:}\nu$ then e_1 else e_2, in which case we happily apply R-Case1. The second possibility is that, by further traversing the type expressions, we can find a B' such that $B[\nu][\nu] = B'[\tau_1'][\tau_2']$ for some τ_1' and τ_2' of which at least one is not in normal form yet. However, this fact does not help us much, because this type is not necessarily a part of ν (consider $B = B''[\langle \_, \tau_1' \rangle][\langle \_, \tau_2' \rangle]$ and $B' = B''[\langle \nu, \_ \rangle][\langle \nu, \_ \rangle]$), and so, in the proof of lemma 1, we cannot make use of the induction hypothesis. For a similar reason induction on the generation of B is not possible either.

Definition 7 (Decompositions)**.** *Let X range over B and P. We call (X, τ_1, τ_2) a* decomposition *of a term e, if $e = X[\tau_1][\tau_2]$. Furthermore, we say that (X', τ_1', τ_2') is a* deeper decomposition *than (X, τ_1, τ_2), if both decompose the same term e, and X' emerges from X by descending deeper into e.*

We therefore construct a special function, $\mathit{weight}(B, \tau_1, \tau_2)$ (and analogous $\mathit{weight}(P, \tau_1, \tau_2)$), that weights the decomposition of a term into a context and the resulting types. The general idea is that we prove lemma 1 by induction on the weight of the decomposition. In the case $LB[\nu][\nu]$ we use lemma 2, which either tells us directly that the term can be reduced or otherwise gives us a decomposition that has a lighter weight than (B, ν, ν) and therefore allows us to use the induction hypothesis.

The weight function has to reflect that our comparison contexts force a traversal that resembles left-to-right depth-first search: assuming $B[\tau_1][\tau_2] = B'[\tau_1'][\tau_2']$, then $\mathit{weight}(B, \tau_1, \tau_2) < \mathit{weight}(B', \tau_1', \tau_2')$ iff τ_1 is a predecessor of τ_1' in the preorder projection of the whole type's syntax tree.

Definition 8 (Weight of decompositions)**.**

For regular comparison contexts:

$$\begin{aligned}
\mathit{weight}(\mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2, \tau_1, \tau_2) &\overset{def}{=} \mathit{size}(\tau_1)\\
\mathit{weight}(B[\_ \to \tau_1'][\_ \to \tau_2'], \tau_1, \tau_2) &\overset{def}{=} \mathit{weight}(B, \tau_1 \to \tau_1', \tau_2 \to \tau_2') - \mathit{size}(\tau_1')\\
\mathit{weight}(B[\nu \to \_][\nu \to \_], \tau_1, \tau_2) &\overset{def}{=} \mathit{weight}(B, \nu \to \tau_1, \nu \to \tau_2) - \mathit{size}(\nu) - \mathit{size}(\tau_1)\\
\mathit{weight}(B[\_ \times \tau_1'][\_ \times \tau_2'], \tau_1, \tau_2) &\overset{def}{=} \mathit{weight}(B, \tau_1 \times \tau_1', \tau_2 \times \tau_2') - \mathit{size}(\tau_1')\\
\mathit{weight}(B[\nu \times \_][\nu \times \_], \tau_1, \tau_2) &\overset{def}{=} \mathit{weight}(B, \nu \times \tau_1, \nu \times \tau_2) - \mathit{size}(\nu) - \mathit{size}(\tau_1)\\
\mathit{weight}(B[\forall\alpha{:}\kappa.\_][\forall\alpha{:}\kappa.\_], \tau_1, \tau_2) &\overset{def}{=} \mathit{weight}(B, \forall\alpha{:}\kappa.\tau_1, \forall\alpha{:}\kappa.\tau_2) - 1\\
\mathit{weight}(B[\exists\alpha{:}\kappa.\_][\exists\alpha{:}\kappa.\_], \tau_1, \tau_2) &\overset{def}{=} \mathit{weight}(B, \exists\alpha{:}\kappa.\tau_1, \exists\alpha{:}\kappa.\tau_2) - 1\\
\mathit{weight}(B[\lambda\alpha{:}\kappa.\_][\lambda\alpha{:}\kappa.\_], \tau_1, \tau_2) &\overset{def}{=} \mathit{weight}(B, \lambda\alpha{:}\kappa.\tau_1, \lambda\alpha{:}\kappa.\tau_2) - 1\\
\mathit{weight}(P[p\ \_][p\ \_], \tau_1, \tau_2) &\overset{def}{=} \mathit{weight}(P, p\ \tau_1, p\ \tau_2) - \mathit{size}(p) - \mathit{size}(\tau_1)\\
\mathit{weight}(B[\langle \_, \tau_1' \rangle][\langle \_, \tau_2' \rangle], \tau_1, \tau_2) &\overset{def}{=} \mathit{weight}(B, \langle \tau_1, \tau_1' \rangle, \langle \tau_2, \tau_2' \rangle) - \mathit{size}(\tau_1')\\
\mathit{weight}(B[\langle \nu, \_ \rangle][\langle \nu, \_ \rangle], \tau_1, \tau_2) &\overset{def}{=} \mathit{weight}(B, \langle \nu, \tau_1 \rangle, \langle \nu, \tau_2 \rangle) - \mathit{size}(\nu) - \mathit{size}(\tau_1)
\end{aligned}$$

For path comparison contexts:

$$\begin{aligned}
\mathit{weight}(B, \tau_1, \tau_2) &\overset{def}{=} \mathit{weight}(B, \tau_1, \tau_2)\\
\mathit{weight}(P[\_.1][\_.1], \tau_1, \tau_2) &\overset{def}{=} \mathit{weight}(P, \tau_1.1, \tau_2.1) - 1\\
\mathit{weight}(P[\_.2][\_.2], \tau_1, \tau_2) &\overset{def}{=} \mathit{weight}(P, \tau_1.2, \tau_2.2) - 1\\
\mathit{weight}(P[\_\ \tau_1'][\_\ \tau_2'], \tau_1, \tau_2) &\overset{def}{=} \mathit{weight}(P, \tau_1\ \tau_1', \tau_2\ \tau_2') - \mathit{size}(\tau_1')
\end{aligned}$$

Definition 9 (Size of types)**.**

$$\begin{aligned}
\mathit{size}(\xi) &\overset{def}{=} 1\\
\mathit{size}(\tau_1 \to \tau_2) &\overset{def}{=} 2 \cdot \mathit{size}(\tau_1) + 2 \cdot \mathit{size}(\tau_2)\\
\mathit{size}(\tau_1 \times \tau_2) &\overset{def}{=} 2 \cdot \mathit{size}(\tau_1) + 2 \cdot \mathit{size}(\tau_2)\\
\mathit{size}(\forall\alpha{:}\kappa.\tau) &\overset{def}{=} 2 + \mathit{size}(\tau)\\
\mathit{size}(\exists\alpha{:}\kappa.\tau) &\overset{def}{=} 2 + \mathit{size}(\tau)\\
\mathit{size}(\lambda\alpha{:}\kappa.\tau) &\overset{def}{=} 2 + \mathit{size}(\tau)\\
\mathit{size}(\tau_1\ \tau_2) &\overset{def}{=} 2 \cdot \mathit{size}(\tau_1) + 2 \cdot \mathit{size}(\tau_2)\\
\mathit{size}(\langle \tau_1, \tau_2 \rangle) &\overset{def}{=} 2 \cdot \mathit{size}(\tau_1) + 2 \cdot \mathit{size}(\tau_2)\\
\mathit{size}(\tau.1) &\overset{def}{=} 2 + \mathit{size}(\tau)\\
\mathit{size}(\tau.2) &\overset{def}{=} 2 + \mathit{size}(\tau)
\end{aligned}$$

The proof of lemma 2 is by induction on the generation of B. But consider for instance the case $B = B''[\forall\alpha{:}\kappa.\_][\forall\alpha{:}\kappa.\_]$. The induction hypothesis yields that either the term can be reduced or is equal to $LB'[\tau_1][\tau_2]$ with $\mathit{weight}(B', \tau_1, \tau_2) < \mathit{weight}(B'', \forall\alpha{:}\kappa.\nu, \forall\alpha{:}\kappa.\nu)$. In the latter case we need to show that $\mathit{weight}(B', \tau_1, \tau_2)$ is also less than $\mathit{weight}(B''[\forall\alpha{:}\kappa.\_][\forall\alpha{:}\kappa.\_], \nu, \nu)$. However, all we know is that, by definition, $\mathit{weight}(B''[\forall\alpha{:}\kappa.\_][\forall\alpha{:}\kappa.\_], \nu, \nu) < \mathit{weight}(B'', \forall\alpha{:}\kappa.\nu, \forall\alpha{:}\kappa.\nu)$ and this obviously is of no use here. To come up with a proof we must strengthen the induction hypothesis such that, at this point, we not only know $\mathit{weight}(B', \tau_1, \tau_2) < \mathit{weight}(B'', \forall\alpha{:}\kappa.\nu, \forall\alpha{:}\kappa.\nu)$ but even $\mathit{weight}(B', \tau_1, \tau_2) < \mathit{weight}(B''', \tau_1', \tau_2')$, for all deeper decompositions (B''', τ_1', τ_2') of $(B'', \forall\alpha{:}\kappa.\nu, \forall\alpha{:}\kappa.\nu)$. The cases that do not use induction but directly construct a lighter decomposition therefore require part (3) of the following proposition.

Proposition 16 (Weight Property)**.** *Let X range over B and P.*

1. *If (X', τ_1', τ_2') is equal to or a deeper decomposition than (X, τ_1, τ_2), then $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X', \tau_1', \tau_2') \leq \mathit{size}(\tau_1) - \mathit{size}(\tau_1')$.*
2. *If $\tau = X[\tau_1][\tau_2]$, then $\mathit{weight}(X, \tau_1, \tau_2) > 0$.*
3. *Let (X_1, ν_1, ν_1) be equal to or a deeper decomposition than (B', ν, ν) and similar (X_2, ν_2, ν_2) equal to or a deeper decomposition than (P', p, p).*
   - *If $B' = B[\_ \to \tau_1][\_ \to \tau_2]$, then $\mathit{weight}(B[\nu \to \_][\nu \to \_], \tau_1, \tau_2) < \mathit{weight}(X_1, \nu_1, \nu_1)$.*
   - *If $B' = B[\_ \times \tau_1][\_ \times \tau_2]$, then $\mathit{weight}(B[\nu \times \_][\nu \times \_], \tau_1, \tau_2) < \mathit{weight}(X_1, \nu_1, \nu_1)$.*
   - *If $B' = B[\langle \_, \tau_1\rangle][\langle \_, \tau_2\rangle]$, then $\mathit{weight}(B[\langle \nu, \_\rangle][\langle \nu, \_\rangle], \tau_1, \tau_2) < \mathit{weight}(X_1, \nu_1, \nu_1)$.*
   - *If $B' = B[\langle \_, \tau_1\rangle][\langle \_, \tau_2\rangle]$, then $\mathit{weight}(B[\langle \nu, \_\rangle][\langle \nu, \_\rangle], \tau_1, \tau_2) < \mathit{weight}(X_1, \nu_1, \nu_1)$.*
   - *If $P' = P[\_\ \tau_1][\_\ \tau_2]$, then $\mathit{weight}(P[p\ \_][p\ \_], \tau_1, \tau_2) < \mathit{weight}(X_2, \nu_2, \nu_2)$.*

Proof. (1) by induction on the generation of X. (2) follows from (1). (3) follows from (1) and the definition of the size function.

1. Let (X', τ_1', τ_2') be equal to or a deeper decomposition than (X, τ_1, τ_2).
   - Case $X' = X''[\nu \times \_][\nu \times \_]$ and $(X'', \nu \times \tau_1', \nu \times \tau_2')$ is equal to or a deeper decomposition than (X, τ_1, τ_2):
     (a) By induction: $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X'', \nu \times \tau_1', \nu \times \tau_2') \leq \mathit{size}(\tau_1) - \mathit{size}(\nu \times \tau_1')$
     (b) By definition of *weight* and *size*: $\mathit{weight}(X'', \nu \times \tau_1', \nu \times \tau_2') - \mathit{weight}(X''[\nu \times \_][\nu \times \_], \tau_1', \tau_2') = \mathit{size}(\nu) + \mathit{size}(\tau_1') < 2 \cdot \mathit{size}(\nu) + \mathit{size}(\tau_1') = \mathit{size}(\nu \times \tau_1') - \mathit{size}(\tau_1')$
     (c) By (a) and (b): $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X''[\nu \times \_][\nu \times \_], \tau_1', \tau_2') \leq \mathit{size}(\tau_1) - \mathit{size}(\tau_1')$
2. Let (X, τ_1', τ_2') be a decomposition of e.
   (a) By definition of binary contexts: $e = \mathsf{tcase}\ v{:}\tau_1\ \mathsf{of}\ x{:}\tau_2\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$
   (b) By (a) and (1): $\mathit{weight}(\mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2, \tau_1, \tau_2) - \mathit{weight}(X, \tau_1', \tau_2') \leq \mathit{size}(\tau_1) - \mathit{size}(\tau_1')$
   (c) By (b): $\mathit{weight}(X, \tau_1', \tau_2') \geq \mathit{weight}(\mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2, \tau_1, \tau_2) - \mathit{size}(\tau_1) + \mathit{size}(\tau_1')$
   (d) By (c) and definition: $\mathit{weight}(X, \tau_1', \tau_2') \geq \mathit{size}(\tau_1')$
   (e) By (d) and definition: $\mathit{weight}(X, \tau_1', \tau_2') > 0$
3. Let (X_1, ν_1, ν_1) be equal to or a deeper decomposition than (B', ν, ν) and similar (X_2, p', p') equal to or a deeper decomposition than (P', p, p).

- Assume $B' = B[\_ \to \tau_1][\_ \to \tau_2]$.
  (a) By (1): $\mathit{weight}(B[\_ \to \tau_1][\_ \to \tau_2], \nu, \nu) - \mathit{weight}(X_1, \nu_1, \nu_1) \leq \mathit{size}(\nu) - \mathit{size}(\nu_1) < \mathit{size}(\nu)$
  (b) By definition: $\mathit{weight}(B[\_ \to \tau_1][\_ \to \tau_2], \nu, \nu) - \mathit{weight}(B[\nu \to \_][\nu \to \_], \tau_1, \tau_2) = \mathit{size}(\nu)$
  (c) By (a) and (b): $\mathit{weight}(B[\nu \to \_][\nu \to \_], \tau_1, \tau_2) < \mathit{weight}(X_1, \nu_1, \nu_1)$.

□

Definition 10 (Relation for decompositions)**.** *Let (X, τ_1, τ_2) and (X', τ_1', τ_2') be two decompositions of the same term (where X again ranges over P and B). We write $(X, \tau_1, \tau_2) \ll (X', \tau_1', \tau_2')$ iff $\mathit{weight}(X, \tau_1, \tau_2) < \mathit{weight}(X', \tau_1', \tau_2')$ and, for all decompositions $(X'', \tau_1'', \tau_2'')$ that are deeper than (X', τ_1', τ_2'), $\mathit{weight}(X, \tau_1, \tau_2) < \mathit{weight}(X'', \tau_1'', \tau_2'')$.*

Intuitively, $(X, \tau_1, \tau_2) \ll (X', \tau_1', \tau_2')$ holds iff (X', τ_1', τ_2') is a successor of (X, τ_1, τ_2) in the depth-first traversal but not deeper than (X, τ_1, τ_2). For example, if $e_1 = B_1[\nu \to \tau_1][\nu \to \tau_2]$, then $(B_1[\nu \to \_][\nu \to \_], \tau_1, \tau_2) \ll (B_1[\_ \to \tau_1][\_ \to \tau_2], \nu, \nu)$. However, if $e_2 = B_2[\forall\alpha{:}\Omega.\tau_1][\forall\alpha{:}\Omega.\tau_2]$, then $(B_2[\forall\alpha{:}\Omega.\_][\forall\alpha{:}\Omega.\_], \tau_1, \tau_2) \not\ll (B_2, \forall\alpha{:}\Omega.\tau_1, \forall\alpha{:}\Omega.\tau_2)$, because $B_2[\forall\alpha{:}\Omega.\_][\forall\alpha{:}\Omega.\_]$ is a descendant of B_2 and therefore $(B_2[\forall\alpha{:}\Omega.\_][\forall\alpha{:}\Omega.\_], \tau_1, \tau_2)$ deeper than $(B_2, \forall\alpha{:}\Omega.\tau_1, \forall\alpha{:}\Omega.\tau_2)$.

Proving that a closed and well-formed term $LB[\tau_1][\tau_2]$ can be reduced actually requires a few lemmata, namely parts (1) to (4) of the following proposition. While our reduction is deterministic, this is unfortunately not immediately clear from looking at these proofs.

Proposition 17 (Type Progress)**.**

1. *If $\cdot \vdash LCT[\tau_0] : \tau$ and τ_0 is not in weak head normal form, then $LCT[\tau_0] \longrightarrow e$.*
2. *If $e = LP[p][p]$, then there exists a decomposition $(B, \tau_1, \tau_2) \ll (P, p, p)$ or (P, p, p) is equal to or deeper than some decomposition (B, p', p').*
3. *If $e = LB[\nu][\nu]$, then $e \longrightarrow e'$ or there exists a decomposition $(B', \tau_1, \tau_2) \ll (B, \nu, \nu)$.*
4. *If $e = LP[q_1][q_2]$, then $e \longrightarrow e'$ or there exists a decomposition (B, τ_1, τ_2) of $P[q_1][q_2]$ with $\mathit{weight}(B, \tau_1, \tau_2) < \mathit{weight}(P, q_1, q_2)$.*
5. *If $\cdot \vdash LB[\tau_1][\tau_2] : \tau$, then $LB[\tau_1][\tau_2] \longrightarrow e$.*

Proof. (1) by structural induction on τ_0. (2) by induction on the generation of P. (3) by induction on the generation of B. (4) by structural induction on q_1. (5) by induction on $\mathit{weight}(B, \tau_1, \tau_2)$.

1. Let $\cdot \vdash LCT[\tau_0] : \tau$ where τ_0 is not in weak head normal form.
   - Case $\tau_0 = \tau_0'.1$:
     - Subcase $\tau_0' = \langle\tau_1, \tau_2\rangle$:
       (a) By RT-Proj1: $LCT[\tau_0] = LCT[\langle\tau_1, \tau_2\rangle.1] \longrightarrow LCT[\tau_1]$
     - Subcase τ_0' is not a type pair:
       (a) By assumption and Context Elimination: $\Gamma \vdash CT[\tau_0'.1] : \tau'$
       (b) By (a) and Type Context Elimination: $\Gamma, \Gamma' \vdash T[\tau_0'.1] : \kappa$
       (c) By (b) and Type Context Elimination: $\Gamma, \Gamma' \vdash \tau_0'.1 : \kappa'$
       (d) By (c) and inversion of K-Proj1: $\Gamma \vdash \tau_0' : \kappa' \times \kappa_2$
       (e) By (d) and definition of the kinding relation: $\tau_0' = \alpha$ or $\tau_0' = \langle\tau_1, \tau_2\rangle$ or $\tau_0' = \tau_1'\ \tau_2'$ or $\tau_0' = \tau_1'.1$ or $\tau_0' = \tau_1'.2$
       (f) $\tau_0' = \alpha$ is not possible since $\alpha.1$ is in weak head normal form

(g) By (e), (f), and subcase assumption: τ_0' is not in weak head normal form

(h) Let $T' = T[\_.1]$. Then: $LCT[\tau_0] = LCT'[\tau_0']$

(i) By (g), (h), assumption, and induction: $LCT[\tau_0] = LCT'[\tau_0'] \longrightarrow e$

3. Let $e = LB[\nu][\nu]$.

   - Case $B = \textsf{tcase } v{:}\_ \textsf{ of } x{:}\_ \textsf{ then } e_1 \textsf{ else } e_2$:

     (a) By R-Case1: $e \longrightarrow L[e_1[x := v]]$

   - Case $B = B'[\_ \to \tau_1][\_ \to \tau_2]$:

     (a) Let $B'' = B'[\nu \to \_][\nu \to \_]$. Then: $LB[\nu][\nu] = LB''[\tau_1][\tau_2]$

     (b) By (a) and Weight Property: $(B'', \tau_1, \tau_2) \ll (B, \nu, \nu)$

   - Case $B = B'[\nu' \to \_][\nu' \to \_]$:

     (a) By induction: $LB'[\nu' \to \nu][\nu' \to \nu] \longrightarrow e'$ or there exists a decomposition $(B'', \tau_1, \tau_2) \ll (B', \nu' \to \nu, \nu' \to \nu)$

       – If $LB'[\nu' \to \nu][\nu' \to \nu] \longrightarrow e'$, then: $LB[\nu][\nu] \longrightarrow e'$

       – If $(B'', \tau_1, \tau_2) \ll (B', \nu' \to \nu, \nu' \to \nu)$, then also $(B'', \tau_1, \tau_2) \ll (B, \nu, \nu)$, since (B, ν, ν) is deeper than $(B', \nu' \to \nu, \nu' \to \nu)$

□

Lemma 19 (Context Extension)**.** *If $L[e] \longrightarrow e'$ and e is not a* lazy *expression, then for all E exists an e'' such that $LE[e] \longrightarrow e''$.*

Proof. By case analysis on the applied reduction rule. □

Given all the previous preparations, the actual progress proof is now trivial. The only case that differs from the progress proof for the applicative order strategy is that e is a tcase expression. It uses part (5) of proposition 17 (Type Progress).

Theorem 4 (Progress)**.** *If $\cdot \vdash L[e] : \tau$ where e is neither a value nor a* lazy *expression, then $L[e] \longrightarrow e'$.*

Proof. By structural induction on e.

- Case $e \neq E[\textsf{lazy} \ldots] \wedge e = \textsf{tcase } e_0{:}\tau_0 \textsf{ of } x{:}\tau_0' \textsf{ then } e_1 \textsf{ else } e_2$:

  – Subcase e_0 is not a value:

    1. Let $E_0 = \textsf{tcase } \_{:}\tau_0 \textsf{ of } x{:}\tau_0' \textsf{ then } e_1 \textsf{ else } e_2$.
    2. By (1) and assumption: $\cdot \vdash LE_0[e_0] : \tau$
    3. By (2) and Context Elimination: $\cdot \vdash L[e_0] : \tau_0''$
    4. By assumption: e_0 is not a lazy expression
    5. By (3), (4), and induction: $L[e_0] \longrightarrow e_0'$
    6. By (4), (5), and Context Extension: $L[e] = LE_0[e_0] \longrightarrow e'$

  – Subcase $e_0 = v$:

    1. Let $B = \textsf{tcase } v{:}\_ \textsf{ of } x{:}\_ \textsf{ then } e_1 \textsf{ else } e_2$.
    2. By (1) and assumption: $\cdot \vdash LB[\tau_0][\tau_0'] : \tau$
    3. By (2) and Type Progress: $L[e] = LB[\tau_0][\tau_0'] \longrightarrow e'$

□

Chapter 4

Related work

Implicit lazy evaluation was thoroughly investigated by Zena Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler in the form of an untyped call-by-need calculus [14, 4, 5]. They in particular prove it sound and complete with respect to the call-by-name lambda calculus.

An overview of various reduction strategies for the untyped lambda calculus is given by Peter Sestoft in [23], though he does not consider any form of lazy evaluation. The different reduction relations are specified by a big-step semantics and can be classified as either *uniform* or *hybrid* strategies. Applicative order reduction to normal form, for instance, is uniform, because its definition does not use any other reduction relation. On the other hand, normal order reduction to normal form is hybrid: in order to always reduce the leftmost outermost redex first, it has to use call-by-name to reduce the left-hand side of an application to an abstraction. Our second strategy resembles normal order reduction because repeatedly descending and performing call-by-name reduction produces normal forms in the end (assuming the compared types turn out to be equal).

Equivalence checking on term level (in the absence of laziness) is introduced by Karl Crary in [8]. He shortly presents the normalize-and-compare approach but then extends his system in a way that requires a different algorithm. This sort of algorithm is called *type-driven*, because it depends on the types of the terms that are being checked. Its completeness proof requires the concept of *logical relations*, on which Crary then elaborates. In [25], Christopher A. Stone and Robert Harper investigate type equivalence checking in a system with singleton kinds, using a similar but more involved algorithm. They prove it correct and terminating. In comparison, our algorithm does not always terminate—due to the simple fact that the type-level normalization may trigger a diverging term-level reduction.

Intensional type analysis in statically typed languages is covered e.g. in [1] by Martín Abadi, Luca Cardelli, Benjamin Pierce, and Gordon Plotkin. Their typecase operator only applies to dynamics but is more expressive than ours: by using so-called *pattern variables*, it allows to match subexpressions of the embedded type. That way it is for example possible to extend our string representation function from section 2.4 to handle pairs as well, by recursively applying it to a dynamic of each component. A later paper, [2], deals with extending the mechanism of pattern variables to account for polymorphic types. In [13], Xavier Leroy and Michel Mauny show how to augment ML with dynamics, where the typecase is integrated into the usual pattern matching.

If a typecase operator is added to a calculus that uses existential types to encode abstract types, then it becomes possible to inspect such types and thus to break the abstraction layer. This can be repaired by dynamically generating new type names, as shown by Andreas Rossberg in [19].

The combination of lazy linking and dynamic type checking has been modeled before by Matthias Berg [6]—however, not in the presence of higher-order polymorphism but based on System F. Of course, in such a calculus type equivalence checking is trivial. While Berg is using the same approach regarding lazy evaluation, i.e., providing a lazy construct for opening packages, he does not define the operational semantics with the help of lazy and strict contexts. Instead, his reduction relation depends on a stack and a partial function that models a global state. The stack determines the lazy variable that is needed next and the state maps it to its associated term. The specification of the static semantics therefore requires additional definitions for the well-formedness of these configurations.

Extensive notes on the design and implementation of Alice ML can be found in [21] and [20].

Chapter 5

Conclusion and future work

We have given a calculus that models the integration of dynamic type checking with lazy linking into a language that provides higher-order polymorphism. It is based on system F_ω and its key ingredients are existential types, intensional type analysis in form of a simplistic typecase operator, and lazy evaluation in the form of a lazy construct for unpacking elements of existential types.

The evaluation of a typecase expression depends on the equivalence of type expressions, which may contain lazy variables. Formulating an appropriate algorithm for type equivalence checking requires reduction on type level. We have presented two strategies for this. The naive one normalizes the type expressions using applicative order reduction. Lazy variables are eliminated (by opening their associated package) when reduction encounters them. The second, smarter, algorithm achieves a much higher degree of laziness by interleaving call-by-name reduction with comparison of weak head normal forms. It is formulated with the help of special binary contexts that determine how to descent into type expressions of the same shape.

Both reduction relations have been proven sound. While the proofs for the first one were mostly straightforward, the progress proof for the smarter algorithm were much more involved and required a sophisticated weight-function for doing induction.

The alert reader might have noticed that our calculus completely ignored any form of eta equivalence regarding types. Neither the definitional type equivalence nor the type-level reduction strategies let us conclude that a type τ of kind $\kappa_1 \to \kappa_2$ is equal to $\lambda\alpha{:}\kappa_1.\tau\ \alpha$ (assuming $\alpha \notin ftv(\tau)$), altough they behave identically in any context. Similarly, if τ is a type pair, then it cannot be semantically distinguished from $\langle\tau.1, \tau.2\rangle$, but again our system considers both types to be different. It is trivial to add appropriate inference rules to the definition of the type equivalence relation. However, it is nontrivial to integrate eta equivalence into the type-level reduction. A more practical approach is to restrict the system such that it allows only types in long-eta normal form.

Even though our second strategy is clearly more lazy than the first, it is still not perfect. For instance it may happen that normalizing the left-hand sides of two arrow types requires multiple triggering while it is already clear that the types are different from looking at the right-hand side. Obviously, using a right-to-left reduction instead will not solve the general problem. It may be interesting, however, to formulate a breadth-first algorithm (and probably more complicated).

Various other ideas come to mind for future work:

- Providing a richer typecase, with support for pattern matching. This should not impose any major problems.
- Adding subtyping. Nothing would change for the applicative order type-reduction if the subtype check is performed after normalization. With regard to our second strategy, how-

ever, it is not clear at all how to handle subtyping (consider the contravariant part of arrow types).

- Incorporating Alice ML's concept of futures [17]. This requires additional machinery like *configurations* to handle concurrent computations.
- Allowing a type-erasure[9] implementation by representing run-time types as terms, thus making the model more realistic. Of course this makes the concept of lazy types redundant again.

Bibliography

[1] M. Abadi, L. Cardelli, B. Pierce, and G. Plotkin. Dynamic typing in a statically typed language. *ACM Transactions on Programming Languages and Systems*, 13(2):237–268, April 1991.

[2] M. Abadi, L. Cardelli, B. Pierce, G. Plotkin, and D. Rémy. Dynamic typing in polymorphic languages. In *Proceedings of the ACM SIGPLAN Workshop on ML and its Applications*, San Francisco, June 1992.

[3] Alice Team. The Alice system, 2003. `http://www.ps.uni-sb.de/alice/`.

[4] Z. M. Ariola and M. Felleisen. The call-by-need lambda calculus. *Journal of Functional Programming*, 7(3):265–301, 1997.

[5] Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. A call-by-need lambda calculus. In *Proceedings of 22nd Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*, pages 233–246, 1995.

[6] M. Berg. Polymorphic lambda calculus with dynamic types, Oct. 2004. `http://www.ps.uni-sb.de/~berg/fopra.html`.

[7] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, 17(4):471–522, 1985.

[8] K. Crary. Logical relations and a case study in equivalence checking. In B. C. Pierce, editor, *Advanced topics in types and programming languages.* MIT Press, 2005.

[9] K. Crary, S. Weirich, and J. G. Morrisett. Intensional polymorphism in type-erasure semantics. In *International Conference on Functional Programming*, pages 301–312, 1998.

[10] M. Felleisen and R. Hieb. A revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 103(2):235–271, 1992.

[11] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur.* PhD thesis, Université Paris VII, 1972.

[12] S. L. P. Jones. *The Implementation of Functional Programming Languages (Prentice-Hall International Series in Computer Science).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1987.

[13] X. Leroy and M. Mauny. Dynamics in ML. In J. Hughes, editor, *Functional Programming Languages and Computer Architecture, 5th ACM Conference*, volume 523, pages 406–426. Springer-Verlag, Berlin, Heidelberg, New York, 1991.

[14] J. Maraist, M. Odersky, and P. Wadler. The call-by-need lambda calculus, 1994.

[15] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML - Revised*. The MIT Press, May 1997.

[16] J. C. Mitchell and G. D. Plotkin. Abstract types have existential type. *ACM Trans. Program. Lang. Syst.*, 10(3):470–502, July 1988.

[17] J. Niehren, J. Schwinghammer, and G. Smolka. A concurrent lambda calculus with futures. In B. Gramlich, editor, *5th International Workshop on Frontiers in Combining Systems*, volume 3717 of *Lecture Notes in Computer Science*, pages 248–263. Springer, Aug. 2005.

[18] B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[19] A. Rossberg. Generativity and dynamic opacity for abstract types. In D. Miller, editor, *Proceedings of the 5th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, Uppsala, Sweden, Aug. 2003. ACM Press.

[20] A. Rossberg. The missing link - dynamic components for ML. In *11th International Conference on Functional Programming*, Portland, Oregon, USA, Sept. 2006. ACM Press.

[21] A. Rossberg. *Typed open programming*. PhD thesis, Programming Systems Lab, Universität des Saarlandes, 2006. To appear.

[22] A. Rossberg, D. L. Botlan, G. Tack, T. Brunklaus, and G. Smolka. *Alice Through the Looking Glass*, volume 5 of *Trends in Functional Programming*, pages 79–96. Intellect Books, Bristol, UK, ISBN 1-84150144-1, Munich, Germany, Feb. 2006.

[23] P. Sestoft. Demonstrating lambda calculus reduction. In *The essence of computation: complexity, analysis, transformation*, pages 420–435. Springer-Verlag New York, Inc., 2002.

[24] C. A. Stone. Type definitions. In B. C. Pierce, editor, *Advanced topics in types and programming languages*. MIT Press, 2005.

[25] C. A. Stone and R. Harper. Deciding type equivalence in a language with singleton kinds. In *ACM Symposium on Principles of Programming Languages (POPL), Boston, Massachusetts*, pages 214–227, 2000.

Appendix A

Proofs

A.1 The basic calculus

term variables	$x \in \mathit{Var}$
regular type variables	$\alpha \in \mathit{RTVar}$
lazy type variables	$\zeta \in \mathit{LTVar}$
terms	$e ::= x \mid \lambda x{:}\tau.e \mid e_1\ e_2 \mid \lambda\alpha{:}\kappa.e \mid e\ \tau \mid \langle e_1, e_2\rangle \mid \mathsf{let}\ \langle x_1, x_2\rangle = e_1\ \mathsf{in}\ e_2 \mid$ $\langle \tau_1, e\rangle{:}\tau_2 \mid \mathsf{let}\ \langle \alpha, x\rangle = e_1\ \mathsf{in}\ e_2 \mid \mathsf{lazy}\ \langle \zeta, x\rangle = e_1\ \mathsf{in}\ e_2$
type variables	$\xi ::= \alpha \mid \zeta$
types	$\tau ::= \xi \mid \tau_1 \to \tau_2 \mid \tau_1 \times \tau_2 \mid \forall\alpha{:}\kappa.\tau \mid \exists\alpha{:}\kappa.\tau \mid$ $\lambda\alpha{:}\kappa.\tau \mid \tau_1\ \tau_2 \mid \langle \tau_1, \tau_2\rangle \mid \tau.1 \mid \tau.2$
kinds	$\kappa ::= \Omega \mid \kappa_1 \to \kappa_2 \mid \kappa_1 \times \kappa_2$
values	$v ::= x \mid \lambda x{:}\tau.e \mid \lambda\alpha{:}\kappa.e \mid \langle v_1, v_2\rangle \mid \langle \tau_1, v\rangle{:}\tau_2$
evaluation contexts	$E ::= \_ \mid E\ e \mid (\lambda x{:}\tau.e)\ E \mid E\ \tau \mid \langle E, e\rangle \mid \langle v, E\rangle \mid \mathsf{let}\ \langle x_1, x_2\rangle = E\ \mathsf{in}\ e \mid$ $\langle \tau, E\rangle{:}\tau \mid \mathsf{let}\ \langle \alpha, x\rangle = E\ \mathsf{in}\ e$
lazy contexts	$L ::= \_ \mid \mathsf{lazy}\ \langle \zeta, x\rangle = e\ \mathsf{in}\ L$
strict contexts	$S ::= \_\ e \mid \_\ \tau \mid \mathsf{let}\ \langle x_1, x_2\rangle = \_\ \mathsf{in}\ e \mid \mathsf{let}\ \langle \alpha, x\rangle = \_\ \mathsf{in}\ e$

Figure A.1: The basic calculus (syntax)

Proposition 1 (Substitution)**.** *If $\xi_1 \neq \xi_2$ and $\xi_1 \notin ftv(\tau_2)$, then, for all τ and τ_1, $\tau[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$.*

Proof. By induction on the structure of τ.

- Case $\tau = \xi$:
  - Subcase $\xi = \xi_1$:
    1. $\xi_1[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau_1[\xi_2 := \tau_2] = \xi_1[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
  - Subcase $\xi = \xi_2$:
    1. $\xi_2[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]] = \tau_2[\xi_1 := \tau_1[\xi_2 := \tau_2]]$
    2. By assumption: $\tau_2[\xi_1 := \tau_1[\xi_2 := \tau_2]] = \tau_2$

Well-formedness of environments $\boxed{\Gamma \vdash \Box}$

$$(\textsc{E-Empty})\ \frac{}{\cdot \vdash \Box} \qquad (\textsc{E-Type})\ \frac{\Gamma \vdash \Box \qquad \alpha \notin \mathit{dom}(\Gamma)}{\Gamma, \alpha{:}\kappa \vdash \Box}$$

$$(\textsc{E-Term})\ \frac{\Gamma \vdash \tau : \Omega \qquad x \notin \mathit{dom}(\Gamma)}{\Gamma, x{:}\tau \vdash \Box}$$

Well-formedness of types $\boxed{\Gamma \vdash \tau : \kappa}$

$$(\textsc{K-Var})\ \frac{\Gamma \vdash \Box}{\Gamma \vdash \alpha : \Gamma(\alpha)} \qquad (\textsc{K-Arrow})\ \frac{\Gamma \vdash \tau_1 : \Omega \qquad \Gamma \vdash \tau_2 : \Omega}{\Gamma \vdash \tau_1 \to \tau_2 : \Omega}$$

$$(\textsc{K-Times})\ \frac{\Gamma \vdash \tau_1 : \Omega \qquad \Gamma \vdash \tau_2 : \Omega}{\Gamma \vdash \tau_1 \times \tau_2 : \Omega} \qquad (\textsc{K-Univ})\ \frac{\Gamma, \alpha{:}\kappa \vdash \tau : \Omega}{\Gamma \vdash \forall\alpha{:}\kappa.\tau : \Omega}$$

$$(\textsc{K-Exist})\ \frac{\Gamma, \alpha{:}\kappa \vdash \tau : \Omega}{\Gamma \vdash \exists\alpha{:}\kappa.\tau : \Omega} \qquad (\textsc{K-Abs})\ \frac{\Gamma, \alpha{:}\kappa \vdash \tau : \kappa'}{\Gamma \vdash \lambda\alpha{:}\kappa.\tau : \kappa \to \kappa'}$$

$$(\textsc{K-App})\ \frac{\Gamma \vdash \tau_1 : \kappa_2 \to \kappa \qquad \Gamma \vdash \tau_2 : \kappa_2}{\Gamma \vdash \tau_1\ \tau_2 : \kappa} \qquad (\textsc{K-Pair})\ \frac{\Gamma \vdash \tau_1 : \kappa_1 \qquad \Gamma \vdash \tau_2 : \kappa_2}{\Gamma \vdash \langle\tau_1, \tau_2\rangle : \kappa_1 \times \kappa_2}$$

$$(\textsc{K-Proj1})\ \frac{\Gamma \vdash \tau : \kappa_1 \times \kappa_2}{\Gamma \vdash \tau.1 : \kappa_1} \qquad (\textsc{K-Proj2})\ \frac{\Gamma \vdash \tau : \kappa_1 \times \kappa_2}{\Gamma \vdash \tau.2 : \kappa_2}$$

Figure A.2: The basic calculus (static semantics, part 1 of 2)

3. By (1) and (2): $\xi_2[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]] = \tau_2 = \xi_2[\xi_1 := \tau_1][\xi_2 := \tau_2]$

– Subcase $\xi \neq \xi_1$ and $\xi \neq \xi_2$:

1. $\xi[\xi_1 := \tau_1][\xi_2 := \tau_2] = \xi = \xi_1[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$

- Case $\tau = \tau' \to \tau''$:

1. $(\tau' \to \tau'')[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau'[\xi_1 := \tau_1][\xi_2 := \tau_2] \to \tau''[\xi_1 := \tau_1][\xi_2 := \tau_2]$
2. By induction: $\tau'[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau'[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
3. By induction: $\tau''[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau''[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
4. $(\tau' \to \tau'')[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]] =$
   $\tau'[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]] \to \tau''[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
5. By (1), (2), (3), and (4):
   $(\tau' \to \tau'')[\xi_1 := \tau_1][\xi_2 := \tau_2] = (\tau' \to \tau'')[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$

- Case $\tau = \tau' \times \tau''$:

1. $(\tau' \times \tau'')[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau'[\xi_1 := \tau_1][\xi_2 := \tau_2] \times \tau''[\xi_1 := \tau_1][\xi_2 := \tau_2]$
2. By induction: $\tau'[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau'[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
3. By induction: $\tau''[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau''[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
4. $(\tau' \times \tau'')[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]] =$
   $\tau'[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]] \times \tau''[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
5. By (1), (2), (3), and (4):
   $(\tau' \times \tau'')[\xi_1 := \tau_1][\xi_2 := \tau_2] = (\tau' \times \tau'')[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$

Well-formedness of terms $\boxed{\Gamma \vdash e : \tau}$

$$\text{(T-Equiv)}\ \frac{\Gamma \vdash e : \tau' \qquad \tau' \equiv \tau \qquad \Gamma \vdash \tau : \Omega}{\Gamma \vdash e : \tau} \qquad \text{(T-Var)}\ \frac{\Gamma \vdash \Box}{\Gamma \vdash x : \Gamma(x)}$$

$$\text{(T-Abs)}\ \frac{\Gamma, x{:}\tau \vdash e : \tau'}{\Gamma \vdash \lambda x{:}\tau.e : \tau \to \tau'} \qquad \text{(T-App)}\ \frac{\Gamma \vdash e_1 : \tau_2 \to \tau \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\ e_2 : \tau}$$

$$\text{(T-Gen)}\ \frac{\Gamma, \alpha{:}\kappa \vdash e : \tau}{\Gamma \vdash \lambda \alpha{:}\kappa.e : \forall \alpha{:}\kappa.\tau} \qquad \text{(T-Inst)}\ \frac{\Gamma \vdash e : \forall \alpha{:}\kappa.\tau' \qquad \Gamma \vdash \tau : \kappa}{\Gamma \vdash e\ \tau : \tau'[\alpha := \tau]}$$

$$\text{(T-Pair)}\ \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2} \qquad \text{(T-Proj)}\ \frac{\Gamma \vdash e_1 : \tau_1 \times \tau_2 \qquad \Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_2 : \tau}{\Gamma \vdash \mathsf{let}\ \langle x_1, x_2 \rangle = e_1\ \mathsf{in}\ e_2 : \tau}$$

$$\text{(T-Close)}\ \frac{\Gamma \vdash \tau : \kappa \qquad \Gamma \vdash e : \tau'[\alpha := \tau] \qquad \Gamma \vdash \exists \alpha{:}\kappa.\tau' : \Omega}{\Gamma \vdash \langle \tau, e \rangle{:}\exists \alpha{:}\kappa.\tau' : \exists \alpha{:}\kappa.\tau'}$$

$$\text{(T-Open)}\ \frac{\Gamma \vdash e_1 : \exists \alpha{:}\kappa.\tau' \qquad \Gamma, \alpha{:}\kappa, x{:}\tau' \vdash e_2 : \tau \qquad \alpha \notin ftv(\tau)}{\Gamma \vdash \mathsf{let}\ \langle \alpha, x \rangle = e_1\ \mathsf{in}\ e_2 : \tau}$$

$$\text{(T-Lazy)}\ \frac{\Gamma \vdash e_1 : \exists \alpha{:}\kappa.\tau' \qquad \Gamma, \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta] \vdash e_2 : \tau \qquad \zeta \notin ftv(\tau)}{\Gamma \vdash \mathsf{lazy}\ \langle \zeta, x \rangle = e_1\ \mathsf{in}\ e_2 : \tau}$$

Equivalence of types $\boxed{\tau \equiv \tau'}$

$$\text{(Q-Refl)}\ \frac{}{\tau \equiv \tau} \qquad \text{(Q-Symm)}\ \frac{\tau_2 \equiv \tau_1}{\tau_1 \equiv \tau_2} \qquad \text{(Q-Trans)}\ \frac{\tau_1 \equiv \tau_2 \qquad \tau_2 \equiv \tau_3}{\tau_1 \equiv \tau_3}$$

$$\text{(Q-Arrow)}\ \frac{\tau_1 \equiv \tau_1' \qquad \tau_2 \equiv \tau_2'}{\tau_1 \to \tau_2 \equiv \tau_1' \to \tau_2'} \qquad \text{(Q-Times)}\ \frac{\tau_1 \equiv \tau_1' \qquad \tau_2 \equiv \tau_2'}{\tau_1 \times \tau_2 \equiv \tau_1' \times \tau_2'}$$

$$\text{(Q-Univ)}\ \frac{\tau_1 \equiv \tau_2}{\forall \alpha{:}\kappa.\tau_1 \equiv \forall \alpha{:}\kappa.\tau_2} \qquad \text{(Q-Exist)}\ \frac{\tau_1 \equiv \tau_2}{\exists \alpha{:}\kappa.\tau_1 \equiv \exists \alpha{:}\kappa.\tau_2}$$

$$\text{(Q-Abs)}\ \frac{\tau_1 \equiv \tau_2}{\lambda \alpha{:}\kappa.\tau_1 \equiv \lambda \alpha{:}\kappa.\tau_2} \qquad \text{(Q-App)}\ \frac{\tau_1 \equiv \tau_2 \qquad \tau_1' \equiv \tau_2'}{\tau_1\ \tau_2 \equiv \tau_1'\ \tau_2'}$$

$$\text{(Q-Beta)}\ \frac{}{(\lambda \alpha{:}\kappa.\tau_1)\ \tau_2 \equiv \tau_1[\alpha := \tau_2]} \qquad \text{(Q-Pair)}\ \frac{\tau_1 \equiv \tau_1' \qquad \tau_2 \equiv \tau_2'}{\langle \tau_1, \tau_2 \rangle \equiv \langle \tau_1', \tau_2' \rangle}$$

$$\text{(Q-Proj1a)}\ \frac{\tau \equiv \tau'}{\tau.1 \equiv \tau'.1} \qquad \text{(Q-Proj1b)}\ \frac{}{\langle \tau_1, \tau_2 \rangle.1 \equiv \tau_1}$$

$$\text{(Q-Proj2a)}\ \frac{\tau \equiv \tau'}{\tau.2 \equiv \tau'.2} \qquad \text{(Q-Proj2b)}\ \frac{}{\langle \tau_1, \tau_2 \rangle.2 \equiv \tau_2}$$

Figure A.3: The basic calculus (static semantics, part 2 of 2)

- Case $\tau = \forall\alpha{:}\kappa.\tau'$ (w.l.o.g. $\alpha \notin \{\xi_1, \xi_2\}$ and $\alpha \notin ftv(\tau_1) \cup ftv(\tau_2)$):

  1. $(\forall\alpha{:}\kappa.\tau')[\xi_1 := \tau_1][\xi_2 := \tau_2] = \forall\alpha{:}\kappa.\tau'[\xi_1 := \tau_1][\xi_2 := \tau_2]$
  2. By induction: $\tau'[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau'[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
  3. $(\forall\alpha{:}\kappa.\tau')[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]] = \forall\alpha{:}\kappa.\tau'[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
  4. By (1), (2), and (3): $(\forall\alpha{:}\kappa.\tau')[\xi_1 := \tau_1][\xi_2 := \tau_2] = (\forall\alpha{:}\kappa.\tau')[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$

- Case $\tau = \exists\alpha{:}\kappa.\tau'$ (w.l.o.g. $\alpha \notin \{\xi_1, \xi_2\}$ and $\alpha \notin ftv(\tau_1) \cup ftv(\tau_2)$):

  1. $(\exists\alpha{:}\kappa.\tau')[\xi_1 := \tau_1][\xi_2 := \tau_2] = \exists\alpha{:}\kappa.\tau'[\xi_1 := \tau_1][\xi_2 := \tau_2]$
  2. By induction: $\tau'[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau'[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
  3. $(\exists\alpha{:}\kappa.\tau')[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]] = \exists\alpha{:}\kappa.\tau'[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
  4. By (1), (2), and (3): $(\exists\alpha{:}\kappa.\tau')[\xi_1 := \tau_1][\xi_2 := \tau_2] = (\exists\alpha{:}\kappa.\tau')[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$

- Case $\tau = \lambda\alpha{:}\kappa.\tau'$ (w.l.o.g. $\alpha \notin \{\xi_1, \xi_2\}$ and $\alpha \notin ftv(\tau_1) \cup ftv(\tau_2)$):

  1. $(\lambda\alpha{:}\kappa.\tau')[\xi_1 := \tau_1][\xi_2 := \tau_2] = \lambda\alpha{:}\kappa.\tau'[\xi_1 := \tau_1][\xi_2 := \tau_2]$
  2. By induction: $\tau'[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau'[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
  3. $(\lambda\alpha{:}\kappa.\tau')[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]] = \lambda\alpha{:}\kappa.\tau'[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
  4. By (1), (2), and (3): $(\lambda\alpha{:}\kappa.\tau')[\xi_1 := \tau_1][\xi_2 := \tau_2] = (\lambda\alpha{:}\kappa.\tau')[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$

- Case $\tau = \tau'\ \tau''$:

  1. $(\tau'\ \tau'')[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau'[\xi_1 := \tau_1][\xi_2 := \tau_2]\ \tau''[\xi_1 := \tau_1][\xi_2 := \tau_2]$
  2. By induction: $\tau'[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau'[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
  3. By induction: $\tau''[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau''[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
  4. $(\tau'\ \tau'')[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]] =$
     $\tau'[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]\ \tau''[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
  5. By (1), (2), (3), and (4):
     $(\tau'\ \tau'')[\xi_1 := \tau_1][\xi_2 := \tau_2] = (\tau'\ \tau'')[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$

- Case $\tau = \langle\tau', \tau''\rangle$:

  1. $\langle\tau', \tau''\rangle[\xi_1 := \tau_1][\xi_2 := \tau_2] = \langle\tau'[\xi_1 := \tau_1][\xi_2 := \tau_2], \tau''[\xi_1 := \tau_1][\xi_2 := \tau_2]\rangle$
  2. By induction: $\tau'[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau'[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
  3. By induction: $\tau''[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau''[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
  4. $\langle\tau', \tau''\rangle[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]] =$
     $\langle\tau'[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]], \tau''[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]\rangle$
  5. By (1), (2), (3), and (4):
     $\langle\tau', \tau''\rangle[\xi_1 := \tau_1][\xi_2 := \tau_2] = \langle\tau', \tau''\rangle[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$

- Case $\tau = \tau'.1$:

  1. $(\tau'.1)[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau'[\xi_1 := \tau_1][\xi_2 := \tau_2].1$
  2. By induction: $\tau'[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau'[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
  3. $(\tau'.1)[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]] = \tau'[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]].1$

4. By (1), (2), and (3): $(\tau'.1)[\xi_1 := \tau_1][\xi_2 := \tau_2] = (\tau'.1)[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$

- Case $\tau = \tau'.2$:
  1. $(\tau'.2)[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau'[\xi_1 := \tau_1][\xi_2 := \tau_2].2$
  2. By induction: $\tau'[\xi_1 := \tau_1][\xi_2 := \tau_2] = \tau'[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$
  3. $(\tau'.2)[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]] = \tau'[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]].2$
  4. By (1), (2), and (3): $(\tau'.2)[\xi_1 := \tau_1][\xi_2 := \tau_2] = (\tau'.2)[\xi_2 := \tau_2][\xi_1 := \tau_1[\xi_2 := \tau_2]]$

□

Proposition 2 (Environment Validity)**.**

1. If $\Gamma \vdash \tau : \kappa$*, then* $\Gamma \vdash \Box$*.*

2. If $\Gamma \vdash e : \tau$*, then* $\Gamma \vdash \Box$*.*

Proof. By induction on the derivation.

1. Let $\Gamma \vdash \tau : \kappa$.
   - Case K-Var:
     (a) By inversion: $\Gamma \vdash \Box$
   - Case K-Arrow: $\tau = \tau_1 \to \tau_2$
     (a) By inversion: $\Gamma \vdash \tau_1 : \Omega$
     (b) By (a) and induction: $\Gamma \vdash \Box$
   - Case K-Times: $\tau = \tau_1 \times \tau_2$
     (a) By inversion: $\Gamma \vdash \tau_1 : \Omega$
     (b) By (a) and induction: $\Gamma \vdash \Box$
   - Case K-Univ: $\tau = \forall\alpha{:}\kappa'.\tau'$
     (a) By inversion: $\Gamma, \alpha{:}\kappa' \vdash \tau' : \Omega$
     (b) By (a) and induction: $\Gamma, \alpha{:}\kappa' \vdash \Box$
     (c) By (b) and inversion of E-Type: $\Gamma \vdash \Box$
   - Case K-Exist: $\tau = \exists\alpha{:}\kappa'.\tau'$
     (a) By inversion: $\Gamma, \alpha{:}\kappa' \vdash \tau' : \Omega$
     (b) By (a) and induction: $\Gamma, \alpha{:}\kappa' \vdash \Box$
     (c) By (b) and inversion of E-Type: $\Gamma \vdash \Box$
   - Case K-Abs: $\tau = \lambda\alpha{:}\kappa'.\tau'$
     (a) By inversion: $\Gamma, \alpha{:}\kappa' \vdash \tau' : \kappa''$
     (b) By (a) and induction: $\Gamma, \alpha{:}\kappa' \vdash \Box$
     (c) By (b) and inversion of E-Type: $\Gamma \vdash \Box$
   - Case K-App: $\tau = \tau_1\ \tau_2$
     (a) By inversion: $\Gamma \vdash \tau_2 : \kappa'$
     (b) By (a) and induction: $\Gamma \vdash \Box$
   - Case K-Pair: $\tau = \langle\tau_1, \tau_2\rangle$
     (a) By inversion: $\Gamma \vdash \tau_1 : \kappa_1$
     (b) By (a) and induction: $\Gamma \vdash \Box$

- Case K-Proj1: $\tau = \tau'.1$
  (a) By inversion: $\Gamma \vdash \tau' : \kappa_1 \times \kappa_2$
  (b) By (a) and induction: $\Gamma \vdash \Box$
- Case K-Proj2: $\tau = \tau'.2$
  (a) By inversion: $\Gamma \vdash \tau' : \kappa_1 \times \kappa_2$
  (b) By (a) and induction: $\Gamma \vdash \Box$

2. Let $\Gamma \vdash e : \tau$.

- Case T-Equiv:
  (a) By inversion: $\Gamma \vdash e : \tau'$
  (b) By (a) and induction: $\Gamma \vdash \Box$
- Case T-Var:
  (a) By inversion: $\Gamma \vdash \Box$
- Case T-Abs: $e = \lambda x{:}\tau_1.e_1$
  (a) By inversion: $\Gamma, x{:}\tau_1 \vdash e_1 : \tau_2$
  (b) By (a) and induction: $\Gamma, x{:}\tau_1 \vdash \Box$
  (c) By (b) and inversion of E-Term: $\Gamma \vdash \tau_1 : \Omega$
  (d) By (c) and (1): $\Gamma \vdash \Box$
- Case T-App: $e = e_1\ e_2$
  (a) By inversion: $\Gamma \vdash e_2 : \tau'$
  (b) By (a) and induction: $\Gamma \vdash \Box$
- Case T-Gen: $e = \lambda \alpha{:}\kappa.e_1$
  (a) By inversion: $\Gamma, \alpha{:}\kappa \vdash e_1 : \tau_1$
  (b) By (a) and induction: $\Gamma, \alpha{:}\kappa \vdash \Box$
  (c) By (b) and inversion of E-Type: $\Gamma \vdash \Box$
- Case T-Inst: $e = e_1\ \tau_2$
  (a) By inversion: $\Gamma \vdash \tau_2 : \kappa$
  (b) By (a) and (1): $\Gamma \vdash \Box$
- Case T-Pair: $e = \langle e_1, e_2 \rangle$
  (a) By inversion: $\Gamma \vdash e_1 : \tau_1$
  (b) By (a) and induction: $\Gamma \vdash \Box$
- Case T-Proj: $e = \mathsf{let}\ \langle x_1, x_2 \rangle = e_1\ \mathsf{in}\ e_2$
  (a) By inversion: $\Gamma \vdash e_1 : \tau_1 \times \tau_2$
  (b) By (a) and induction: $\Gamma \vdash \Box$
- Case T-Close: $e = \langle \tau_1, e' \rangle{:}\tau$
  (a) By inversion: $\Gamma \vdash \tau : \Omega$
  (b) By (a) and (1): $\Gamma \vdash \Box$
- Case T-Open: $e = \mathsf{let}\ \langle \alpha, x \rangle = e_1\ \mathsf{in}\ e_2$
  (a) By inversion: $\Gamma \vdash e_1 : \exists \alpha{:}\kappa.\tau'$
  (b) By (a) and induction: $\Gamma \vdash \Box$
- Case T-Lazy: $e = \mathsf{lazy}\ \langle \zeta, x \rangle = e_1\ \mathsf{in}\ e_2$

(a) By inversion: $\Gamma \vdash e_1 : \exists\alpha{:}\kappa.\tau'$
(b) By (a) and induction: $\Gamma \vdash \Box$

- Case T-Case: $e = \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$

  (a) By inversion: $\Gamma \vdash e_0 : \tau_0$
  (b) By (a) and induction: $\Gamma \vdash \Box$

□

Proposition 3 (Subenvironment Validity). *If* $\Gamma_1, \Gamma_2 \vdash \Box$, *then* $\Gamma_1 \vdash \Box$.

Proof. By induction on the structure of Γ_2.

- Case $\Gamma_2 = \cdot$:

  1. By assumption: $\Gamma_1 \vdash \Box$

- Case $\Gamma_2 = \Gamma_2', \xi{:}\kappa$:

  1. By inversion of E-Type: $\Gamma_1, \Gamma_2' \vdash \Box$
  2. By (1) and induction: $\Gamma_1 \vdash \Box$

- Case $\Gamma_2 = \Gamma_2', x{:}\tau$:

  1. By inversion of E-Term: $\Gamma_1, \Gamma_2' \vdash \tau : \Omega$
  2. By (1) and Environment Validity: $\Gamma_1, \Gamma_2' \vdash \Box$
  3. By (2) and induction: $\Gamma_1 \vdash \Box$

□

Proposition 4 (Variable Containment).

1. If $\Gamma \vdash \tau : \kappa$, *then* $ftv(\tau) \subseteq dom(\Gamma)$.

2. If $\Gamma \vdash e : \tau$, *then* $fv(e) \cup ftv(\tau) \subseteq dom(\Gamma)$.

Proof. By induction on the derivation.

1. Let $\Gamma \vdash \tau : \kappa$.

   - Case K-Var: $\tau = \xi$

     (a) By inversion: $\kappa = \Gamma(\xi)$, hence $\xi \in dom(\Gamma)$
     (b) By definition: $ftv(\xi) = \{\xi\}$
     (c) By (b) and (a): $ftv(\xi) \subseteq dom(\Gamma)$

   - Case K-Arrow: $\tau = \tau_1 \to \tau_2$

     (a) By inversion: $\Gamma \vdash \tau_1 : \Omega$ and $\Gamma \vdash \tau_2 : \Omega$
     (b) By (a) and induction: $ftv(\tau_1) \subseteq dom(\Gamma)$ and $ftv(\tau_2) \subseteq dom(\Gamma)$
     (c) By definition: $ftv(\tau_1 \to \tau_2) = ftv(\tau_1) \cup ftv(\tau_2)$
     (d) By (c) and (b): $ftv(\tau_1 \to \tau_2) \subseteq dom(\Gamma)$

   - Case K-Times: $\tau = \tau_1 \times \tau_2$

     (a) By inversion: $\Gamma \vdash \tau_1 : \Omega$ and $\Gamma \vdash \tau_2 : \Omega$
     (b) By (a) and induction: $ftv(\tau_1) \subseteq dom(\Gamma)$ and $ftv(\tau_2) \subseteq dom(\Gamma)$
     (c) By definition: $ftv(\tau_1 \times \tau_2) = ftv(\tau_1) \cup ftv(\tau_2)$

(d) By (c) and (b): $ftv(\tau_1 \times \tau_2) \subseteq dom(\Gamma)$

- Case K-Univ: $\tau = \forall\alpha{:}\kappa'.\tau'$
  (a) By inversion: $\Gamma, \alpha{:}\kappa' \vdash \tau' : \Omega$
  (b) By (a) and induction: $ftv(\tau') \subseteq dom(\Gamma, \alpha{:}\kappa')$
  (c) By (a) and Environment Validity: $\Gamma, \alpha{:}\kappa' \vdash \Box$
  (d) By (c) and inversion of E-Type: $\alpha \notin dom(\Gamma)$
  (e) By (b) and (d): $ftv(\tau') - \{\alpha\} \subseteq dom(\Gamma)$
  (f) By definition: $ftv(\forall\alpha{:}\kappa'.\tau') = ftv(\tau') - \{\alpha\}$
  (g) By (f) and (e): $ftv(\forall\alpha{:}\kappa'.\tau') \subseteq dom(\Gamma)$
- Case K-Exist: $\tau = \exists\alpha{:}\kappa'.\tau'$
  (a) By inversion: $\Gamma, \alpha{:}\kappa' \vdash \tau' : \Omega$
  (b) By (a) and induction: $ftv(\tau') \subseteq dom(\Gamma, \alpha{:}\kappa')$
  (c) By (a) and Environment Validity: $\Gamma, \alpha{:}\kappa' \vdash \Box$
  (d) By (c) and inversion of E-Type: $\alpha \notin dom(\Gamma)$
  (e) By (b) and (d): $ftv(\tau') - \{\alpha\} \subseteq dom(\Gamma)$
  (f) By definition: $ftv(\exists\alpha{:}\kappa'.\tau') = ftv(\tau') - \{\alpha\}$
  (g) By (f) and (e): $ftv(\exists\alpha{:}\kappa'.\tau') \subseteq dom(\Gamma)$
- Case K-Abs: $\tau = \lambda\alpha{:}\kappa'.\tau'$
  (a) By inversion: $\Gamma, \alpha{:}\kappa' \vdash \tau' : \kappa''$
  (b) By (a) and induction: $ftv(\tau') \subseteq dom(\Gamma, \alpha{:}\kappa')$
  (c) By (a) and Environment Validity: $\Gamma, \alpha{:}\kappa' \vdash \Box$
  (d) By (c) and inversion of E-Type: $\alpha \notin dom(\Gamma)$
  (e) By (b) and (d): $ftv(\tau') - \{\alpha\} \subseteq dom(\Gamma)$
  (f) By definition: $ftv(\lambda\alpha{:}\kappa'.\tau') = ftv(\tau') - \{\alpha\}$
  (g) By (f) and (e): $ftv(\lambda\alpha{:}\kappa'.\tau') \subseteq dom(\Gamma)$
- Case K-App: $\tau = \tau_1\ \tau_2$
  (a) By inversion: $\Gamma \vdash \tau_1 : \kappa' \to \kappa$ and $\Gamma \vdash \tau_2 : \kappa'$
  (b) By (a) and induction: $ftv(\tau_1) \subseteq dom(\Gamma)$ and $ftv(\tau_2) \subseteq dom(\Gamma)$
  (c) By definition: $ftv(\tau_1\ \tau_2) = ftv(\tau_1) \cup ftv(\tau_2)$
  (d) By (c) and (b): $ftv(\tau_1\ \tau_2) \subseteq dom(\Gamma)$
- Case K-Pair: $\tau = \langle\tau_1, \tau_2\rangle$
  (a) By inversion: $\Gamma \vdash \tau_1 : \kappa_1$ and $\Gamma \vdash \tau_2 : \kappa_2$
  (b) By (a) and induction: $ftv(\tau_1) \subseteq dom(\Gamma)$ and $ftv(\tau_2) \subseteq dom(\Gamma)$
  (c) By definition: $ftv(\langle\tau_1, \tau_2\rangle) = ftv(\tau_1) \cup ftv(\tau_2)$
  (d) By (c) and (b): $ftv(\langle\tau_1, \tau_2\rangle) \subseteq dom(\Gamma)$
- Case K-Proj1: $\tau = \tau'.1$
  (a) By inversion: $\Gamma \vdash \tau' : \kappa_1 \times \kappa_2$
  (b) By (a) and induction: $ftv(\tau') \subseteq dom(\Gamma)$
  (c) By definition: $ftv(\tau'.1) = ftv(\tau')$
  (d) By (c) and (b): $ftv(\tau'.1) \subseteq dom(\Gamma)$
- Case K-Proj2: $\tau = \tau'.2$
  (a) By inversion: $\Gamma \vdash \tau' : \kappa_1 \times \kappa_2$
  (b) By (a) and induction: $ftv(\tau') \subseteq dom(\Gamma)$

   (c) By definition: $ftv(\tau'.2) = ftv(\tau')$
   (d) By (c) and (b): $ftv(\tau'.2) \subseteq dom(\Gamma)$

2. Let $\Gamma \vdash e : \tau$.
   - Case T-Equiv:
     (a) By inversion: $\Gamma \vdash e : \tau'$ and $\Gamma \vdash \tau : \Omega$
     (b) By (a) and induction: $fv(e) \subseteq dom(\Gamma)$
     (c) By (a) and (1): $ftv(\tau) \subseteq dom(\Gamma)$
     (d) By (b) and (c): $fv(e) \cup ftv(\tau) \subseteq dom(\Gamma)$
   - Case T-Var: $e = x$
     (a) By inversion: $\Gamma \vdash \square$ and $\tau = \Gamma(x)$
     (b) By (a): $x \in dom(\Gamma)$, hence $\Gamma = \Gamma_1, x{:}\tau, \Gamma_2$
     (c) By definition: $fv(x) = \{x\}$
     (d) By (c) and (b): $fv(x) \subseteq dom(\Gamma)$
     (e) By (a), (b), and Subenvironment Validity: $\Gamma_1, x{:}\tau \vdash \square$
     (f) By (e) and inversion of E-Term: $\Gamma_1 \vdash \tau\Omega$
     (g) By (f) and (1): $ftv(\tau) \subseteq dom(\Gamma_1)$
     (h) By (g) and (b): $ftv(\tau) \subseteq dom(\Gamma)$
     (i) By (d) and (h): $fv(x) \cup ftv(\tau) \subseteq dom(\Gamma)$
   - Case T-Abs: $e = \lambda x{:}\tau_1.e_1$ and $\tau = \tau_1 \to \tau_2$
     (a) By inversion: $\Gamma, x{:}\tau_1 \vdash e_1 : \tau_2$
     (b) By (a) and induction: $fv(e_1) \cup ftv(\tau_2) \subseteq dom(\Gamma, x{:}\tau_1)$
     (c) By (a) and Environment Validity: $\Gamma, x{:}\tau_1 \vdash \square$
     (d) By (c) and inversion of E-Term: $x \notin dom(\Gamma)$ and $\Gamma \vdash \tau_1 : \Omega$
     (e) By (b) and (d): $fv(e_1) - \{x\} \subseteq dom(\Gamma)$
     (f) By definition: $fv(\lambda x{:}\tau_1.e_1) = fv(e_1) - \{x\}$
     (g) By (d) and (1): $ftv(\tau_1) \subseteq dom(\Gamma)$
     (h) By (b): $ftv(\tau_2) \subseteq dom(\Gamma)$
     (i) By definition: $ftv(\tau_1 \to \tau_2) = ftv(\tau_1) \cup ftv(\tau_2)$
     (j) By (f), (e), (i), (g), and (h): $fv(\lambda x{:}\tau_1.e_1) \cup ftv(\tau_1 \to \tau_2) \subseteq dom(\Gamma)$
   - Case T-App: $e = e_1\ e_2$
     (a) By inversion: $\Gamma \vdash e_1 : \tau' \to \tau$ and $\Gamma \vdash e_2 : \tau'$
     (b) By (a) and induction: $fv(e_1) \cup ftv(\tau' \to \tau) \subseteq dom(\Gamma)$ and $fv(e_2) \subseteq dom(\Gamma)$
     (c) By definition: $fv(e_1\ e_2) = fv(e_1) \cup fv(e_2)$
     (d) By (b) and (c): $fv(e_1\ e_2) \subseteq dom(\Gamma)$
     (e) By definition: $ftv(\tau' \to \tau) = ftv(\tau') \cup ftv(\tau)$
     (f) By (e) and (b): $ftv(\tau) \subseteq dom(\Gamma)$
     (g) By (d) and (f): $fv(e_1\ e_2) \cup ftv(\tau) \subseteq dom(\Gamma)$
   - Case T-Gen: $e = \lambda\alpha{:}\kappa.e_1$ and $\tau = \forall\alpha{:}\kappa.\tau'$
     (a) By inversion: $\Gamma, \alpha{:}\kappa \vdash e_1 : \tau'$
     (b) By (a) and induction: $fv(e_1) \cup ftv(\tau') \subseteq dom(\Gamma, \alpha{:}\kappa)$
     (c) By definition: $fv(\lambda\alpha{:}\kappa.e_1) = fv(e_1)$
     (d) By (c) and (b): $fv(\lambda\alpha{:}\kappa.e_1) \subseteq dom(\Gamma)$

(e) By (a) and Environment Validity: $\Gamma, \alpha{:}\kappa \vdash \Box$
(f) By (e) and inversion of E-Type: $\alpha \notin dom(\Gamma)$
(g) By (b) and (f): $ftv(\tau') - \{\alpha\} \subseteq dom(\Gamma)$
(h) By definition: $ftv(\forall\alpha{:}\kappa.\tau') = ftv(\tau') - \{\alpha\}$
(i) By (g) and (f): $ftv(\forall\alpha{:}\kappa.\tau') \subseteq dom(\Gamma)$
(j) By (d) and (h): $fv(\lambda\alpha{:}\kappa.e_1) \cup ftv(\forall\alpha{:}\kappa.\tau') \subseteq dom(\Gamma)$

- Case T-Inst: $e = e_1\ \tau_2$ and $\tau = \tau'[\alpha := \tau_2]$

(a) By inversion: $\Gamma \vdash e_1 : \forall\alpha{:}\kappa.\tau'$ and $\Gamma \vdash \tau_2 : \kappa$
(b) By (a) and induction: $fv(e_1) \cup ftv(\forall\alpha{:}\kappa.\tau') \subseteq dom(\Gamma)$
(c) By definition: $fv(e_1\ \tau_2) = fv(e_1)$
(d) By (c) and (b): $fv(e_1\ \tau_2) \subseteq dom(\Gamma)$
(e) By definition: $ftv(\forall\alpha{:}\kappa.\tau') = ftv(\tau') - \{\alpha\}$
(f) By (b) and (e): $ftv(\tau') - \{\alpha\} \subseteq dom(\Gamma)$
(g) By (a) and (1): $ftv(\tau_2) \subseteq dom(\Gamma)$
(h) By (f) and (g): $ftv(\tau'[\alpha := \tau_2]) \subseteq dom(\Gamma)$
(i) By (d) and (h): $fv(e_1\ \tau_2) \cup ftv(\tau'[\alpha := \tau_2]) \subseteq dom(\Gamma)$

- Case T-Pair: $e = \langle e_1, e_2\rangle$ and $\tau = \tau_1 \times \tau_2$

(a) By inversion: $\Gamma \vdash e_1 : \tau_1$ and $\Gamma \vdash e_2 : \tau_2$
(b) By (a) and induction: $fv(e_1) \cup ftv(\tau_1) \subseteq dom(\Gamma)$ and $fv(e_2) \cup ftv(\tau_2) \subseteq dom(\Gamma)$
(c) By definition: $fv(\langle e_1, e_2\rangle) = fv(e_1) \cup fv(e_2)$
(d) By (c) and (b): $fv(\langle e_1, e_2\rangle) \subseteq dom(\Gamma)$
(e) By definition: $ftv(\tau_1 \times \tau_2) = ftv(\tau_1) \cup ftv(\tau_2)$
(f) By (e) and (b): $ftv(\tau \times \tau_2) \subseteq dom(\Gamma)$
(g) By (d) and (f): $fv(\langle e_1, e_2\rangle) \cup ftv(\tau \times \tau_2) \subseteq dom(\Gamma)$

- Case T-Proj: $e = \mathsf{let}\ \langle x_1, x_2\rangle = e_1\ \mathsf{in}\ e_2$

(a) By inversion: $\Gamma \vdash e_1 : \tau_1 \times \tau_2$ and $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_2 : \tau$
(b) By (a) and induction: $fv(e_1) \cup ftv(\tau_1 \times \tau_2) \subseteq dom(\Gamma)$ and $fv(e_2) \cup ftv(\tau) \subseteq dom(\Gamma, x_1{:}\tau_1, x_2{:}\tau_2)$
(c) By (a) and Environment Validity: $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash \Box$
(d) By (c) and inversion of E-Term: $\Gamma, x_1{:}\tau_1 \vdash \tau_2 : \Omega$ and $x_2 \notin dom(\Gamma, x_1{:}\tau_2)$
(e) By (d) and Environment Validity: $\Gamma, x_1{:}\tau_1 \vdash \Box$
(f) By (e) and inversion of E-Term: $x_1 \notin dom(\Gamma)$
(g) By (b), (d), and (f): $fv(e_2) - \{x_1, x_2\} \subseteq dom(\Gamma)$
(h) By (b) and (g): $fv(e_1) \cup (fv(e_2) - \{x_1, x_2\}) \subseteq dom(\Gamma)$
(i) By definition: $fv(\mathsf{let}\ \langle x_1, x_2\rangle = e_1\ \mathsf{in}\ e_2) = fv(e_1) \cup (fv(e_2) - \{x_1, x_2\})$
(j) By (i) and (h): $fv(\mathsf{let}\ \langle x_1, x_2\rangle = e_1\ \mathsf{in}\ e_2) \subseteq dom(\Gamma)$
(k) By (b): $ftv(\tau) \subseteq dom(\Gamma)$
(l) By (j) and (k): $fv(\mathsf{let}\ \langle x_1, x_2\rangle = e_1\ \mathsf{in}\ e_2) \cup ftv(\tau) \subseteq dom(\Gamma)$

- Case T-Close: $e = \langle \tau_1, e'\rangle{:}\tau$ where $\tau = \exists\alpha{:}\kappa.\tau_2$

(a) By inversion: $\Gamma \vdash e' : \tau_2[\alpha := \tau_1]$ and $\Gamma \vdash \exists\alpha{:}\kappa.\tau_2 : \Omega$
(b) By (a) and induction: $fv(e') \cup ftv(\tau_2[\alpha := \tau_1] \subseteq dom(\Gamma)$
(c) By definition: $fv(\langle \tau_1, e'\rangle{:}\tau) = fv(e')$
(d) By (c) and (b): $fv(\langle \tau_1, e'\rangle{:}\tau) \subseteq dom(\Gamma)$
(e) By (a) and (1): $ftv(\alpha{:}\kappa.\tau_2) \subseteq dom(\Gamma)$

(f) By (d) and (e): $fv(\langle\tau_1, e'\rangle{:}\tau) \cup ftv(\alpha{:}\kappa.\tau_2) \subseteq dom(\Gamma)$

- Case T-Open: $e = \mathsf{let}\ \langle\alpha, x\rangle\ \texttt{=}\ e_1\ \mathsf{in}\ e_2$
  (a) By inversion: $\Gamma \vdash e_1 : \exists\alpha{:}\kappa.\tau'$ and $\Gamma, \alpha{:}\kappa, x{:}\tau' \vdash e_2 : \tau$ where $\alpha \notin ftv(\tau)$
  (b) By (a) and induction: $fv(e_1) \cup ftv(\exists\alpha{:}\kappa.\tau') \subseteq dom(\Gamma)$ and $fv(e_2) \cup ftv(\tau) \subseteq dom(\Gamma, \alpha{:}\kappa, x{:}\tau')$
  (c) By (a) and Environment Validity: $\Gamma, \alpha{:}\kappa, x{:}\tau' \vdash \Box$
  (d) By (c) and inversion of E-Term: $x \notin dom(\Gamma, \alpha{:}\kappa)$
  (e) By (b) and (d): $fv(e_2) - \{x\} \subseteq dom(\Gamma)$
  (f) By (b) and (e): $fv(e_1) \cup (fv(e_2) - \{x\}) \subseteq dom(\Gamma)$
  (g) By definition: $fv(\mathsf{let}\ \langle\alpha, x\rangle\ \texttt{=}\ e_1\ \mathsf{in}\ e_2) = fv(e_1) \cup (fv(e_2) - \{x\})$
  (h) By (g) and (f): $fv(\mathsf{let}\ \langle\alpha, x\rangle\ \texttt{=}\ e_1\ \mathsf{in}\ e_2) \subseteq dom(\Gamma)$
  (i) By (b) and (a): $ftv(\tau) \subseteq dom(\Gamma)$
  (j) By (h) and (i): $fv(\mathsf{let}\ \langle\alpha, x\rangle\ \texttt{=}\ e_1\ \mathsf{in}\ e_2) \cup ftv(\tau) \subseteq dom(\Gamma)$

- Case T-Lazy: $e = \mathsf{lazy}\ \langle\zeta, x\rangle\ \texttt{=}\ e_1\ \mathsf{in}\ e_2$
  (a) By inversion: $\Gamma \vdash e_1 : \exists\alpha{:}\kappa.\tau'$ and $\Gamma, \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta] \vdash e_2 : \tau$ where $\zeta \notin ftv(\tau)$
  (b) By (a) and induction: $fv(e_1) \cup ftv(\exists\alpha{:}\kappa.\tau') \subseteq dom(\Gamma)$ and $fv(e_2) \cup ftv(\tau) \subseteq dom(\Gamma, \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta])$
  (c) By (a) and Environment Validity: $\Gamma, \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta] \vdash \Box$
  (d) By (c) and inversion of E-Term: $x \notin dom(\Gamma, \zeta{:}\kappa)$
  (e) By (b) and (d): $fv(e_2) - \{x\} \subseteq dom(\Gamma)$
  (f) By (b) and (e): $fv(e_1) \cup (fv(e_2) - \{x\}) \subseteq dom(\Gamma)$
  (g) By definition: $fv(\mathsf{lazy}\ \langle\zeta, x\rangle\ \texttt{=}\ e_1\ \mathsf{in}\ e_2) = fv(e_1) \cup (fv(e_2) - \{x\})$
  (h) By (g) and (f): $fv(\mathsf{lazy}\ \langle\zeta, x\rangle\ \texttt{=}\ e_1\ \mathsf{in}\ e_2) \subseteq dom(\Gamma)$
  (i) By (b) and (a): $ftv(\tau) \subseteq dom(\Gamma)$
  (j) By (h) and (i): $fv(\mathsf{lazy}\ \langle\zeta, x\rangle\ \texttt{=}\ e_1\ \mathsf{in}\ e_2) \cup ftv(\tau) \subseteq dom(\Gamma)$

- Case T-Case: $e = \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$
  (a) By inversion:
    i. $\Gamma \vdash e_0 : \tau_0$
    ii. $\Gamma, x{:}\tau_0' \vdash e_1 : \tau$
    iii. $\Gamma \vdash e_2 : \tau$
  (b) By (a) and induction:
    i. $fv(e_0) \subseteq dom(\Gamma)$
    ii. $fv(e_1) \subseteq dom(\Gamma, x{:}\tau_0')$
    iii. $fv(e_2) \cup ftv(\tau) \subseteq dom(\Gamma)$
  (c) By (a-ii) and Environment Validity: $\Gamma, x{:}\tau_0' \vdash \Box$
  (d) By (c) and inversion of E-Term: $x \notin dom(\Gamma)$
  (e) By (b-ii) and (d): $fv(e_1) - \{x\} \subseteq dom(\Gamma)$
  (f) By (b-i), (e), and (b-iii): $fv(e_0) \cup (fv(e_1) - \{x\}) \cup fv(e_2) \subseteq dom(\Gamma)$
  (g) By definition: $fv(\mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) = fv(e_0) \cup (fv(e_1) - \{x\}) \cup fv(e_2)$
  (h) By (g) and (f): $fv(\mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) \subset dom(\Gamma)$
  (i) By (h) and (b-iii): $fv(\mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) \cup ftv(\tau) \subset dom(\Gamma)$

$\Box$

Proposition 5 (Strengthening)**.**

1. *If* $\Gamma \vdash \tau : \kappa$ *and* $\Gamma' \subseteq \Gamma$ *with* $\Gamma' \vdash \Box$ *and* $ftv(\tau) \subseteq dom(\Gamma')$*, then* $\Gamma' \vdash \tau : \kappa$*.*
2. *If* $\Gamma_1, x{:}\tau, \Gamma_2 \vdash \Box$*, then* $\Gamma_1, \Gamma_2 \vdash \Box$*.*

Proof. (1) by induction on the derivation of $\Gamma \vdash \tau : \kappa$. (2) by induction on the structure of Γ_2.

1. Let $\Gamma \vdash \tau : \kappa$ and $\Gamma' \subseteq \Gamma$ and $\Gamma' \vdash \Box$ and $ftv(\tau) \subseteq dom(\Gamma')$.

   - Case K-Var: $\tau = \xi$
     - (a) By inversion: $\kappa = \Gamma(\xi)$
     - (b) By (a) and assumption: $\Gamma'(\xi) = \kappa$
     - (c) By (b), assumption, and K-Var: $\Gamma' \vdash \xi : \kappa$
   - Case K-Arrow: $\tau = \tau_1 \to \tau_2$ where $\kappa = \Omega$
     - (a) By inversion: $\Gamma \vdash \tau_1 : \Omega$ and $\Gamma \vdash \tau_2 : \Omega$
     - (b) By assumption: $ftv(\tau_1) \subseteq dom(\Gamma')$ and $ftv(\tau_2) \subseteq dom(\Gamma')$
     - (c) By (a), assumption, (b), and induction: $\Gamma' \vdash \tau_1 : \Omega$ and $\Gamma' \vdash \tau_2 : \Omega$
     - (d) By (c) and K-Arrow: $\Gamma' \vdash \tau_1 \to \tau_2 : \Omega$
   - Case K-Times: $\tau = \tau_1 \times \tau_2$ where $\kappa = \Omega$
     - (a) By inversion: $\Gamma \vdash \tau_1 : \Omega$ and $\Gamma \vdash \tau_2 : \Omega$ and $\kappa = \Omega$
     - (b) By assumption: $ftv(\tau_1) \subseteq dom(\Gamma')$ and $ftv(\tau_2) \subseteq dom(\Gamma')$
     - (c) By (a), assumption, (b), and induction: $\Gamma' \vdash \tau_1 : \Omega$ and $\Gamma' \vdash \tau_2 : \Omega$
     - (d) By (c) and K-Times: $\Gamma' \vdash \tau_1 \times \tau_2 : \Omega$
   - Case K-Univ: $\tau = \forall\alpha{:}\kappa'.\tau'$ where $\kappa = \Omega$
     - (a) By inversion: $\Gamma, \alpha{:}\kappa' \vdash \tau' : \Omega$
     - (b) By definition: $ftv(\tau') \subseteq ftv(\tau) \cup \{\alpha\}$
     - (c) By (b) and assumption: $ftv(\tau') \subseteq dom(\Gamma', \alpha{:}\kappa')$
     - (d) By (a) and Environment Validity: $\Gamma, \alpha{:}\kappa' \vdash \Box$
     - (e) By (d) and inversion of E-Type: $\alpha \notin dom(\Gamma)$
     - (f) By (e) and assumption: $\alpha \notin dom(\Gamma')$
     - (g) By (e), (f), and assumption: $\Gamma', \alpha{:}\kappa' \subseteq \Gamma, \alpha{:}\kappa'$
     - (h) By (f), assumption, and E-Type: $\Gamma', \alpha{:}\kappa' \vdash \Box$
     - (i) By (a), (g), (h), (c), and induction: $\Gamma', \alpha{:}\kappa' \vdash \tau' : \Omega$
     - (j) By (i) and K-Univ: $\Gamma' \vdash \forall\alpha{:}\kappa'.\tau' : \Omega$
   - Case K-Exist: $\tau = \exists\alpha{:}\kappa'.\tau'$ where $\kappa = \Omega$
     - (a) By inversion: $\Gamma, \alpha{:}\kappa' \vdash \tau' : \Omega$
     - (b) By definition: $ftv(\tau') \subseteq ftv(\tau) \cup \{\alpha\}$
     - (c) By (b) and assumption: $ftv(\tau') \subseteq dom(\Gamma', \alpha{:}\kappa')$
     - (d) By (a) and Environment Validity: $\Gamma, \alpha{:}\kappa' \vdash \Box$
     - (e) By (d) and inversion of E-Type: $\alpha \notin dom(\Gamma)$
     - (f) By (e) and assumption: $\alpha \notin dom(\Gamma')$
     - (g) By (e), (f), and assumption: $\Gamma', \alpha{:}\kappa' \subseteq \Gamma, \alpha{:}\kappa'$
     - (h) By (f), assumption, and E-Type: $\Gamma', \alpha{:}\kappa' \vdash \Box$
     - (i) By (a), (g), (h), (c), and induction: $\Gamma', \alpha{:}\kappa' \vdash \tau' : \Omega$
     - (j) By (i) and K-Exist: $\Gamma' \vdash \exists\alpha{:}\kappa'.\tau' : \Omega$

- Case K-Abs: $\tau = \lambda\alpha{:}\kappa'.\tau'$ where $\kappa = \kappa' \to \kappa''$
  (a) By inversion: $\Gamma, \alpha{:}\kappa' \vdash \tau' : \kappa''$
  (b) By definition: $ftv(\tau') \subseteq ftv(\tau) \cup \{\alpha\}$
  (c) By (b) and assumption: $ftv(\tau') \subseteq dom(\Gamma', \alpha{:}\kappa')$
  (d) By (a) and Environment Validity: $\Gamma, \alpha{:}\kappa' \vdash \Box$
  (e) By (d) and inversion of E-Type: $\alpha \notin dom(\Gamma)$
  (f) By (e) and assumption: $\alpha \notin dom(\Gamma')$
  (g) By (e), (f), and assumption: $\Gamma', \alpha{:}\kappa' \subseteq \Gamma, \alpha{:}\kappa'$
  (h) By (f), assumption, and E-Type: $\Gamma', \alpha{:}\kappa' \vdash \Box$
  (i) By (a), (g), (h), (c), and induction: $\Gamma', \alpha{:}\kappa' \vdash \tau' : \kappa''$
  (j) By (i) and K-Abs: $\Gamma' \vdash \lambda\alpha{:}\kappa'.\tau' : \kappa' \to \kappa''$
- Case K-App: $\tau = \tau_1\ \tau_2$
  (a) By inversion: $\Gamma \vdash \tau_1 : \kappa' \to \kappa$ and $\Gamma \vdash \tau_2 : \kappa'$
  (b) By assumption: $ftv(\tau_1) \subseteq dom(\Gamma')$ and $ftv(\tau_2) \subseteq dom(\Gamma')$
  (c) By (a), assumption, (b), and induction: $\Gamma' \vdash \tau_1 : \kappa' \to \kappa$ and $\Gamma' \vdash \tau_2 : \kappa'$
  (d) By (c) and K-App: $\Gamma' \vdash \tau_1\ \tau_2 : \kappa$
- Case K-Pair: $\tau = \langle\tau_1, \tau_2\rangle$ where $\kappa = \kappa_1 \times \kappa_2$
  (a) By inversion: $\Gamma \vdash \tau_1 : \kappa_1$ and $\Gamma \vdash \tau_2 : \kappa_2$
  (b) By assumption: $ftv(\tau_1) \subseteq dom(\Gamma')$ and $ftv(\tau_2) \subseteq dom(\Gamma')$
  (c) By (a), assumption, (b), and induction: $\Gamma' \vdash \tau_1 : \kappa_1$ and $\Gamma' \vdash \tau_2 : \kappa_2$
  (d) By (c) and K-Pair: $\Gamma' \vdash \langle\tau_1, \tau_2\rangle : \kappa_1 \times \kappa_2$
- Case K-Proj1: $\tau = \tau'.1$
  (a) By inversion: $\Gamma \vdash \tau' : \kappa \times \kappa_2$
  (b) By assumption: $ftv(\tau') \subseteq dom(\Gamma')$
  (c) By (a), assumption, (b), and induction: $\Gamma' \vdash \tau' : \kappa \times \kappa_2$
  (d) By (c) and K-Proj1: $\Gamma' \vdash \tau'.1 : \kappa$
- Case K-Proj2: $\tau = \tau'.2$
  (a) By inversion: $\Gamma \vdash \tau' : \kappa_1 \times \kappa$
  (b) By assumption: $ftv(\tau') \subseteq dom(\Gamma')$
  (c) By (a) and induction: $\Gamma' \vdash \tau' : \kappa_1 \times \kappa$
  (d) By (c) and K-Proj2: $\Gamma' \vdash \tau'.2 : \kappa$

2. Let $\Gamma_1, x{:}\tau, \Gamma_2 \vdash \Box$.

- Case $\Gamma_2 = \cdot$:
  (a) By inversion of E-Term: $\Gamma_1 \vdash \tau : \Omega$
  (b) By (a) and Environment Validity: $\Gamma_1 \vdash \Box$
- Case $\Gamma_2 = \Gamma_2', \xi{:}\kappa$:
  (a) By inversion of E-Type: $\Gamma_1, x{:}\tau, \Gamma_2' \vdash \Box$ and $\xi \notin dom(\Gamma_1, x{:}\tau, \Gamma_2')$
  (b) By (a) and induction: $\Gamma_1, \Gamma_2' \vdash \Box$
  (c) By (a): $\xi \notin dom(\Gamma_1, \Gamma_2')$
  (d) By (b), (c), and E-Type: $\Gamma_1, \Gamma_2', \xi{:}\kappa \vdash \Box$
- Case $\Gamma_2 = \Gamma_2', x'{:}\tau'$:
  (a) By inversion of E-Term: $\Gamma_1, x{:}\tau, \Gamma_2' \vdash \tau' : \Omega$ and $x' \notin dom(\Gamma_1, x{:}\tau, \Gamma_2')$

(b) By (a) and Environment Validity: $\Gamma_1, x{:}\tau, \Gamma_2' \vdash \Box$

(c) By (b) and induction: $\Gamma_1, \Gamma_2' \vdash \Box$

(d) By (a) and Variable Containment: $\mathit{ftv}(\tau') \subseteq \mathit{dom}(\Gamma_1, x{:}\tau, \Gamma_2')$

(e) By (d): $\mathit{ftv}(\tau') \subseteq \mathit{dom}(\Gamma_1, \Gamma_2')$

(f) By (a), (c), (e), and (1): $\Gamma_1, \Gamma_2' \vdash \tau' : \Omega$

(g) By (a): $x' \notin \mathit{dom}(\Gamma_1, \Gamma_2')$

(h) By (f), (g), and E-Term: $\Gamma_1, \Gamma_2', x'{:}\tau' \vdash \Box$

$\Box$

Proposition 6 (Weakening)**.**

1. If $\Gamma \vdash \tau : \kappa$ *and* $\Gamma \subseteq \Gamma'$ *where* $\Gamma' \vdash \Box$*, then* $\Gamma' \vdash \tau : \kappa$*.*

2. If $\Gamma \vdash e : \tau$ *and* $\Gamma \subseteq \Gamma'$ *where* $\Gamma' \vdash \Box$*, then* $\Gamma' \vdash e : \tau$*.*

Proof. (1) by induction on the derivation of $\Gamma \vdash \tau : \kappa$. (2) by induction on the derivation of $\Gamma \vdash e : \tau$.

1. Let $\Gamma \vdash \tau : \kappa$ with $\Gamma \subseteq \Gamma'$ and $\Gamma' \vdash \Box$ where w.l.o.g. $\mathit{bv}() \cap \mathit{dom}(') = \emptyset$.

- Case K-Var: $\tau = \xi$

  (a) By inversion: $\kappa = \Gamma(\xi)$

  (b) By assumption: $\Gamma'(\xi) = \Gamma(\xi)$

  (c) By (b), (a), assumption, and K-Var: $\Gamma' \vdash \xi : \kappa$

- Case K-Arrow: $\tau = \tau_1 \to \tau_2$ and $\kappa = \Omega$

  (a) By inversion: $\Gamma \vdash \tau_1 : \Omega$ and $\Gamma \vdash \tau_2 : \Omega$

  (b) By (a), assumption, and induction: $\Gamma' \vdash \tau_1 : \Omega$ and $\Gamma' \vdash \tau_2 : \Omega$

  (c) By (b) and K-Arrow: $\Gamma' \vdash \tau_1 \to \tau_2 : \Omega$

- Case K-Times: $\tau = \tau_1 \times \tau_2$ and $\kappa = \Omega$

  (a) By inversion: $\Gamma \vdash \tau_1 : \Omega$ and $\Gamma \vdash \tau_2 : \Omega$

  (b) By (a), assumption, and induction: $\Gamma' \vdash \tau_1 : \Omega$ and $\Gamma' \vdash \tau_2 : \Omega$

  (c) By (b) and K-Arrow: $\Gamma' \vdash \tau_1 \times \tau_2 : \Omega$

- Case K-Univ: $\tau = \forall\alpha{:}\kappa'.\tau'$ and $\kappa = \Omega$

  (a) By inversion: $\Gamma, \alpha{:}\kappa' \vdash \tau' : \Omega$

  (b) By assumption: $\alpha \notin \mathit{dom}(\Gamma')$ and hence $\Gamma, \alpha{:}\kappa' \subseteq \Gamma', \alpha{:}\kappa'$

  (c) By (b), assumption, and E-Type: $\Gamma', \alpha{:}\kappa' \vdash \Box$

  (d) By (a), (b), (c), and induction: $\Gamma', \alpha{:}\kappa' \vdash \tau' : \Omega$

  (e) By (d) and K-Univ: $\Gamma' \vdash \forall\alpha{:}\kappa'.\tau' : \Omega$

- Case K-Exist: $\tau = \exists\alpha{:}\kappa'.\tau'$ and $\kappa = \Omega$

  (a) By inversion: $\Gamma, \alpha{:}\kappa' \vdash \tau' : \Omega$

  (b) By assumption: $\alpha \notin \mathit{dom}(\Gamma')$ and hence $\Gamma, \alpha{:}\kappa' \subseteq \Gamma', \alpha{:}\kappa'$

  (c) By (b), assumption, and E-Type: $\Gamma', \alpha{:}\kappa' \vdash \Box$

  (d) By (a), (b), (c), and induction: $\Gamma', \alpha{:}\kappa' \vdash \tau' : \Omega$

  (e) By (d) and K-Exist: $\Gamma' \vdash \exists\alpha{:}\kappa'.\tau' : \Omega$

- Case K-Abs: $\tau = \lambda\alpha{:}\kappa'.\tau'$ and $\kappa = \kappa' \to \kappa''$

  (a) By inversion: $\Gamma, \alpha{:}\kappa' \vdash \tau' : \kappa''$

(b) By assumption: $\alpha \notin dom(\Gamma')$ and hence $\Gamma, \alpha{:}\kappa' \subseteq \Gamma', \alpha{:}\kappa'$

(c) By (b), assumption, and E-Type: $\Gamma', \alpha'{:}\kappa' \vdash \Box$

(d) By (a), (b), (c), and induction: $\Gamma', \alpha{:}\kappa' \vdash \tau' : \kappa''$

(e) By (d) and K-Abs: $\Gamma' \vdash \lambda\alpha{:}\kappa'.\tau' : \kappa' \to \kappa''$

- Case K-App: $\tau = \tau_1\ \tau_2$

  (a) By inversion: $\Gamma \vdash \tau_1 : \kappa' \to \kappa$ and $\Gamma \vdash \tau_2 : \kappa'$

  (b) By (a), assumption, and induction: $\Gamma' \vdash \tau_1 : \kappa' \to \kappa$ and $\Gamma' \vdash \tau_2 : \kappa'$

  (c) By (b) and K-App: $\Gamma' \vdash \tau_1\ \tau_2 : \kappa$

- Case K-Pair: $\tau = \langle\tau_1, \tau_2\rangle$ and $\kappa = \kappa_1 \times \kappa_2$

  (a) By inversion: $\Gamma \vdash \tau_1 : \kappa_1$ and $\Gamma \vdash \tau_2 : \kappa_2$

  (b) By (a), assumption, and induction: $\Gamma' \vdash \tau_1 : \kappa_1$ and $\Gamma' \vdash \tau_2 : \kappa_2$

  (c) By (b) and K-Pair: $\Gamma' \vdash \langle\tau_1, \tau_2\rangle : \kappa_1 \times \kappa_2$

- Case K-Proj1: $\tau = \tau'.1$

  (a) By inversion: $\Gamma \vdash \tau' : \kappa \times \kappa_2$

  (b) By (a), assumption, and induction: $\Gamma' \vdash \tau' : \kappa \times \kappa_2$

  (c) By (b) and K-Proj1: $\Gamma' \vdash \tau'.1 : \kappa$

- Case K-Proj2: $\tau = \tau'.2$

  (a) By inversion: $\Gamma \vdash \tau' : \kappa_1 \times \kappa$

  (b) By (a), assumption, and induction: $\Gamma' \vdash \tau' : \kappa_1 \times \kappa$

  (c) By (b) and K-Proj2: $\Gamma' \vdash \tau'.2 : \kappa$

2. Let $\Gamma \vdash e : \tau$ and $\Gamma \subseteq \Gamma'$ and $\Gamma' \vdash \Box$.

- Case T-Equiv:

  (a) By inversion: $\Gamma \vdash e : \tau'$ and $\Gamma \vdash \tau : \Omega$ where $\tau' \equiv \tau$

  (b) By (a), assumption, and induction: $\Gamma' \vdash e : \tau'$

  (c) By (a), assumption and (1): $\Gamma' \vdash \tau : \Omega$

  (d) By (b), (a), (c), and T-Equiv: $\Gamma' \vdash e : \tau$

- Case T-Var: $e = x$

  (a) By inversion: $\Gamma(x) = \tau$

  (b) By assumption: $\Gamma'(x) = \Gamma(x)$

  (c) By (b), (a), assumption, and T-Var: $\Gamma' \vdash x : \tau$

- Case T-Abs: $e = \lambda x{:}\tau_1.e_1$ and $\tau = \tau_1 \to \tau_2$

  (a) By inversion: $\Gamma, x{:}\tau_1 \vdash e_1 : \tau_2$

  (b) By assumption: $x \notin dom(\Gamma')$ and hence $\Gamma, x{:}\tau_1 \subseteq \Gamma', x{:}\tau_1$

  (c) By (a) and Environment Validity: $\Gamma \vdash \tau_1 : \Omega$

  (d) By (c), assumption, and (1): $\Gamma' \vdash \tau_1 : \Omega$

  (e) By (b) and E-Term: $\Gamma', x{:}\tau_1 \vdash \Box$

  (f) By (a), (b), (e), and induction: $\Gamma', x{:}\tau_1 \vdash e_1 : \tau_2$

  (g) By (f) and T-Abs: $\Gamma' \vdash \lambda x{:}\tau_1.e_1 : \tau_1 \to \tau_2$

- Case T-App: $e = e_1\ e_2$

  (a) By inversion: $\Gamma \vdash e_1 : \tau' \to \tau$ and $\Gamma \vdash e_2 : \tau'$

  (b) By (a), assumption, and induction: $\Gamma' \vdash e_1 : \tau' \to \tau$ and $\Gamma' \vdash e_2 : \tau'$

  (c) By (b) and T-App: $\Gamma' \vdash e_1\ e_2 : \tau$

- Case T-Gen: $e = \lambda\alpha{:}\kappa.e_1$ and $\tau = \forall\alpha{:}\kappa.\tau_1$
  (a) By inversion: $\Gamma, \alpha{:}\kappa \vdash e_1 : \tau_1$
  (b) By assumption: $\alpha \notin dom(\Gamma')$ and hence $\Gamma, \alpha{:}\kappa \subseteq \Gamma', \alpha{:}\kappa$
  (c) By (b), assumption, and E-Type: $\Gamma', \alpha{:}\kappa \vdash \Box$
  (d) By (a), (b), (c), and induction: $\Gamma', \alpha{:}\kappa \vdash e_1 : \tau_1$
  (e) By (d) and T-Gen: $\Gamma' \vdash \lambda\alpha{:}\kappa.e_1 : \forall\alpha{:}\kappa.\tau_1$
- Case T-Inst: $e = e_1\ \tau_2$
  (a) By inversion: $\Gamma \vdash e_1 : \forall\alpha{:}\kappa.\tau_1$ and $\Gamma \vdash \tau_2 : \kappa$ where $\tau = \tau_1[\alpha := \tau_2]$
  (b) By (a), assumption, and induction: $\Gamma' \vdash e_1 : \forall\alpha{:}\kappa.\tau_1$
  (c) By (a) and (1): $\Gamma' \vdash \tau_2 : \kappa$
  (d) By (b), (c), and T-Inst: $\Gamma' \vdash e_1\ \tau_2 : \tau_1[\alpha := \tau_2]$
- Case T-Pair: $e = \langle e_1, e_2\rangle$ and $\tau = \tau_1 \times \tau_2$
  (a) By inversion: $\Gamma \vdash e_1 : \tau_1$ and $\Gamma \vdash e_2 : \tau_2$
  (b) By (a), assumption, and induction: $\Gamma' \vdash e_1 : \tau_1$ and $\Gamma' \vdash e_2 : \tau_2$
  (c) By (b) and T-Pair: $\Gamma' \vdash \langle e_1, e_2\rangle : \tau_1 \times \tau_2$
- Case T-Proj: $e = \mathsf{let}\ \langle x_1, x_2\rangle = e_1\ \mathsf{in}\ e_2$
  (a) By inversion: $\Gamma \vdash e_1 : \tau_1 \times \tau_2$ and $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_2 : \tau$
  (b) By assumption: $\{x_1, x_2\} \cap dom(\Gamma') = \emptyset$ and hence $\Gamma, x_1{:}\tau_1 \subseteq \Gamma', x_1{:}\tau_1$ and $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \subseteq \Gamma', x_1{:}\tau_1, x_2{:}\tau_2$
  (c) By (a) and Environment Validity: $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash \Box$
  (d) By (c) and inversion of E-Term: $\Gamma, x_1{:}\tau_1 \vdash \tau_2 : \Omega$
  (e) By (d) and Environment Validity: $\Gamma, x_1{:}\tau_1 \vdash \Box$
  (f) By (e) and inversion of E-Term: $\Gamma \vdash \tau_1 : \Omega$
  (g) By (f), assumption, and (1): $\Gamma' \vdash \tau_1 : \Omega$
  (h) By (g), (b), and E-Term: $\Gamma', x_1{:}\tau_1 \vdash \Box$
  (i) By (d), (h), (b), and induction: $\Gamma', x_1{:}\tau_1 \vdash \tau_2 : \Omega$
  (j) By (i), (b), and E-Term: $\Gamma', x_1{:}\tau_1, x_2{:}\tau_2 \vdash \Box$
  (k) By (j), (a), (b), and induction: $\Gamma', x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_2 : \tau$
  (l) By (a), assumption, and induction: $\Gamma' \vdash e_1 : \tau_1 \times \tau_2$
  (m) By (k), (l), and T-Proj: $\Gamma' \vdash \mathsf{let}\ \langle x_1, x_2\rangle = e_1\ \mathsf{in}\ e_2 : \tau$
- Case T-Close: $e = \langle \tau_1, e'\rangle{:}\tau$ and $\tau = \exists\alpha{:}\kappa.\tau_2$
  (a) By inversion: $\Gamma \vdash \tau_1 : \kappa$ and $\Gamma \vdash e' : \tau_2[\alpha := \tau_1]$ and $\Gamma \vdash \exists\alpha{:}\kappa.\tau_2 : \Omega$
  (b) By (a), assumption, and induction: $\Gamma' \vdash e' : \tau_2[\alpha := \tau_1]$
  (c) By (a), assumption, and (1): $\Gamma' \vdash \tau_1 : \kappa$ and $\Gamma' \vdash \exists\alpha{:}\kappa.\tau_2 : \Omega$
  (d) By (b), (c), and T-Close: $\Gamma' \vdash \langle \tau_1, e'\rangle{:}\exists\alpha{:}\kappa.\tau_2 : \exists\alpha{:}\kappa.\tau_2$
- Case T-Open: $e = \mathsf{let}\ \langle \alpha, x\rangle = e_1\ \mathsf{in}\ e_2$
  (a) By inversion: $\Gamma \vdash e_1 : \exists\alpha{:}\kappa.\tau'$ and $\Gamma, \alpha{:}\kappa, x{:}\tau' \vdash e_2 : \tau$ where $\alpha \notin ftv(\tau)$
  (b) By assumption: $\{\alpha, x\} \cap dom(\Gamma') = \emptyset$ and hence $\Gamma, \alpha{:}\kappa \subseteq \Gamma', \alpha{:}\kappa$ and $\Gamma, \alpha{:}\kappa, x{:}\tau' \subseteq \Gamma', \alpha{:}\kappa, x{:}\tau'$
  (c) By (a) and Environment Validity: $\Gamma, \alpha{:}\kappa, x{:}\tau' \vdash \Box$
  (d) By (c) and inversion of E-Type: $\Gamma, \alpha{:}\kappa \vdash \tau' : \Omega$
  (e) By (b), assumption, and E-Type: $\Gamma', \alpha{:}\kappa \vdash \Box$
  (f) By (d), (e), (b), and induction: $\Gamma', \alpha{:}\kappa \vdash \tau' : \Omega$

(g) By (f), (b), and E-Term: $\Gamma', \alpha{:}\kappa, x{:}\tau' \vdash \Box$
(h) By (g), (a), (b), and induction: $\Gamma', \alpha{:}\kappa, x{:}\tau' \vdash e_2 : \tau$
(i) By (a), assumption, and induction: $\Gamma' \vdash e_1 : \exists\alpha{:}\kappa.\tau'$
(j) By (h), (i), (a), and T-Open: $\Gamma' \vdash \mathsf{let}\ \langle\alpha, x\rangle = e_1\ \mathsf{in}\ e_2 : \tau$

- Case T-Lazy: $e = \mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ e_2$
  (a) By inversion: $\Gamma \vdash e_1 : \exists\alpha{:}\kappa.\tau'[\alpha := \zeta]$ and $\Gamma, \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta][\alpha := \zeta] \vdash e_2 : \tau$ where $\zeta \notin ftv(\tau)$
  (b) By assumption: $\{\zeta, x\} \cap dom(\Gamma') = \emptyset$ and hence $\Gamma, \zeta{:}\kappa \subseteq \Gamma', \zeta{:}\kappa$ and $\Gamma, \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta] \subseteq \Gamma', \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta]$
  (c) By (a) and Environment Validity: $\Gamma, \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta] \vdash \Box$
  (d) By (c) and inversion of E-Term: $\Gamma, \zeta{:}\kappa \vdash \tau'[\alpha := \zeta] : \Omega$
  (e) By (b), assumption, and E-Type: $\Gamma', \zeta{:}\kappa \vdash \Box$
  (f) By (d), (e), (b), and induction: $\Gamma', \zeta{:}\kappa \vdash \tau'[\alpha := \zeta] : \Omega$
  (g) By (f), (b), and E-Term: $\Gamma', \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta] \vdash \Box$
  (h) By (g), (a), (b), and induction: $\Gamma', \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta] \vdash e_2 : \tau$
  (i) By (a), assumption, and induction: $\Gamma' \vdash e_1 : \exists\alpha{:}\kappa.\tau'$
  (j) By (h), (i), (a), and T-Lazy: $\Gamma' \vdash \mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ e_2 : \tau$
- Case T-Case: $e = \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$
  (a) By inversion:
    i. $\Gamma \vdash e_0 : \tau_0$
    ii. $\Gamma, x{:}\tau_0' \vdash e_1 : \tau$
    iii. $\Gamma \vdash e_2 : \tau$
  (b) By assumption: $x \notin dom(\Gamma')$ and hence $\Gamma, x{:}\tau_0' \subseteq \Gamma', x{:}\tau_0'$
  (c) By (a-ii) and Environment Validity: $\Gamma, x{:}\tau_0' \vdash \Box$
  (d) By (c) and inversion of E-Term: $\Gamma \vdash \tau_0' : \Omega$
  (e) By (d), assumption, and (1): $\Gamma' \vdash \tau_0' : \Omega$
  (f) By (e), (b), and E-Term: $\Gamma', x{:}\tau_0' \vdash \Box$
  (g) By (f), (a-ii), (b), and induction: $\Gamma', x{:}\tau_0' \vdash e_1 : \tau$
  (h) By (a-i), assumption, and induction: $\Gamma' \vdash e_0 : \tau_0$
  (i) By (a-iii), assumption, and induction: $\Gamma' \vdash e_2 : \tau$
  (j) By (h), (g), (i), and T-Case: $\Gamma' \vdash \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau$

□

Proposition 7 (Type Substitution)**.**

1. *If* $\Gamma_1 \vdash \tau' : \kappa'$ *and* $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \Box$*, then* $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \Box$.
2. *If* $\Gamma_1 \vdash \tau' : \kappa'$ *and* $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \tau : \kappa$*, then* $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau[\xi := \tau'] : \kappa$.
3. *If* $\Gamma_1 \vdash \tau' : \kappa'$ *and* $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash e : \tau$*, then* $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e[\xi := \tau'] : \tau[\xi := \tau']$.
4. *If* $\tau \equiv \tau''$*, then* $\tau[\xi := \tau'] \equiv \tau''[\xi := \tau']$.

Proof. (1), (2), and (3) by simultaneous induction on the kinding derivations. (4) by induction on the derivation.

1. Let $\Gamma_1 \vdash \tau' : \kappa'$ and $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \Box$.
   - Case E-Empty: not possible

- Case E-Type:
  - Subcase $\Gamma_2 = \cdot$:
    (a) By inversion: $\Gamma_1 \vdash \Box$
  - Subcase $\Gamma_2 = \Gamma_2', \xi'{:}\kappa''$:
    (a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2' \vdash \Box$ and $\xi' \notin \mathit{dom}(\Gamma_1, \xi{:}\kappa', \Gamma_2')$
    (b) By (a): $\xi' \notin \mathit{dom}(\Gamma_1, \Gamma_2'[\xi := \tau'])$
    (c) By (a) and induction: $\Gamma_1, \Gamma_2'[\xi := \tau'] \vdash \Box$
    (d) By (c), (b), and E-Type: $\Gamma_1, \Gamma_2'[\xi := \tau'], \xi'{:}\kappa'' \vdash \Box$
    (e) By (d): $\Gamma_1, (\Gamma_2', \xi'{:}\kappa'')[\xi := \tau'] \vdash \Box$
- Case E-Term: $\Gamma_2 = \Gamma_2', x{:}\tau''$
  (a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2' \vdash \tau'' : \Omega$ and $x \notin \mathit{dom}(\Gamma_1, \xi{:}\kappa', \Gamma_2')$
  (b) By (a): $x \notin \mathit{dom}(\Gamma_1, \Gamma_2'[\xi := \tau'])$
  (c) By (a) and induction: $\Gamma_1, \Gamma_2'[\xi := \tau'] \vdash \tau''[\xi := \tau'] : \Omega$
  (d) By (c), (b), and E-Type: $\Gamma_1, \Gamma_2'[\xi := \tau'], x{:}\tau''[\xi := \tau'] \vdash \Box$
  (e) By (d): $\Gamma_1, (\Gamma_2', x{:}\tau'')[\xi := \tau'] \vdash \Box$

2. Let $\Gamma_1 \vdash \tau' : \kappa'$ and $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \tau : \kappa$.

- Case K-Var:
  - Subcase $\tau = \xi$
    (a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \Box$ and $\kappa = \kappa'$
    (b) By (a) and induction: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \Box$
    (c) By (b), assumption, and Weakening: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau' : \kappa$
    (d) $\tau' = \xi[\xi := \tau']$
  - Subcase $\tau = \xi' \neq \xi$
    (a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \Box$
    (b) By (a) and induction: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \Box$
    (c) $\kappa = (\Gamma_1, \xi{:}\kappa', \Gamma_2)(\xi') = (\Gamma_1, \Gamma_2[\xi := \tau'])(\xi')$
    (d) By (b), (c), and K-Var: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \xi' : \kappa$
    (e) $\xi' = \xi'[\xi := \tau']$
- Case K-Arrow: $\tau = \tau_1 \to \tau_2$ and $\kappa = \Omega$
  (a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \tau_1 : \Omega$ and $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \tau_2 : \Omega$.
  (b) By (a) and induction: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau_1[\xi := \tau'] : \Omega$ and $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau_2[\xi := \tau'] : \Omega$
  (c) By (b) and K-Arrow: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau_1[\xi := \tau'] \to \tau_2[\xi := \tau'] : \Omega$
  (d) $\tau_1[\xi := \tau'] \to \tau_2[\xi := \tau'] = (\tau_1 \to \tau_2)[\xi := \tau']$
- Case K-Times: $\tau = \tau_1 \times \tau_2$ and $\kappa = \Omega$
  (a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \tau_1 : \Omega$ and $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \tau_2 : \Omega$.
  (b) By (a) and induction: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau_1[\xi := \tau'] : \Omega$ and $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau_2[\xi := \tau'] : \Omega$
  (c) By (b) and K-Times: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau_1[\xi := \tau'] \times \tau_2[\xi := \tau'] : \Omega$
  (d) $\tau_1[\xi := \tau'] \times \tau_2[\xi := \tau'] = (\tau_1 \times \tau_2)[\xi := \tau']$
- Case K-Univ: $\tau = \forall\alpha{:}\kappa''.\tau''$ and $\kappa = \Omega$
  (a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2, \alpha{:}\kappa'' \vdash \tau'' : \Omega$
  (b) By (a) and induction: $\Gamma_1, (\Gamma_2, \alpha{:}\kappa'')[\xi := \tau'] \vdash \tau''[\xi := \tau'] : \Omega$

(c) By (b): $\Gamma_1, \Gamma_2[\xi := \tau'], \alpha{:}\kappa'' \vdash \tau''[\xi := \tau'] : \Omega$
(d) By (c) and K-Univ: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \forall\alpha{:}\kappa''.\tau''[\xi := \tau'] : \Omega$
(e) By (a) and Environment Validity: $\Gamma_1, \xi{:}\kappa', \Gamma_2, \alpha{:}\kappa'' \vdash \Box$
(f) By (e) and inversion of E-Type: $\alpha \notin \mathit{dom}(\Gamma_1, \xi{:}\kappa', \Gamma_2)$ and hence $\alpha \neq \xi$
(g) By assumption and Variable Containment: $\mathit{ftv}(\tau') \subseteq \mathit{dom}(\Gamma_1)$
(h) By (f) and (g): $\alpha \notin \mathit{ftv}(\tau')$
(i) By (d), (f), and (h): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash (\forall\alpha{:}\kappa''.\tau'')[\xi := \tau'] : \Omega$

- Case K-Exist: $\tau = \exists\alpha{:}\kappa''.\tau''$ and $\kappa = \Omega$

(a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2, \alpha{:}\kappa'' \vdash \tau'' : \Omega$
(b) By (a) and induction: $\Gamma_1, (\Gamma_2, \alpha{:}\kappa'')[\xi := \tau'] \vdash \tau''[\xi := \tau'] : \Omega$
(c) By (b): $\Gamma_1, \Gamma_2[\xi := \tau'], \alpha{:}\kappa'' \vdash \tau''[\xi := \tau'] : \Omega$
(d) By (c) and K-Exist: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \exists\alpha{:}\kappa''.\tau''[\xi := \tau'] : \Omega$
(e) By (a) and Environment Validity: $\Gamma_1, \xi{:}\kappa', \Gamma_2, \alpha{:}\kappa'' \vdash \Box$
(f) By (e) and inversion of E-Type: $\alpha \notin \mathit{dom}(\Gamma_1, \xi{:}\kappa', \Gamma_2)$ and hence $\alpha \neq \xi$
(g) By assumption and Variable Containment: $\mathit{ftv}(\tau') \subseteq \mathit{dom}(\Gamma_1)$
(h) By (f) and (g): $\alpha \notin \mathit{ftv}(\tau')$
(i) By (d), (f), and (h): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash (\exists\alpha{:}\kappa''.\tau'')[\xi := \tau'] : \Omega$

- Case K-Abs: $\tau = \lambda\alpha{:}\kappa_1.\tau''$ and $\kappa = \kappa_1 \to \kappa_2$

(a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2, \alpha{:}\kappa_1 \vdash \tau'' : \kappa_2$
(b) By (a) and induction: $\Gamma_1, (\Gamma_2, \alpha{:}\kappa_1)[\xi := \tau'] \vdash \tau''[\xi := \tau'] : \kappa_2$
(c) By (b): $\Gamma_1, \Gamma_2[\xi := \tau'], \alpha{:}\kappa_1 \vdash \tau''[\xi := \tau'] : \kappa_2$
(d) By (a) and Environment Validity: $\Gamma_1, \xi{:}\kappa', \Gamma_2, \alpha{:}\kappa_1 \vdash \Box$
(e) By (e) and inversion of E-Type: $\alpha \notin \mathit{dom}(\Gamma_1, \xi{:}\kappa', \Gamma_2)$ and hence $\alpha \neq \xi$
(f) By assumption and Variable Containment: $\mathit{ftv}(\tau') \subseteq \mathit{dom}(\Gamma_1)$
(g) By (f) and (g): $\alpha \notin \mathit{ftv}(\tau')$
(h) By (d), (f), and (h): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash (\lambda\alpha{:}\kappa_1.\tau'')[\xi := \tau'] : \kappa_1 \to \kappa_2$

- Case K-App: $\tau = \tau_1\ \tau_2$

(a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \tau_1 : \kappa'' \to \kappa$ and $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \tau_2 : \kappa''$
(b) By (a) and induction: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau_1[\xi := \tau'] : \kappa'' \to \kappa$ and $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau_2[\xi := \tau'] : \kappa''$
(c) By (b) and K-App: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau_1[\xi := \tau']\ \tau_2[\xi := \tau'] : \kappa$
(d) $\tau_1[\xi := \tau']\ \tau_2[\xi := \tau'] = (\tau_1\ \tau_2)[\xi := \tau']$

- Case K-Pair: $\tau = \langle\tau_1, \tau_2\rangle$

(a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \tau_1 : \kappa_1$ and $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \tau_2 : \kappa_2$ where $\kappa = \kappa_1 \times \kappa_2$.
(b) By (a) and induction: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau_1[\xi := \tau'] : \kappa_1$ and $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau_2[\xi := \tau'] : \kappa_2$
(c) By (b) and K-Pair: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \langle\tau_1[\xi := \tau'], \tau_2[\xi := \tau']\rangle : \kappa_1 \times \kappa_2$
(d) $\langle\tau_1[\xi := \tau'], \tau_2[\xi := \tau']\rangle = \langle\tau_1, \tau_2\rangle[\xi := \tau']$

- Case K-Proj1: $\tau = \tau''.1$

(a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \tau'' : \kappa \times \kappa_2$
(b) By (a) and induction: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau''[\xi := \tau'] : \kappa \times \kappa_2$
(c) By (b) and K-Proj1: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau''[\xi := \tau'].1 : \kappa$
(d) $\tau''[\xi := \tau'].1 = \tau''.1[\xi := \tau']$

- Case K-Proj2: $\tau = \tau''.2$
  (a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \tau'' : \kappa_1 \times \kappa$
  (b) By (a) and induction: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau''[\xi := \tau'] : \kappa_1 \times \kappa$
  (c) By (b) and K-Proj2: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau''[\xi := \tau'].2 : \kappa$
  (d) $\tau''[\xi := \tau'].2 = \tau''.2[\xi := \tau']$

3. Let $\Gamma_1 \vdash \tau' : \kappa'$ and $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash e : \tau$.

- Case T-Equiv:
  (a) By inversion:
    i. $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash e : \tau_1$
    ii. $\tau_1 \equiv \tau$
    iii. $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \tau : \Omega$
  (b) By (a) and induction:
    i. $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e[\xi := \tau'] : \tau_1[\xi := \tau']$
    ii. $\tau_1[\xi := \tau'] \equiv \tau[\xi := \tau']$
    iii. $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau[\xi := \tau'] : \Omega$
  (c) By (b) and T-Equiv: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e[\xi := \tau'] : \tau[\xi := \tau']$
- Case T-Var: $e = x$
  (a) By inversion: $\tau = (\Gamma_1, \xi{:}\kappa', \Gamma_2)(x)$ and $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \square$
  (b) By (a) and induction: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \square$
  (c) We show: $(\Gamma_1, \Gamma_2[\xi := \tau'])(x) = \tau[\xi := \tau']$
    – Subcase $\Gamma_1(x) = \tau$:
      i. By (a) and Subenvironment Validity: $\Gamma_1, \xi{:}\kappa' \vdash \square$
      ii. By (b) and inversion of E-Type: $\Gamma_1 \vdash \square$ and $\xi \notin dom(\Gamma_1)$
      iii. By (ii) and T-Var: $\Gamma_1 \vdash x : \tau$
      iv. By (iii) and Variable Containment: $ftv(\tau) \subseteq dom(\Gamma_1)$
      v. By (ii) and (iv): $\xi \notin ftv(\tau)$ and hence $\tau[\xi := \tau'] = \tau$
      vi. By (v): $(\Gamma_1, \Gamma_2[\xi := \tau'])(x) = \tau[\xi := \tau']$
    – Subcase $\Gamma_2(x) = \tau$:
      i. Then: $\Gamma_2[\xi := \tau'](x) = \tau[\xi := \tau']$
      ii. By (i): $(\Gamma_1, \Gamma_2[\xi := \tau'])(x) = \tau[\xi := \tau']$
  (d) By (b), (c), and T-Var: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash x : \tau[\xi := \tau']$
- Case T-Abs: $e = \lambda x{:}\tau_1.e'$ and $\tau = \tau_1 \to \tau_2$
  (a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2, x{:}\tau_1 \vdash e' : \tau_2$
  (b) By (a) and induction: $\Gamma_1, (\Gamma_2, x{:}\tau_1)[\xi := \tau'] \vdash e'[\xi := \tau'] : \tau_2[\xi := \tau']$
  (c) By (b): $\Gamma_1, \Gamma_2[\xi := \tau'], x{:}\tau_1[\xi := \tau'] \vdash e'[\xi := \tau'] : \tau_2[\xi := \tau']$
  (d) By (c) and T-Abs: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \lambda x{:}\tau_1[\xi := \tau'].e'[\xi := \tau'] : \tau_1[\xi := \tau'] \to \tau_2[\xi := \tau']$
  (e) By (d): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash (\lambda x{:}\tau_1.e')[\xi := \tau'] : (\tau_1 \to \tau_2)[\xi := \tau']$
- Case T-App: $e = e_1\ e_2$
  (a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash e_1 : \tau_2 \to \tau$ and $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash e_2 : \tau_2$
  (b) By (a) and induction: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e_1[\xi := \tau'] : (\tau_2 \to \tau)[\xi := \tau']$ and $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e_2[\xi := \tau'] : \tau_2[\xi := \tau']$
  (c) By (b): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e_1[\xi := \tau'] : \tau_2[\xi := \tau'] \to \tau[\xi := \tau']$

(d) By (c), (b), and T-App: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e_1[\xi := \tau']\ e_2[\xi := \tau'] : \tau[\xi := \tau']$

(e) By (d): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash (e_1\ e_2)[\xi := \tau'] : \tau[\xi := \tau']$

- Case T-Gen: $e = \lambda\alpha{:}\kappa.e_1$ and $\tau = \forall\alpha{:}\kappa.\tau_1$

  (a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2, \alpha{:}\kappa \vdash e_1 : \tau_1$

  (b) By (a) and induction: $\Gamma_1, (\Gamma_2, \alpha{:}\kappa)[\xi := \tau'] \vdash e_1[\xi := \tau'] : \tau_1[\xi := \tau']$

  (c) By (b): $\Gamma_1, \Gamma_2[\xi := \tau'], \alpha{:}\kappa \vdash e_1[\xi := \tau'] : \tau_1[\xi := \tau']$

  (d) By (d) and T-Gen: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \lambda\alpha{:}\kappa.e_1[\xi := \tau'] : \forall\alpha{:}\kappa.\tau_1[\xi := \tau']$

  (e) By (a) and Environment Validity: $\Gamma_1, \xi{:}\kappa', \Gamma_2, \alpha{:}\kappa \vdash \Box$

  (f) By (f) and inversion of E-Type: $\alpha \notin \mathit{dom}(\Gamma_1, \xi{:}\kappa', \Gamma_2)$ and hence $\alpha \neq \xi$

  (g) By (e) and (g): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash (\lambda\alpha{:}\kappa.e_1)[\xi := \tau'] : (\forall\alpha{:}\kappa.\tau_1)[\xi := \tau']$

- Case T-Inst: $e = e'\ \tau_2$

  (a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash e' : \forall\alpha{:}\kappa.\tau_1$ (w.l.o.g. $\alpha \neq \xi$ and $\alpha \notin \mathit{ftv}(\tau')$) and $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \tau_2 : \kappa$ where $\tau = \tau_1[\alpha := \tau_2]$

  (b) By (a) and induction: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e'[\xi := \tau'] : (\forall\alpha{:}\kappa.\tau_1)[\xi := \tau']$ and $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau_2[\xi := \tau'] : \kappa$

  (c) By (b) and (a): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e'[\xi := \tau'] : \forall\alpha{:}\kappa.\tau_1[\xi := \tau']$

  (d) By (c), (b), and T-Inst: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e'[\xi := \tau']\ \tau_2[\xi := \tau'] : \tau_1[\xi := \tau'][\alpha := \tau_2[\xi := \tau']]$

  (e) By (d): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash (e'\ \tau_2)[\xi := \tau'] : \tau_1[\xi := \tau'][\alpha := \tau_2[\xi := \tau']]$

  (f) By (e), (a), and Substitution: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash (e'\ \tau_2)[\xi := \tau'] : \tau_1[\alpha := \tau_2][\xi := \tau']$

- Case T-Pair: $e = \langle e_1, e_2 \rangle$ and $\tau = \tau_1 \times \tau_2$

  (a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash e_1 : \tau_1$ and $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash e_2 : \tau_2$

  (b) By (a) and induction: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e_1[\xi := \tau'] : \tau_1[\xi := \tau']$ and $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e_2[\xi := \tau'] : \tau_2[\xi := \tau']$

  (c) By (b) and T-Pair: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \langle e_1[\xi := \tau'], e_2[\xi := \tau'] \rangle : \tau_1[\xi := \tau'] \times \tau_2[\xi := \tau']$

  (d) By (c): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \langle e_1, e_2 \rangle[\xi := \tau'] : (\tau_1 \times \tau_2)[\xi := \tau']$

- Case T-Proj: $e = \mathsf{let}\ \langle x_1, x_2 \rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ e_2$

  (a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash e_1 : \tau_1 \times \tau_2$ and $\Gamma_1, \xi{:}\kappa', \Gamma_2, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_2 : \tau$

  (b) By (a) and induction: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e_1[\xi := \tau'] : (\tau_1 \times \tau_2)[\xi := \tau']$ and $\Gamma_1, (\Gamma_2, x_1{:}\tau_1, x_2{:}\tau_2)[\xi := \tau'] \vdash e_2[\xi := \tau'] : \tau[\xi := \tau']$

  (c) By (b): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e_1[\xi := \tau'] : \tau_1[\xi := \tau'] \times \tau_2[\xi := \tau']$ and $\Gamma_1, \Gamma_2[\xi := \tau'], x_1{:}\tau_1[\xi := \tau'], x_2{:}\tau_2[\xi := \tau'] \vdash e_2[\xi := \tau'] : \tau[\xi := \tau']$

  (d) By (c) and T-Proj: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \mathsf{let}\ \langle x_1, x_2 \rangle\ \mathsf{=}\ e_1[\xi := \tau']\ \mathsf{in}\ e_2[\xi := \tau'] : \tau[\xi := \tau']$

  (e) By (d): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \mathsf{let}\ \langle x_1, x_2 \rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ e_2[\xi := \tau'] : \tau[\xi := \tau']$

- Case T-Close: $e = \langle \tau_1, e' \rangle{:}\tau$ and $\tau = \exists\alpha{:}\kappa.\tau_2$ (w.l.o.g. $\alpha \neq \xi$)

  (a) By inversion:

    i. $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \tau_1 : \kappa$

    ii. $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash e' : \tau_2[\alpha := \tau_1]$

    iii. $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash \exists\alpha{:}\kappa.\tau_2 : \Omega$

  (b) By (a) and induction:

    i. $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \tau_1[\xi := \tau'] : \kappa$

    ii. $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e'[\xi := \tau'] : (\tau_2[\alpha := \tau_1])[\xi := \tau']$

iii. $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash (\exists\alpha{:}\kappa.\tau_2)[\xi := \tau'] : \Omega$

(c) By (b-iii): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \exists\alpha{:}\kappa.\tau_2[\xi := \tau'] : \Omega$

(d) By (b-i), (b-ii), (c), and T-Close:
$\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \langle\tau_1[\xi := \tau'], e'[\xi := \tau']\rangle{:}\exists\alpha{:}\kappa.\tau_2[\xi := \tau'] : \exists\alpha{:}\kappa.\tau_2[\xi := \tau']$

(e) By (d): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \langle\tau_1[\xi := \tau'], e'[\xi := \tau']\rangle{:}(\exists\alpha{:}\kappa.\tau_2)[\xi := \tau'] : (\exists\alpha{:}\kappa.\tau_2)[\xi := \tau']$

(f) By (e): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash (\langle\tau_1, e'\rangle{:}\exists\alpha{:}\kappa.\tau_2)[\xi := \tau'] : (\exists\alpha{:}\kappa.\tau_2)[\xi := \tau']$

- Case T-Open: $e = \mathsf{let}\ \langle\alpha, x\rangle = e_1\ \mathsf{in}\ e_2$

(a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash e_1 : \exists\alpha{:}\kappa.\tau_1$ and $\Gamma_1, \xi{:}\kappa', \Gamma_2, \alpha{:}\kappa, x{:}\tau_1 \vdash e_2 : \tau$ where $\alpha \notin \mathit{ftv}(\tau)$

(b) By (a) and induction: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e_1[\xi := \tau'] : (\exists\alpha{:}\kappa.\tau_1)[\xi := \tau']$ and $\Gamma_1, (\Gamma_2, \alpha{:}\kappa, x{:}\tau_1)[\xi := \tau'] \vdash e_2[\xi := \tau'] : \tau[\xi := \tau']$

(c) By (a) and Environment Validity: $\Gamma_1, \xi{:}\kappa', \Gamma_2, \alpha{:}\kappa, x{:}\tau_1 \vdash \Box$

(d) By (c) and Subenvironment Validity: $\Gamma_1, \xi{:}\kappa', \Gamma_2, \alpha{:}\kappa \vdash \Box$

(e) By (d) and inversion of E-Type: $\alpha \notin \mathit{dom}(\Gamma_1, \xi{:}\kappa', \Gamma_2)$ and hence $\alpha \neq \xi$

(f) By assumption and Variable Containment: $\mathit{ftv}(\tau') \subseteq \mathit{dom}(\Gamma_1)$

(g) By (f) and (e): $\alpha \notin \mathit{ftv}(\tau')$

(h) By (b), (e), and (g): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e_1[\xi := \tau'] : \exists\alpha{:}\kappa.\tau_1[\xi := \tau']$

(i) By (b): $\Gamma_1, \Gamma_2[\xi := \tau'], \alpha{:}\kappa, x{:}\tau_1[\xi := \tau'] \vdash e_2[\xi := \tau'] : \tau[\xi := \tau']$

(j) By (a) and (g): $\alpha \notin \mathit{ftv}(\tau[\xi := \tau'])$

(k) By (h), (i), (j), and T-Open: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \mathsf{let}\ \langle\alpha, x\rangle = e_1[\xi := \tau']\ \mathsf{in}\ e_2[\xi := \tau'] : \tau[\xi := \tau']$

(l) By (k) and (e): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash (\mathsf{let}\ \langle\alpha, x\rangle = e_1\ \mathsf{in}\ e_2)[\xi := \tau'] : \tau[\xi := \tau']$

- Case T-Lazy: $e = \mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ e_2$

(a) By inversion: $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash e_1 : \exists\alpha{:}\kappa.\tau_1$ (w.l.o.g. $\alpha \neq \xi$ and $\alpha \notin \mathit{ftv}(\tau')$) and $\Gamma_1, \xi{:}\kappa', \Gamma_2, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash e_2 : \tau$ where $\zeta \notin \mathit{ftv}(\tau)$

(b) By (a) and induction: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e_1[\xi := \tau'] : (\exists\alpha{:}\kappa.\tau_1)[\xi := \tau']$ and $\Gamma_1, (\Gamma_2, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta])[\xi := \tau'] \vdash e_2[\xi := \tau'] : \tau[\xi := \tau']$

(c) By (b) and (a): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e_1[\xi := \tau'] : \exists\alpha{:}\kappa.\tau_1[\xi := \tau']$

(d) By (b): $\Gamma_1, \Gamma_2[\xi := \tau'], \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta][\xi := \tau'] \vdash e_2[\xi := \tau'] : \tau[\xi := \tau']$

(e) By (a) and Environment Validity: $\Gamma_1, \xi{:}\kappa', \Gamma_2, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash \Box$

(f) By (e) and Subenvironment Validity: $\Gamma_1, \xi{:}\kappa', \Gamma_2, \zeta{:}\kappa \vdash \Box$

(g) By (f) and inversion of E-Type: $\zeta \notin \mathit{dom}(\Gamma_1, \xi{:}\kappa', \Gamma_2)$ and hence $\zeta \neq \xi$

(h) By assumption and Variable Containment: $\mathit{ftv}(\tau') \subseteq \mathit{dom}(\Gamma_1)$

(i) By (h) and (g): $\zeta \notin \mathit{ftv}(\tau')$

(j) By (i) and (a): $\zeta \notin \mathit{ftv}(\tau[\xi := \tau'])$

(k) By (c), (d), (j), and T-Open: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \mathsf{lazy}\ \langle\zeta, x\rangle = e_1[\xi := \tau']\ \mathsf{in}\ e_2[\xi := \tau'] : \tau[\xi := \tau']$

(l) By (k) and (g): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash (\mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ e_2)[\xi := \tau'] : \tau[\xi := \tau']$

- Case T-Case: $e = \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$

(a) By inversion:

i. $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash e_0 : \tau_0$

ii. $\Gamma_1, \xi{:}\kappa', \Gamma_2, x{:}\tau_0' \vdash e_1 : \tau$

iii. $\Gamma_1, \xi{:}\kappa', \Gamma_2 \vdash e_2 : \tau$

(b) By (a) and induction:

    i. $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e_0[\xi := \tau'] : \tau_0[\xi := \tau']$
    ii. $\Gamma_1, (\Gamma_2, x{:}\tau_0')[\xi := \tau'] \vdash e_1[\xi := \tau'] : \tau[\xi := \tau']$
    iii. $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash e_2[\xi := \tau'] : \tau[\xi := \tau']$

(c) By (b-ii): $\Gamma_1, \Gamma_2[\xi := \tau'], x{:}\tau_0'[\xi := \tau'] \vdash e_1[\xi := \tau'] : \tau[\xi := \tau']$

(d) By (b-i), (c), (b-iii), and T-Case: $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash \mathsf{tcase}\ e_0[\xi := \tau']{:}\tau_0[\xi := \tau']\ \mathsf{of}\ x{:}\tau_0'[\xi := \tau']\ \mathsf{then}\ e_1[\xi := \tau']\ \mathsf{else}\ e_2[\xi := \tau'] : \tau[\xi := \tau']$

(e) By (d): $\Gamma_1, \Gamma_2[\xi := \tau'] \vdash (\mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2)[\xi := \tau'] : \tau[\xi := \tau']$

4. Let $\tau \equiv \tau''$ and w.l.o.g. $(btv(\tau) \cup btv(\tau')) \cap \{\xi\} \cup ftv(\tau') = \emptyset$.

- Case Q-Refl:
  (a) By inversion: $\tau = \tau''$
  (b) By (a): $\tau[\xi := \tau'] = \tau''[\xi := \tau']$
  (c) By (b) and Q-Refl: $\tau[\xi := \tau'] \equiv \tau''[\xi := \tau']$
- Case Q-Symm:
  (a) By inversion: $\tau'' \equiv \tau$
  (b) By (a) and induction: $\tau''[\xi := \tau'] \equiv \tau[\xi := \tau']$
  (c) By (b) and Q-Symm: $\tau[\xi := \tau'] \equiv \tau''[\xi := \tau']$
- Case Q-Trans:
  (a) By inversion: $\tau \equiv \tau_1$ and $\tau_1 \equiv \tau''$
  (b) By (a) and induction: $\tau[\xi := \tau'] \equiv \tau_1[\xi := \tau']$ and $\tau_1[\xi := \tau'] \equiv \tau''[\xi := \tau']$
  (c) By (b) and Q-Trans: $\tau[\xi := \tau'] \equiv \tau''[\xi := \tau']$
- Case Q-Arrow: $\tau = \tau_1 \to \tau_2$ and $\tau'' = \tau_1' \to \tau_2'$
  (a) By inversion: $\tau_1 \equiv \tau_1'$ and $\tau_2 \equiv \tau_2'$
  (b) By (a) and induction: $\tau_1[\xi := \tau'] \equiv \tau_1'[\xi := \tau']$ and $\tau_2[\xi := \tau'] \equiv \tau_2'[\xi := \tau']$
  (c) By (b) and Q-Arrow: $\tau_1[\xi := \tau'] \to \tau_2[\xi := \tau'] \equiv \tau_1'[\xi := \tau'] \to \tau_2'[\xi := \tau']$
  (d) By (c): $(\tau_1 \to \tau_2)[\xi := \tau'] \equiv (\tau_1' \to \tau_2')[\xi := \tau']$
- Case Q-Times: $\tau = \tau_1 \times \tau_2$ and $\tau'' = \tau_1' \times \tau_2'$
  (a) By inversion: $\tau_1 \equiv \tau_1'$ and $\tau_2 \equiv \tau_2'$
  (b) By (a) and induction: $\tau_1[\xi := \tau'] \equiv \tau_1'[\xi := \tau']$ and $\tau_2[\xi := \tau'] \equiv \tau_2'[\xi := \tau']$
  (c) By (b) and Q-Times: $\tau_1[\xi := \tau'] \times \tau_2[\xi := \tau'] \equiv \tau_1'[\xi := \tau'] \times \tau_2'[\xi := \tau']$
  (d) By (c): $(\tau_1 \times \tau_2)[\xi := \tau'] \equiv (\tau_1' \times \tau_2')[\xi := \tau']$
- Case Q-Univ: $\tau = \forall\alpha{:}\kappa.\tau_1$ and $\tau'' = \forall\alpha{:}\kappa.\tau_2$
  (a) By inversion: $\tau_1 \equiv \tau_2$
  (b) By (a) and induction: $\tau_1[\xi := \tau'] \equiv \tau_2[\xi := \tau']$
  (c) By (b) and Q-Univ: $\forall\alpha{:}\kappa.\tau_1[\xi := \tau'] \equiv \forall\alpha{:}\kappa.\tau_2[\xi := \tau']$
  (d) By (c) and assumption: $(\forall\alpha{:}\kappa.\tau_1)[\xi := \tau'] \equiv (\forall\alpha{:}\kappa.\tau_2[\alpha := \alpha'])[\xi := \tau']$
- Case Q-Exist: $\tau = \exists\alpha{:}\kappa.\tau_1$ and $\tau'' = \exists\alpha{:}\kappa.\tau_2$
  (a) By inversion: $\tau_1 \equiv \tau_2$
  (b) By (a) and induction: $\tau_1[\xi := \tau'] \equiv \tau_2[\xi := \tau']$
  (c) By (b) and Q-Exist: $\exists\alpha{:}\kappa.\tau_1[\xi := \tau'] \equiv \exists\alpha{:}\kappa.\tau_2[\xi := \tau']$
  (d) By (c) and assumption: $(\exists\alpha{:}\kappa.\tau_1)[\xi := \tau'] \equiv (\exists\alpha{:}\kappa.\tau_2)[\xi := \tau']$
- Case Q-Abs: $\tau = \lambda\alpha{:}\kappa.\tau_1$ and $\tau'' = \lambda\alpha{:}\kappa.\tau_2$
  (a) By inversion: $\tau_1 \equiv \tau_2$

(b) By (a) and induction: $\tau_1[\alpha := \alpha'][\xi := \tau'] \equiv \tau_2[\alpha := \alpha'][\xi := \tau']$

(c) By (b) and Q-Abs: $\lambda\alpha{:}\kappa.\tau_1[\xi := \tau'] \equiv \lambda\alpha{:}\kappa.\tau_2[\xi := \tau']$

(d) By (c) and assumption: $(\lambda\alpha{:}\kappa.\tau_1)[\xi := \tau'] \equiv (\lambda\alpha{:}\kappa.\tau_2)[\xi := \tau']$

- Case Q-App: $\tau = \tau_1\ \tau_2$ and $\tau'' = \tau_1'\ \tau_2'$

  (a) By inversion: $\tau_1 \equiv \tau_1'$ and $\tau_2 \equiv \tau_2'$

  (b) By (a) and induction: $\tau_1[\xi := \tau'] \equiv \tau_1'[\xi := \tau']$ and $\tau_2[\xi := \tau'] \equiv \tau_2'[\xi := \tau']$

  (c) By (b) and Q-App: $\tau_1[\xi := \tau']\ \tau_2[\xi := \tau'] \equiv \tau_1'[\xi := \tau']\ \tau_2'[\xi := \tau']$

  (d) By (c): $(\tau_1\ \tau_2)[\xi := \tau'] \equiv (\tau_1'\ \tau_2')[\xi := \tau']$

- Case Q-Beta: $\tau = (\lambda\alpha{:}\kappa.\tau_1)\ \tau_2$ and $\tau'' = \tau_1[\alpha := \tau_2]$ (w.l.o.g. $\alpha \notin ftv(\tau')$ and $\alpha \neq \xi$)

  (a) By Q-Beta: $(\lambda\alpha{:}\kappa.\tau_1[\xi := \tau'])\ \tau_2[\xi := \tau'] \equiv \tau_1[\xi := \tau'][\alpha := \tau_2[\xi := \tau']]$

  (b) By (a) and Substitution: $(\lambda\alpha{:}\kappa.\tau_1[\xi := \tau'])\ \tau_2[\xi := \tau'] \equiv \tau_1[\alpha := \tau_2][\xi := \tau']$

  (c) By (b): $(\lambda\alpha{:}\kappa.\tau_1)[\xi := \tau']\ \tau_2[\xi := \tau'] \equiv \tau_1[\alpha := \tau_2][\xi := \tau']$

  (d) By (c): $((\lambda\alpha{:}\kappa.\tau_1)\ \tau_2)[\xi := \tau'] \equiv \tau_1[\alpha := \tau_2][\xi := \tau']$

- Case Q-Pair: $\tau = \langle\tau_1, \tau_2\rangle$ and $\tau'' = \langle\tau_1', \tau_2'\rangle$

  (a) By inversion: $\tau_1 \equiv \tau_1'$ and $\tau_2 \equiv \tau_2'$

  (b) By (a) and induction: $\tau_1[\xi := \tau'] \equiv \tau_1'[\xi := \tau']$ and $\tau_2[\xi := \tau'] \equiv \tau_2'[\xi := \tau']$

  (c) By (b) and Q-Pair: $\langle\tau_1[\xi := \tau'], \tau_2[\xi := \tau']\rangle \equiv \langle\tau_1'[\xi := \tau'], \tau_2'[\xi := \tau']\rangle$

  (d) By (c): $\langle\tau_1, \tau_2\rangle[\xi := \tau'] \equiv \langle\tau_1', \tau_2'\rangle[\xi := \tau']$

- Case Q-Proj1a: $\tau = \tau 1.1$ and $\tau'' = \tau_2.1$

  (a) By inversion: $\tau_1 \equiv \tau_2$

  (b) By (a) and induction: $\tau_1[\xi := \tau'] \equiv \tau_2[\xi := \tau']$

  (c) By (b) and Q-Proj1a: $\tau_1[\xi := \tau'].1 \equiv \tau_2[\xi := \tau'].1$

  (d) By (c): $\tau_1.1[\xi := \tau'] \equiv \tau_2.1[\xi := \tau']$

- Case Q-Proj1b: $\tau = \langle\tau', \tau_2\rangle.1$

  (a) By Q-Proj1b: $\langle\tau'[\xi := \tau'], \tau_2[\xi := \tau']\rangle.1 \equiv \tau'[\xi := \tau']$

  (b) By (a): $\langle\tau', \tau_2\rangle.1[\xi := \tau'] \equiv \tau'[\xi := \tau']$

- Case Q-Proj2a: $\tau = \tau_1.2$ and $\tau' = \tau_2.2$

  (a) By inversion: $\tau_1 \equiv \tau_2$

  (b) By (a) and induction: $\tau_1[\xi := \tau'] \equiv \tau_2[\xi := \tau']$

  (c) By (b) and Q-Proj2a: $\tau_1[\xi := \tau'].2 \equiv \tau_2[\xi := \tau'].2$

  (d) By (c): $\tau_1.2[\xi := \tau'] \equiv \tau_2.2[\xi := \tau']$

- Case Q-Proj2b: $\tau = \langle\tau_1, \tau'\rangle.2$

  (a) By Q-Proj2b: $\langle\tau_1[\xi := \tau'], \tau'[\xi := \tau']\rangle.2 \equiv \tau'[\xi := \tau']$

  (b) By (a): $\langle\tau_1, \tau'\rangle.2[\xi := \tau'] \equiv \tau'[\xi := \tau']$

□

Proposition 8 (Term Substitution). *If $\Gamma \vdash e' : \tau'$ and $\Gamma, x{:}\tau', \Gamma' \vdash e : \tau$, then $\Gamma, \Gamma' \vdash e[x := e'] : \tau$.*

Proof. By induction on the derivation of $\Gamma_1, x{:}\tau', \Gamma_2 \vdash e : \tau$.

- Case T-Equiv:

  1. By inversion: $\Gamma_1, x{:}\tau', \Gamma_2 \vdash e : \tau_1$ and $\tau_1 \equiv \tau$ and $\Gamma_1, x{:}\tau', \Gamma_2 \vdash \tau : \Omega$
  2. By (1), assumption, and induction: $\Gamma_1, \Gamma_2 \vdash e[x := e'] : \tau_1$

3. By (2) and Environment Validity: $\Gamma_1, \Gamma_2 \vdash \Box$
4. By (1) and Variable Containment: $ftv(\tau) \subseteq dom(\Gamma_1, x{:}\tau', \Gamma_2)$ and hence $ftv(\tau) \subseteq dom(\Gamma_1, \Gamma_2)$
5. By (1), (3), (4), and Strengthening: $\Gamma_1, \Gamma_2 \vdash \tau : \Omega$
6. By (2), (1), (5), and T-Equiv: $\Gamma_1, \Gamma_2 \vdash e[x := e'] : \tau$

- Case T-Var:
  - Subcase $e = x$ and hence $\tau = \tau'$:
    1. By inversion: $\Gamma_1, x{:}\tau', \Gamma_2 \vdash \Box$
    2. By assumption: $\Gamma_1 \vdash e' : \tau$
    3. By assumption and Environment Validity: $\Gamma_1, x{:}\tau', \Gamma_2 \vdash \Box$
    4. By (3) and Strengthening: $\Gamma_1, \Gamma_2 \vdash \Box$
    5. By (2), (4), and Weakening: $\Gamma_1, \Gamma_2 \vdash e' : \tau$
  - Subcase $e = x' \neq x$:
    1. By inversion: $\tau = (\Gamma_1, x{:}\tau', \Gamma_2)(x')$ and $\Gamma_1, x{:}\tau', \Gamma_2 \vdash \Box$
    2. By (1): $(\Gamma_1, \Gamma_2)(x') = \tau$
    3. By (1) and Strengthening: $\Gamma_1, \Gamma_2 \vdash \Box$
    4. By (2), (3), and T-Var: $\Gamma_1, \Gamma_2 \vdash x' : \tau$

- Case T-Abs: $e = \lambda x'{:}\tau_1.e_1$ and $\tau = \tau_1 \to \tau_2$
  1. By inversion: $\Gamma_1, x{:}\tau', \Gamma_2, x'{:}\tau_1 \vdash e_1 : \tau_2$
  2. By (1), assumption, and induction: $\Gamma_1, \Gamma_2, x'{:}\tau_1 \vdash e_1[x := e'] : \tau_2$
  3. By (2) and T-Abs: $\Gamma_1, \Gamma_2 \vdash \lambda x'{:}\tau_1.e_1[x := e'] : \tau_1 \to \tau_2$
  4. By (1) and Environment Validity: $\Gamma_1, x{:}\tau', \Gamma_2, x'{:}\tau_1 \vdash \Box$
  5. By (4) and inversion of E-Term: $x' \notin dom(\Gamma_1, x{:}\tau', \Gamma_2, x'{:}\tau_1)$ and hence $x' \neq x$
  6. By assumption and Variable Containment: $fv(e') \subseteq dom(\Gamma_1)$
  7. By (5) and (6): $x' \notin fv(e')$
  8. By (3), (5), and (7): $\Gamma_1, \Gamma_2 \vdash (\lambda x'{:}\tau_1.e_1)[x := e'] : \tau_1 \to \tau_2$

- Case T-App: $e = e_1\ e_2$
  1. By inversion: $\Gamma_1, x{:}\tau', \Gamma_2 \vdash e_1 : \tau_2 \to \tau$ and $\Gamma_1, x{:}\tau', \Gamma_2 \vdash e_2 : \tau_2$
  2. By (1), assumption, and induction: $\Gamma_1, \Gamma_2 \vdash e_1[x := e'] : \tau_2 \to \tau$ and $\Gamma_1, \Gamma_2 \vdash e_2[x := e'] : \tau_2$
  3. By (2) and T-App: $\Gamma_1, \Gamma_2 \vdash e_1[x := e']\ e_2[x := e'] : \tau$
  4. By (3): $\Gamma_1, \Gamma_2 \vdash (e_1\ e_2)[x := e'] : \tau$

- Case T-Gen: $e = \lambda \alpha{:}\kappa.e_1$ and $\tau = \forall \alpha{:}\kappa.\tau_1$
  1. By inversion: $\Gamma_1, x{:}\tau', \Gamma_2, \alpha{:}\kappa \vdash e_1 : \tau_1$
  2. By (1), assumption, and induction: $\Gamma_1, \Gamma_2, \alpha{:}\kappa \vdash e_1[x := e'] : \tau_1$
  3. By (2) and T-Gen: $\Gamma_1, \Gamma_2 \vdash \lambda \alpha{:}\kappa.e_1[x := e'] : \forall \alpha{:}\kappa.\tau_1$
  4. By (3): $\Gamma_1, \Gamma_2 \vdash (\lambda \alpha{:}\kappa.e_1)[x := e'] : \forall \alpha{:}\kappa.\tau_1$

- Case T-Inst: $e = e_1\ \tau_2$

1. By inversion: $\Gamma_1, x{:}\tau', \Gamma_2 \vdash e_1 : \forall\alpha{:}\kappa.\tau_1$ and $\Gamma_1, x{:}\tau', \Gamma_2 \vdash \tau_2 : \kappa$ where $\tau = \tau_1[\alpha := \tau_2]$
2. By (1), assumption, and induction: $\Gamma_1, \Gamma_2 \vdash e_1[x := e'] : \forall\alpha{:}\kappa.\tau_1$
3. By (2) and Environment Validity: $\Gamma_1, \Gamma_2 \vdash \Box$
4. By (1) and Variable Containment: $ftv(\tau_2) \subseteq dom(\Gamma_1, x{:}\tau', \Gamma_2)$ and hence $ftv(\tau_2) \subseteq dom(\Gamma_1, \Gamma_2)$
5. By (1), (3), (4), and Strengthening: $\Gamma_1, \Gamma_2 \vdash \tau_2 : \kappa$
6. By (2), (5), and T-Inst: $\Gamma_1, \Gamma_1 \vdash e_1[x := e']\ \tau_2 : \tau_1[\alpha := \tau_2]$
7. By (6): $\Gamma_1, \Gamma_1 \vdash (e_1\ \tau_2)[x := e'] : \tau_1[\alpha := \tau_2]$

- Case T-Pair: $e = \langle e_1, e_2\rangle$ and $\tau = \tau_1 \times \tau_2$

1. By inversion: $\Gamma_1, x{:}\tau', \Gamma_2 \vdash e_1 : \tau_1$ and $\Gamma_1, x{:}\tau', \Gamma_2 \vdash e_2 : \tau_2$
2. By (1), assumption, and induction: $\Gamma_1, \Gamma_2 \vdash e_1[x := e'] : \tau_1$ and $\Gamma_1, \Gamma_2 \vdash e_2[x := e'] : \tau_2$
3. By (2) and T-Pair: $\Gamma_1, \Gamma_2 \vdash \langle e_1[x := e'], e_2[x := e']\rangle : \tau$
4. By (3): $\Gamma_1, \Gamma_2 \vdash \langle e_1, e_2\rangle[x := e'] : \tau$

- Case T-Proj: $e = \mathsf{let}\ \langle x_1, x_2\rangle = e_1\ \mathsf{in}\ e_2$

1. By inversion: $\Gamma_1, x{:}\tau', \Gamma_2 \vdash e_1 : \tau_1 \times \tau_2$ and $\Gamma_1, x{:}\tau', \Gamma_2, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_2 : \tau$
2. By (1), assumption, and induction: $\Gamma_1, \Gamma_2 \vdash e_1[x := e'] : \tau_1{\times}\tau_2$ and $\Gamma_1, \Gamma_2, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_2[x := e'] : \tau$
3. By (2) and T-Proj: $\Gamma_1, \Gamma_2 \vdash \mathsf{let}\ \langle x_1, x_2\rangle = e_1[x := e']\ \mathsf{in}\ e_2[x := e'] : \tau$
4. By (1) and Environment Validity: $\Gamma_1, x{:}\tau', \Gamma_2, x_1{:}\tau_1, x_2{:}\tau_2 \vdash \Box$
5. By (4) and inversion of E-Term: $x_2 \notin dom(\Gamma_1, x{:}\tau', \Gamma_2, x_1{:}\tau_1)$ and hence $x_2 \neq x$
6. By (4) and Subenvironment Validity: $\Gamma_1, x{:}\tau', \Gamma_2, x_1{:}\tau_1 \vdash \Box$
7. By (6) and inversion of E-Term: $x_1 \notin dom(\Gamma_1, x{:}\tau', \Gamma_2)$ and hence $x_1 \neq x$
8. By assumption and Variable Containment: $fv(e') \subseteq dom(\Gamma_1)$
9. By (5), (7), and (8): $x_1 \notin fv(e')$ and $x_2 \notin fv(e')$
10. By (3), (5), (7), and (9): $\Gamma_1, \Gamma_2 \vdash (\mathsf{let}\ \langle x_1, x_2\rangle = e_1\ \mathsf{in}\ e_2)[x := e'] : \tau$

- Case T-Close: $e = \langle \tau_1, e'\rangle{:}\tau$ where $\tau = \exists\alpha{:}\kappa.\tau_2$

1. By inversion:
   (a) $\Gamma_1, x{:}\tau', \Gamma_2 \vdash \tau_1 : \kappa$
   (b) $\Gamma_1, x{:}\tau', \Gamma_2 \vdash e' : \tau_2[\alpha := \tau_1]$
   (c) $\Gamma_1, x{:}\tau', \Gamma_2 \vdash \tau : \Omega$
2. By (1-b) and induction: $\Gamma_1, \Gamma_2 \vdash e'[x := e'] : \tau_2[\alpha := \tau_1]$
3. By (2) and Environment Validity: $\Gamma_1, \Gamma_2 \vdash \Box$
4. By (1-a) and Variable Containment: $ftv(\tau_1) \subseteq dom(\Gamma_1, x{:}\tau', \Gamma_2)$ and hence $ftv(\tau_1) \subseteq dom(\Gamma_1, \Gamma_2)$
5. By (1-a), (3), (4), and Strengthening: $\Gamma_1, \Gamma_2 \vdash \tau_1 : \kappa$
6. By (1-c) and Variable Containment: $ftv(\tau) \subseteq dom(\Gamma_1, x{:}\tau', \Gamma_2)$ and hence $ftv(\tau) \subseteq dom(\Gamma_1, \Gamma_2)$
7. By (1-c), (3), (6), and Strengthening: $\Gamma_1, \Gamma_2 \vdash \tau : \Omega$

8. By (5), (2), (7), and T-Close: $\Gamma_1, \Gamma_2 \vdash \langle \tau_1, e[x := e'] \rangle{:}\exists\alpha{:}\kappa.\tau_2 : \exists\alpha{:}\kappa.\tau_2$
9. By (8): $\Gamma_1, \Gamma_2 \vdash (\langle \tau_1, e \rangle{:}\exists\alpha{:}\kappa.\tau_2)[x := e'] : \exists\alpha{:}\kappa.\tau_2$

- Case T-Open: $e = \mathsf{let}\ \langle \alpha, x' \rangle = e_1\ \mathsf{in}\ e_2$

  1. By inversion: $\Gamma_1, x{:}\tau', \Gamma_2 \vdash e_1 : \exists\alpha{:}\kappa.\tau_1$ and $\Gamma_1, x{:}\tau', \Gamma_2, \alpha{:}\kappa, x'{:}\tau_1 \vdash e_2 : \tau$ where $\alpha \notin ftv(\tau)$
  2. By (1), assumption, and induction: $\Gamma_1, \Gamma_2 \vdash e_1[x := e'] : \exists\alpha{:}\kappa.\tau_1$ and $\Gamma_1, \Gamma_2, \alpha{:}\kappa, x'{:}\tau_1 \vdash e_2[x := e'] : \tau$
  3. By (2), (1), and T-Open: $\Gamma_1, \Gamma_2 \vdash \mathsf{let}\ \langle \alpha, x' \rangle = e_1[x := e']\ \mathsf{in}\ e_2[x := e'] : \tau$
  4. By (1) and Environment Validity: $\Gamma_1, x{:}\tau', \Gamma_2, \alpha{:}\kappa, x'{:}\tau_1 \vdash \square$
  5. By (4) and inversion of E-Term: $x' \notin dom(\Gamma_1, x{:}\tau', \Gamma_2, \alpha{:}\kappa)$ and hence $x' \neq x$
  6. By assumption and Variable Containment: $fv(e') \subseteq dom(\Gamma_1)$
  7. By (5) and (6): $x' \notin ftv(e')$
  8. By (3), (5), and (7): $\Gamma_1, \Gamma_2 \vdash (\mathsf{let}\ \langle \alpha, x' \rangle = e_1\ \mathsf{in}\ e_2)[x := e'] : \tau$

- Case T-Lazy: $e = \mathsf{lazy}\ \langle \zeta, x' \rangle = e_1\ \mathsf{in}\ e_2$

  1. By inversion: $\Gamma_1, x{:}\tau', \Gamma_2 \vdash e_1 : \exists\alpha{:}\kappa.\tau_1$ and $\Gamma_1, x{:}\tau', \Gamma_2, \zeta{:}\kappa, x'{:}\tau_1[\alpha := \zeta] \vdash e_2 : \tau$ where $\zeta \notin ftv(\tau)$
  2. By (1), assumption, and induction:
     $\Gamma_1, \Gamma_2 \vdash e_1[x := e'] : \exists\alpha{:}\kappa.\tau_1$ and $\Gamma_1, \Gamma_2, \zeta{:}\kappa, x'{:}\tau_1[\alpha := \zeta] \vdash e_2[x := e'] : \tau$
  3. By (2), (1), and T-Lazy: $\Gamma_1, \Gamma_2 \vdash \mathsf{lazy}\ \langle \zeta, x' \rangle = e_1[x := e']\ \mathsf{in}\ e_2[x := e'] : \tau$
  4. By (1) and Environment Validity: $\Gamma_1, x{:}\tau', \Gamma_2, \zeta{:}\kappa, x'{:}\tau_1[\alpha := \zeta] \vdash \square$
  5. By (4) and inversion of E-Term: $x' \notin dom(\Gamma_1, x{:}\tau', \Gamma_2, \zeta{:}\kappa)$ and hence $x' \neq x$
  6. By assumption and Variable Containment: $fv(e') \subseteq dom(\Gamma_1)$
  7. By (5) and (6): $x' \notin ftv(e')$
  8. By (3) and (5): $\Gamma_1, \Gamma_2 \vdash (\mathsf{lazy}\ \langle \zeta, x' \rangle = e_1\ \mathsf{in}\ e_2)[x := e'] : \tau$

- Case T-Case: $e = \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x'{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$

  1. By inversion:
     (a) $\Gamma_1, x{:}\tau', \Gamma_2 \vdash e_0 : \tau_0$
     (b) $\Gamma_1, x{:}\tau', \Gamma_2, x'{:}\tau_0' \vdash e_1 : \tau$
     (c) $\Gamma_1, x{:}\tau', \Gamma_2 \vdash e_2 : \tau$
  2. By (1), assumption, and induction:
     (a) $\Gamma_1, \Gamma_2 \vdash e_0[x := e'] : \tau_0$
     (b) $\Gamma_1, \Gamma_2, x'{:}\tau_0' \vdash e_1[x := e'] : \tau$
     (c) $\Gamma_1, \Gamma_2 \vdash e_2[x := e'] : \tau$
  3. By (2) and T-Case: $\Gamma_1, \Gamma_2 \vdash \mathsf{tcase}\ e_0[x := e']{:}\tau_0\ \mathsf{of}\ x'{:}\tau_0'\ \mathsf{then}\ e_1[x := e']\ \mathsf{else}\ e_2[x := e'] : \tau$
  4. By (1-b) and Environment Validity: $\Gamma_1, x{:}\tau', \Gamma_2, x'{:}\tau_0' \vdash \square$
  5. By (4) and inversion of E-Term: $x' \notin dom(\Gamma_1, x{:}\tau', \Gamma_2)$ and hence $x' \neq x$
  6. By assumption and Variable Containment: $fv(e') \subseteq dom(\Gamma_1)$
  7. By (5) and (6): $x' \notin ftv(e')$

8. By (3) and (5): $\Gamma_1, \Gamma_2 \vdash (\mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x'{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2)[x := e'] : \tau$

□

Proposition 9 (Validity)**.** *If* $\Gamma \vdash e : \tau$*, then* $\Gamma \vdash \tau : \Omega$*.*

Proof. By induction on the derivation.

- Case T-Equiv:
  1. By inversion: $\Gamma \vdash \tau : \Omega$
- Case T-Var: $e = x$
  1. By inversion: $\tau = \Gamma(x)$ and $\Gamma \vdash \Box$
  2. By (1): $\Gamma = \Gamma_1, x{:}\tau, \Gamma_2$
  3. By (1), (2), and Subenvironment Validity: $\Gamma_1, x{:}\tau \vdash \Box$
  4. By (3) and inversion of E-Term: $\Gamma_1 \vdash \tau : \Omega$
  5. By (1), (2), (4), and Weakening: $\Gamma \vdash \tau : \Omega$
- Case T-Abs: $e = \lambda x{:}\tau_1.e_1$ and $\tau = \tau_1 \to \tau_2$
  1. By inversion: $\Gamma, x{:}\tau_1 \vdash e_1 : \tau_2$
  2. By (1) and induction: $\Gamma, x{:}\tau_1 \vdash \tau_2 : \Omega$
  3. By assumption and Environment Validity: $\Gamma \vdash \Box$
  4. By (2) and Variable Containment: $ftv(\tau_2) \subseteq dom(\Gamma, x{:}\tau_1)$ and hence $ftv(\tau_2) \subseteq dom(\Gamma)$
  5. By (2), (3), (4), and Strengthening: $\Gamma \vdash \tau_2 : \Omega$
  6. By (2) and Environment Validity: $\Gamma, x{:}\tau_1 \vdash \Box$
  7. By (6) and inversion of E-Term: $\Gamma \vdash \tau_1 : \Omega$
  8. By (5), (7), and K-Arrow: $\Gamma \vdash \tau_1 \to \tau_2 : \Omega$
- Case T-App: $e = e_1\ e_2$
  1. By inversion: $\Gamma \vdash e_1 : \tau' \to \tau$
  2. By (1) and induction: $\Gamma \vdash \tau' \to \tau : \Omega$
  3. By (2) and inversion of K-Arrow: $\Gamma \vdash \tau : \Omega$
- Case T-Gen: $e = \lambda\alpha{:}\kappa.e_1$ and $\tau = \forall\alpha{:}\kappa.\tau_1$
  1. By inversion: $\Gamma, \alpha{:}\kappa \vdash e_1 : \tau_1$
  2. By (1) and induction: $\Gamma, \alpha{:}\kappa \vdash \tau_1 : \Omega$
  3. By (2) and K-Univ: $\Gamma \vdash \forall\alpha{:}\kappa.\tau_1 : \Omega$
- Case T-Inst: $e = e_1\ \tau_2$
  1. By inversion: $\Gamma \vdash e_1 : \forall\alpha{:}\kappa.\tau_1$ and $\Gamma \vdash \tau_2 : \kappa$ where $\tau = \tau_1[\alpha := \tau_2]$ (w.l.o.g. $\alpha \notin dom(\Gamma)$)
  2. By (1) and induction: $\Gamma \vdash \forall\alpha{:}\kappa.\tau_1 : \Omega$
  3. By (2), (1), and inversion of K-Univ: $\Gamma, \alpha{:}\kappa \vdash \tau_1 : \Omega$
  4. By (3), (1), and Type Substitution: $\Gamma \vdash \tau_1[\alpha := \tau_2] : \Omega$

- Case T-Pair: $e = \langle e_1, e_2 \rangle$ and $\tau = \tau_1 \times \tau_2$
  1. By inversion: $\Gamma \vdash e_1 : \tau_1$ and $\Gamma \vdash e_2 : \tau_2$
  2. By (1) and induction: $\Gamma \vdash \tau_1 : \Omega$ and $\Gamma \vdash \tau_2 : \Omega$
  3. By (2) and K-Times: $\Gamma \vdash \tau_1 \times \tau_2 : \Omega$
- Case T-Proj: $e = \mathsf{let}\ \langle x_1, x_2 \rangle = e_1\ \mathsf{in}\ e_2$
  1. By inversion: $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_2 : \tau$
  2. By (1) and induction: $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash \tau : \Omega$
  3. By assumption and Environment Validity: $\Gamma \vdash \Box$
  4. By (2) and Variable Containment: $ftv(\tau) \subseteq dom(\Gamma, x_1{:}\tau_1, x_2{:}\tau_2)$ and hence $ftv(\tau) \subseteq dom(\Gamma)$
  5. By (2), (3), (4), and Strengthening: $\Gamma \vdash \tau : \Omega$
- Case T-Close: $e = \langle \tau_1, e' \rangle{:}\tau$ where $\tau = \exists \alpha{:}\kappa.\tau_2$
  1. By inversion: $\Gamma \vdash \exists \alpha{:}\kappa.\tau_2 : \Omega$
- Case T-Open: $e = \mathsf{let}\ \langle \alpha, x \rangle = e_1\ \mathsf{in}\ e_2$
  1. By inversion: $\Gamma, \alpha{:}\kappa, x{:}\tau' \vdash e_2 : \tau$
  2. By (1) and induction: $\Gamma, \alpha{:}\kappa, x{:}\tau' \vdash \tau : \Omega$
  3. By assumption and Environment Validity: $\Gamma \vdash \Box$
  4. By assumption and Variable Containment: $ftv(\tau) \subseteq dom(\Gamma)$
  5. By (2), (3), (4), and Strengthening: $\Gamma \vdash \tau : \Omega$
- Case T-Lazy: $e = \mathsf{lazy}\ \langle \zeta, x \rangle = e_1\ \mathsf{in}\ e_2$
  1. By inversion: $\Gamma, \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta] \vdash e_2 : \tau$
  2. By (1) and induction: $\Gamma, \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta] \vdash \tau : \Omega$
  3. By assumption and Environment Validity: $\Gamma \vdash \Box$
  4. By assumption and Variable Containment: $ftv(\tau) \subseteq dom(\Gamma)$
  5. By (2), (3), (4), and Strengthening: $\Gamma \vdash \tau : \Omega$
- Case T-Case: $e = \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$
  1. By inversion: $\Gamma \vdash e_2 : \tau$
  2. By (1) and induction: $\Gamma \vdash \tau : \Omega$

□

Lemma 1 (Typing Inversion)**.**

- *If* $\Gamma \vdash x : \tau$*, then* $\tau \equiv \Gamma(x)$ *and* $\Gamma \vdash \Box$*.*
- *If* $\Gamma \vdash \lambda x{:}\tau.e : \tau'$ *and* $x \notin dom(\Gamma)$*, then* $\Gamma, x{:}\tau \vdash e : \tau''$ *and* $\tau' \equiv \tau \to \tau''$*.*
- *If* $\Gamma \vdash e_1\ e_2 : \tau$*, then* $\Gamma \vdash e_1 : \tau' \to \tau$ *and* $\Gamma \vdash e_2 : \tau'$*.*
- *If* $\Gamma \vdash \lambda \alpha{:}\kappa.e : \tau$ *and* $\alpha \notin dom(\Gamma)$*, then* $\Gamma, \alpha{:}\kappa \vdash e : \tau'$ *and* $\tau \equiv \forall \alpha{:}\kappa.\tau'$*.*
- *If* $\Gamma \vdash e\ \tau' : \tau$*, then* $\Gamma \vdash e : \forall \alpha{:}\kappa.\tau_1$ *where* $\alpha \notin dom(\Gamma)$ *and* $\tau \equiv \tau_1[\alpha := \tau']$ *where* $\Gamma \vdash \tau' : \kappa$*.*

- *If* $\Gamma \vdash \langle e_1, e_2\rangle : \tau$, *then* $\Gamma \vdash e_1 : \tau_1$ *and* $\Gamma \vdash e_2 : \tau_2$ *where* $\tau \equiv \tau_1 \times \tau_2$.
- *If* $\Gamma \vdash \langle \tau_1, e\rangle{:}\tau_2 : \tau$, *then* $\Gamma \vdash \tau_1 : \kappa$ *and* $\Gamma \vdash e : \tau_3[\alpha := \tau_1]$ *where* $\tau_2 = \exists\alpha{:}\kappa.\tau_3$ *and* $\tau \equiv \tau_2$ *and* $\Gamma \vdash \tau_2 : \Omega$.
- *If* $\Gamma \vdash \mathsf{let}\ \langle x_1, x_2\rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ e_2 : \tau$ *and* $\{x_1, x_2\} \cap dom(\Gamma) = \emptyset$, *then* $\Gamma \vdash e_1 : \tau_1 \times \tau_2$ *and* $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_2 : \tau$.
- *If* $\Gamma \vdash \mathsf{let}\ \langle \alpha, x\rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ e_2 : \tau$ *and* $\{\alpha, x\} \cap dom(\Gamma) = \emptyset$, *then* $\Gamma \vdash e_1 : \exists\alpha{:}\kappa.\tau'$ *and* $\Gamma, \alpha{:}\kappa, x{:}\tau' \vdash e_2 : \tau''$ *where* $\alpha \notin ftv(\tau'')$ *and* $\tau'' \equiv \tau$.
- *If* $\Gamma \vdash \mathsf{lazy}\ \langle \zeta, x\rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ e_2 : \tau$ *and* $\{\zeta, x\} \cap dom(\Gamma) = \emptyset$, *then* $\Gamma \vdash e_1 : \exists\alpha{:}\kappa.\tau'$ *where* $\alpha \notin dom(\Gamma)$ *and* $\Gamma, \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta] \vdash e_2 : \tau''$ *where* $\zeta \notin ftv(\tau'')$ *and* $\tau'' \equiv \tau$.
- *If* $\Gamma \vdash \mathsf{tcase}\ e{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau$ *and* $x \notin dom(\Gamma)$, *then* $\Gamma \vdash e : \tau_0$ *and* $\Gamma, x{:}\tau_0' \vdash e_1 : \tau$ *and* $\Gamma \vdash e_2 : \tau$.

Proof. By induction on the corresponding derivation.

- Let $\Gamma \vdash x : \tau$.
  - Case T-Var: $\tau = \Gamma(x)$
    1. By inversion: $\Gamma \vdash \Box$
    2. By Q-Refl: $\tau \equiv \Gamma(x)$ and $\Gamma \vdash \Box$
  - Case T-Equiv:
    1. By inversion: $\Gamma \vdash x : \tau'$ and $\tau' \equiv \tau$
    2. By (1) and induction: $\tau' \equiv \Gamma(x)$ and $\Gamma \vdash \Box$
    3. By (1), (2), Q-Symm, and Q-Trans: $\tau \equiv \Gamma(x)$
- Let $\Gamma \vdash \lambda x{:}\tau.e : \tau'$ where $x \notin dom(\Gamma)$.
  - Case T-Abs: $\tau' = \tau \to \tau''$
    1. By inversion: $\Gamma, x{:}\tau \vdash e : \tau''$
    2. By Q-Refl: $\tau' \equiv \tau \to \tau''$
  - Case T-Equiv:
    1. By inversion: $\Gamma \vdash \lambda x{:}\tau.e : \tau_1$ and $\tau_1 \equiv \tau'$
    2. By (1) and induction: $\Gamma, x{:}\tau \vdash e : \tau''$ and $\tau_1 \equiv \tau \to \tau''$
    3. By (1), (2), Q-Symm, and Q-Trans: $\tau' \equiv \tau \to \tau''$
- Let $\Gamma \vdash e_1\ e_2 : \tau$.
  - Case T-App:
    1. By inversion: $\Gamma \vdash e_1 : \tau' \to \tau$ and $\Gamma \vdash e_2 : \tau'$
  - Case T-Equiv:
    1. By inversion: $\Gamma \vdash e_1\ e_2 : \tau''$ and $\tau'' \equiv \tau$ where $\Gamma \vdash \tau : \Omega$
    2. By (1) and induction: $\Gamma \vdash e_1 : \tau' \to \tau''$ and $\Gamma \vdash e_2 : \tau'$
    3. By (1), (2), Q-Refl, and Q-Arrow: $\tau' \to \tau'' \equiv \tau' \to \tau$
    4. By (2) and Validity: $\Gamma \vdash \tau' : \Omega$
    5. By (1), (4), and K-Arrow: $\Gamma \vdash \tau' \to \tau : \Omega$
    6. By (2), (3), (5), and T-Equiv: $\Gamma \vdash e_1 : \tau' \to \tau$
- Let $\Gamma \vdash \lambda\alpha{:}\kappa.e : \tau$ where $\alpha \notin dom(\Gamma)$.

- Case T-Gen: $\tau = \forall\alpha{:}\kappa.\tau'$
  1. By inversion: $\Gamma, \alpha{:}\kappa \vdash e : \tau'$
  2. By Q-Refl: $\tau \equiv \forall\alpha{:}\kappa.\tau'$
- Case T-Equiv:
  1. By inversion: $\Gamma \vdash \lambda\alpha{:}\kappa.e : \tau''$ and $\tau'' \equiv \tau$
  2. By (1) and induction: $\Gamma, \alpha{:}\kappa \vdash e : \tau'$ and $\tau'' \equiv \forall\alpha{:}\kappa.\tau'$
  3. By (1), (2), Q-Symm, and Q-Trans: $\tau \equiv \forall\alpha{:}\kappa.\tau'$

- Let $\Gamma \vdash e\ \tau' : \tau$.
  - Case T-Inst:
    1. By inversion: $\Gamma \vdash e : \forall\alpha{:}\kappa.\tau_1$ (w.l.o.g. $\alpha \notin dom(\Gamma)$) and $\tau = \tau_1[\alpha := \tau']$ where $\Gamma \vdash \tau' : \kappa$
    2. By (1) and Q-Refl: $\tau \equiv \tau_1[\alpha := \tau']$
  - Case T-Equiv:
    1. By inversion: $\Gamma \vdash e\ \tau' : \tau''$ and $\tau'' \equiv \tau$
    2. By (1) and induction: $\Gamma \vdash e : \forall\alpha{:}\kappa.\tau_1$ where $\alpha \notin dom(\Gamma)$ and $\tau'' \equiv \tau_1[\alpha := \tau']$ where $\Gamma \vdash \tau' : \kappa$
    3. By (1), (2), Q-Symm, and Q-Trans: $\tau \equiv \tau_1[\alpha := \tau']$

- Let $\Gamma \vdash \langle e_1, e_2\rangle : \tau$.
  - Case T-Pair: $\tau = \tau_1 \times \tau_2$
    1. By inversion: $\Gamma \vdash e_1 : \tau_1$ and $\Gamma \vdash e_2 : \tau_2$
    2. By (1) and Q-Refl: $\tau \equiv \tau_1 \times \tau_2$
  - Case T-Equiv:
    1. By inversion: $\Gamma \vdash \langle e_1, e_2\rangle : \tau'$ and $\tau' \equiv \tau$
    2. By (1) and induction: $\Gamma \vdash e_1 : \tau_1$ and $\Gamma \vdash e_2 : \tau_2$ where $\tau' \equiv \tau_1 \times \tau_2$
    3. By (1), (2), Q-Symm, and Q-Trans: $\tau \equiv \tau_1 \times \tau_2$

- Let $\Gamma \vdash \langle \tau_1, e\rangle{:}\tau_2 : \tau$.
  - Case T-Close: $\tau = \tau_2 = \exists\alpha{:}\kappa.\tau_3$
    1. By inversion: $\Gamma \vdash \tau_1 : \kappa$ and $\Gamma \vdash e : \tau_3[\alpha := \tau_1]$ where $\Gamma \vdash \tau_2 : \Omega$
    2. By Q-Refl: $\tau \equiv \tau_2$
  - Case T-Equiv:
    1. By inversion: $\Gamma \vdash \langle \tau_1, e\rangle{:}\tau_2 : \tau'$ and $\tau' \equiv \tau$
    2. By (1) and induction: $\Gamma \vdash \tau_1 : \kappa$ and $\Gamma \vdash e : \tau_3[\alpha := \tau_1]$ where $\tau_2 = \exists\alpha{:}\kappa.\tau_3$ and $\tau' \equiv \tau_2$ and $\Gamma \vdash \tau_2 : \Omega$.
    3. By (1), (2), Q-Symm, and Q-Trans: $\tau \equiv \tau_2$

- Let $\Gamma \vdash \mathsf{let}\ \langle x_1, x_2\rangle = e_1\ \mathsf{in}\ e_2 : \tau$ where $\{x_1, x_2\} \cap dom(\Gamma) = \emptyset$.
  - Case T-Proj:
    1. By inversion: $\Gamma \vdash e_1 : \tau_1 \times \tau_2$ and $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_2 : \tau$
  - Case T-Equiv:
    1. By inversion: $\Gamma \vdash \mathsf{let}\ \langle x_1, x_2\rangle = e_1\ \mathsf{in}\ e_2 : \tau'$ and $\tau' \equiv \tau$ where $\Gamma \vdash \tau : \Omega$
    2. By (1) and induction: $\Gamma \vdash e_1 : \tau_1 \times \tau_2$ and $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_2 : \tau'$

3. By (2) and Environment Validity: $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash \Box$
4. By (1), (3), and Weakening: $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash \tau : \Omega$
5. By (1), (2), (4), and T-Equiv: $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_2 : \tau$

- Let $\Gamma \vdash \mathsf{let}\ \langle \alpha, x\rangle = e_1\ \mathsf{in}\ e_2 : \tau$ where $\{\alpha, x\} \cap dom(\Gamma) = \emptyset$.
  - Case T-Open:
    1. By inversion: $\Gamma \vdash e_1 : \exists \alpha{:}\kappa.\tau_1$ and $\Gamma, \alpha{:}\kappa, x{:}\tau_1 \vdash e_2 : \tau$ where $\alpha \notin ftv(\Gamma)$
    2. By Q-Refl: $\tau \equiv \tau$
  - Case T-Equiv:
    1. By inversion: $\Gamma \vdash \mathsf{let}\ \langle \alpha, x\rangle = e_1\ \mathsf{in}\ e_2 : \tau'$ and $\tau' \equiv \tau$
    2. By (1) and induction: $\Gamma \vdash e_1 : \exists \alpha{:}\kappa.\tau_1$ and $\Gamma, \alpha{:}\kappa, x{:}\tau_1 \vdash e_2 : \tau'$ where $\alpha \notin ftv(\tau')$
- Let $\Gamma \vdash \mathsf{lazy}\ \langle \zeta, x\rangle = e_1\ \mathsf{in}\ e_2 : \tau$ where $\{\zeta, x\} \cap dom(\Gamma) = \emptyset$.
  - Case T-Lazy:
    1. By inversion: $\Gamma \vdash e_1 : \exists \alpha{:}\kappa.\tau_1$ (w.l.o.g. $\alpha \notin dom(\Gamma)$) and $\Gamma, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash e_2 : \tau$ where $\zeta \notin ftv(\Gamma)$
    2. By Q-Refl: $\tau \equiv \tau$
  - Case T-Equiv:
    1. By inversion: $\Gamma \vdash \mathsf{lazy}\ \langle \zeta, x\rangle = e_1\ \mathsf{in}\ e_2 : \tau'$ and $\tau' \equiv \tau$
    2. By (1) and induction: $\Gamma \vdash e_1 : \exists \alpha{:}\kappa.\tau_1$ with $\alpha \notin dom(\Gamma)$ and $\Gamma, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash e_2 : \tau'$ where $\zeta \notin ftv(\tau')$
- Let $\Gamma \vdash \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau$ where $x \notin dom(\Gamma)$.
  - Case T-Case:
    1. By inversion:
       (a) $\Gamma \vdash e_0 : \tau_0$
       (b) $\Gamma, x{:}\tau_0' \vdash e_1 : \tau$
       (c) $\Gamma \vdash e_2 : \tau$
  - Case T-Equiv:
    1. By inversion: $\Gamma \vdash \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau'$ where $\tau' \equiv \tau$ and $\Gamma \vdash \tau : \Omega$
    2. By (1) and induction:
       (a) $\Gamma \vdash e_0 : \tau_0$
       (b) $\Gamma, x{:}\tau_0' \vdash e_1 : \tau'$
       (c) $\Gamma \vdash e_2 : \tau'$
    3. By (2b) and Environment Validity: $\Gamma, x{:}\tau_0' \vdash \Box$
    4. By (3), (1), and Weakening: $\Gamma, x{:}\tau_0' \vdash \tau : \Omega$
    5. By (4), (1), (2b), and T-Equiv: $\Gamma, x{:}\tau_0' \vdash e_1 : \tau$
    6. By (1), (2c), and T-Equiv: $\Gamma \vdash e_2 : \tau$

□

Lemma 2 (Parallel Reduction).
- *$\tau_1 \equiv \tau_2$ iff $\tau_1 \Leftrightarrow^* \tau_2$*
- *If $\tau_1 \equiv \tau_2$, then there is some τ such that $\tau_1 \Rrightarrow^* \tau$ and $\tau_2 \Rrightarrow^* \tau$.*

Proof. See [18]. □

Lemma 3 (Preservation of Shapes Under Reduction)**.**

1. *If $\xi \Rrightarrow^* \tau'$, then $\tau' = \xi$.*
2. *If $\tau_1 \to \tau_2 \Rrightarrow^* \tau'$, then $\tau' = \tau_1' \to \tau_2'$ with $\tau_1 \Rrightarrow^* \tau_1'$ and $\tau_2 \Rrightarrow^* \tau_2'$.*
3. *If $\tau_1 \times \tau_2 \Rrightarrow^* \tau'$, then $\tau' = \tau_1' \times \tau_2'$ with $\tau_1 \Rrightarrow^* \tau_1'$ and $\tau_2 \Rrightarrow^* \tau_2'$.*
4. *If $\langle \tau_1, \tau_2 \rangle \Rrightarrow^* \tau'$, then $\tau' = \langle \tau_1', \tau_2' \rangle$ with $\tau_1 \Rrightarrow^* \tau_1'$ and $\tau_2 \Rrightarrow^* \tau_2'$.*
5. *If $\forall\alpha{:}\kappa.\tau_1 \Rrightarrow^* \tau'$, then $\tau' = \forall\alpha{:}\kappa.\tau_1'$ with $\tau_1 \Rrightarrow^* \tau_1'$.*
6. *If $\exists\alpha{:}\kappa.\tau_1 \Rrightarrow^* \tau'$, then $\tau' = \exists\alpha{:}\kappa.\tau_1'$ with $\tau_1 \Rrightarrow^* \tau_1'$.*
7. *If $\lambda\alpha{:}\kappa.\tau_1 \Rrightarrow^* \tau'$, then $\tau' = \lambda\alpha{:}\kappa.\tau_1'$ with $\tau_1 \Rrightarrow^* \tau_1'$.*

Proof. See [18]. □

Lemma 4 (Type Equivalence Inversion)**.**

- *If $\exists\alpha{:}\kappa_1.\tau_1 \equiv \exists\alpha{:}\kappa_2.\tau_2$, then $\kappa_1 = \kappa_2$ and $\tau_1 \equiv \tau_2$.*
- *If $\tau_1 \to \tau_2 \equiv \tau_1' \to \tau_2'$, then $\tau_1 \equiv \tau_1'$ and $\tau_2 \equiv \tau_2'$.*
- *If $\forall\alpha{:}\kappa_1.\tau_1 \equiv \forall\alpha{:}\kappa_2.\tau_2$, then $\kappa_1 = \kappa_2$ and $\tau_1 \equiv \tau_2$.*
- *If $\tau_1 \times \tau_2 \equiv \tau_1' \times \tau_2'$, then $\tau_1 \equiv \tau_1'$ and $\tau_2 \equiv \tau_2'$.*
- *If $\langle \tau_1, \tau_2 \rangle \equiv \langle \tau_1', \tau_2' \rangle$, then $\tau_1 \equiv \tau_1'$ and $\tau_2 \equiv \tau_2'$.*

Proof. Follows from lemma 3 (Preservation of Shapes Under Reduction).

- Let $\exists\alpha{:}\kappa_1.\tau_1 \equiv \exists\alpha{:}\kappa_2.\tau_2$.
  1. By Parallel Reduction: $\exists\alpha{:}\kappa_1.\tau_1 \Rrightarrow^* \tau$ and $\exists\alpha{:}\kappa_2.\tau_2 \Rrightarrow^* \tau$.
  2. By (1) and Preservation of Shapes Under Reduction: $\tau = \exists\alpha{:}\kappa_1.\tau_1'$ with $\tau_1 \Rrightarrow^* \tau_1'$ and $\tau = \exists\alpha{:}\kappa_2.\tau_2'$ with $\tau_2 \Rrightarrow^* \tau_2'$
  3. By (2): $\kappa_1 = \kappa_2$ and $\tau_1' = \tau_2'$
  4. By (3) and (2): $\tau_1 \Lleftarrow\!\!\Rrightarrow^* \tau_2$
  5. By (4) and Parallel Reduction: $\tau_1 \equiv \tau_2$
- Other parts analogous.

□

Lemma 5 (Shape Consistency)**.**

- *If $\tau \equiv \tau_1 \to \tau_2$, then $\tau = \tau_1' \to \tau_2'$ or $\tau = \tau_1'\ \tau_2'$ or $\tau = \tau'.1$ or $\tau = \tau'.2$.*
- *If $\tau \equiv \tau_1 \times \tau_2$, then $\tau = \tau_1' \times \tau_2'$ or $\tau = \tau_1'\ \tau_2'$ or $\tau = \tau'.1$ or $\tau = \tau'.2$.*
- *If $\tau \equiv \forall\alpha{:}\kappa.\tau_1$, then $\tau = \forall\alpha{:}\kappa.\tau_1'$ or $\tau = \tau_1'\ \tau_2'$ or $\tau = \tau'.1$ or $\tau = \tau'.2$.*
- *If $\tau \equiv \exists\alpha{:}\kappa.\tau_1$, then $\tau = \exists\alpha{:}\kappa.\tau_1'$ or $\tau = \tau_1'\ \tau_2'$ or $\tau = \tau'.1$ or $\tau = \tau'.2$.*

Proof.

- Let $\tau \equiv \tau_1 \to \tau_2$. We show that any other possibility leads to a contradiction.
  - Assume $\tau = \xi$:

1. By Parallel Reduction: $\xi \Rrightarrow \tau'$ and $\tau_1 \to \tau_2 \Rrightarrow \tau'$
2. By (1) and Preservation of Shapes Under Reduction: $\xi = \tau' = \tau_{11} \to \tau_{22}$, which is syntactically impossible

– Assume $\tau = \tau_1' \times \tau_2'$:

1. By Parallel Reduction: $\tau_1' \times \tau_2' \Rrightarrow \tau'$ and $\tau_1 \to \tau_2 \Rrightarrow \tau'$
2. By (1) and Preservation of Shapes Under Reduction: $\tau_{11}' \times \tau_{22}' = \tau' = \tau_{11} \to \tau_{22}$, which is syntactically impossible

– Assume $\tau = \forall\alpha{:}\kappa.\tau'$:

1. By Parallel Reduction: $\forall\alpha{:}\kappa.\tau_1' \Rrightarrow \tau'$ and $\tau_1 \to \tau_2 \Rrightarrow \tau'$
2. By (1) and Preservation of Shapes Under Reduction: $\forall\alpha{:}\kappa.\tau_2' = \tau' = \tau_{11} \to \tau_{22}$, which is syntactically impossible

– Assume $\tau = \exists\alpha{:}\kappa.\tau'$:

1. By Parallel Reduction: $\exists\alpha{:}\kappa.\tau_1' \Rrightarrow \tau'$ and $\tau_1 \to \tau_2 \Rrightarrow \tau'$
2. By (1) and Preservation of Shapes Under Reduction: $\exists\alpha{:}\kappa.\tau_2' = \tau' = \tau_{11} \to \tau_{22}$, which is syntactically impossible

– Assume $\tau = \lambda\alpha{:}\kappa.\tau'$:

1. By Parallel Reduction: $\lambda\alpha{:}\kappa.\tau_1' \Rrightarrow \tau'$ and $\tau_1 \to \tau_2 \Rrightarrow \tau'$
2. By (1) and Preservation of Shapes Under Reduction: $\lambda\alpha{:}\kappa.\tau_2' = \tau' = \tau_{11} \to \tau_{22}$, which is syntactically impossible

– Assume $\tau = \langle\tau_1', \tau_2'\rangle$:

1. By Parallel Reduction: $\langle\tau_1', \tau_2'\rangle \Rrightarrow \tau'$ and $\tau_1 \to \tau_2 \Rrightarrow \tau'$
2. By (1) and Preservation of Shapes Under Reduction: $\langle\tau_{11}', \tau_{22}\rangle' = \tau' = \tau_{11} \to \tau_{22}$, which is syntactically impossible

- Other parts analogous.

□

Proposition 10 (Canonical Values)**.** *Let $\Gamma \vdash v : \tau$ where v is not a variable.*

- *If $\tau \equiv \tau_1 \to \tau_2$, then $v = \lambda x{:}\tau_1'.e$.*
- *If $\tau \equiv \tau_1 \times \tau_2$, then $v = \langle v_1, v_2\rangle$.*
- *If $\tau \equiv \forall\alpha{:}\kappa.\tau_1$, then $v = \lambda\alpha{:}\kappa.e$.*
- *If $\tau \equiv \exists\alpha{:}\kappa.\tau_1$, then $v = \langle\tau_2, v'\rangle{:}\tau'$.*

Proof. By induction on the typing derivation.

- Let $\tau \equiv \tau_1 \to \tau_2$.
  – By Shape Consistency: $\tau = \tau_1' \to \tau_2'$ or $\tau = \tau_1'\ \tau_2'$ or $\tau = \tau'.1$ or $\tau = \tau'.2$
  – Subcase $\tau = \tau_1' \to \tau_2'$ and T-Arrow: $v = \lambda x{:}\tau_1'.e$
  – Subcase $\tau = \tau_1' \to \tau_2'$ and T-Equiv:
    1. By inversion: $\Gamma \vdash v : \tau'$ where $\tau' \equiv \tau$
    2. By (1), assumption, and Q-Trans: $\tau' \equiv \tau_1 \to \tau_2$
    3. By (1), (2), and induction: $v = \lambda x{:}\tau_1''.e$
  – Subcase $\tau = \tau_1'\ \tau_2'$ and T-Equiv:

1. By inversion: $\Gamma \vdash v : \tau'$ where $\tau' \equiv \tau$
2. By (1), assumption, and Q-Trans: $\tau' \equiv \tau_1 \to \tau_2$
3. By (1), (2), and induction: $v = \lambda x{:}\tau_1''.e$

– Subcase $\tau = \tau'.1$ and T-Equiv:

1. By inversion: $\Gamma \vdash v : \tau''$ where $\tau'' \equiv \tau$
2. By (1), assumption, and Q-Trans: $\tau'' \equiv \tau_1 \to \tau_2$
3. By (1), (2), and induction: $v = \lambda x{:}\tau_1''.e$

– Subcase $\tau = \tau'.2$ and T-Equiv:

1. By inversion: $\Gamma \vdash v : \tau''$ where $\tau'' \equiv \tau$
2. By (1), assumption, and Q-Trans: $\tau'' \equiv \tau_1 \to \tau_2$
3. By (1), (2), and induction: $v = \lambda x{:}\tau_1''.e$

- Let $\tau \equiv \tau_1 \times \tau_2$.

  – By Shape Consistency: $\tau = \tau_1' \times \tau_2'$ or $\tau = \tau_1'\ \tau_2'$ or $\tau = \tau'.1$ or $\tau = \tau'.2$

  – Subcase $\tau = \tau_1' \times \tau_2'$ and TTimes: $v = \langle v_1, v_2 \rangle$

  – Subcase $\tau = \tau_1' \times \tau_2'$ and T-Equiv:

  1. By inversion: $\Gamma \vdash v : \tau'$ where $\tau' \equiv \tau$
  2. By (1), assumption, and Q-Trans: $\tau' \equiv \tau_1 \times \tau_2$
  3. By (1), (2), and induction: $v = \langle v_1, v_2 \rangle$

  – Subcase $\tau = \tau_1'\ \tau_2'$ and T-Equiv:

  1. By inversion: $\Gamma \vdash v : \tau'$ where $\tau' \equiv \tau$
  2. By (1), assumption, and Q-Trans: $\tau' \equiv \tau_1 \times \tau_2$
  3. By (1), (2), and induction: $v = \langle v_1, v_2 \rangle$

  – Subcase $\tau = \tau'.1$ and T-Equiv:

  1. By inversion: $\Gamma \vdash v : \tau''$ where $\tau'' \equiv \tau$
  2. By (1), assumption, and Q-Trans: $\tau'' \equiv \tau_1 \times \tau_2$
  3. By (1), (2), and induction: $v = \langle v_1, v_2 \rangle$

  – Subcase $\tau = \tau'.2$ and T-Equiv:

  1. By inversion: $\Gamma \vdash v : \tau''$ where $\tau'' \equiv \tau$
  2. By (1), assumption, and Q-Trans: $\tau'' \equiv \tau_1 \times \tau_2$
  3. By (1), (2), and induction: $v = \langle v_1, v_2 \rangle$

- Let $\tau \equiv \forall\alpha{:}\kappa.\tau'$.

  – By Shape Consistency: $\tau = \forall\alpha{:}\kappa.\tau'$ or $\tau = \tau_1\ \tau_2$ or $\tau = \tau''.1$ or $\tau = \tau''.2$

  – Subcase $\tau = \forall\alpha{:}\kappa.\tau'$ and T-Gen: $v = \lambda\alpha{:}\kappa.e$

  – Subcase $\tau = \forall\alpha{:}\kappa.\tau'$ and T-Equiv:

  1. By inversion: $\Gamma \vdash v : \tau''$ where $\tau'' \equiv \tau$
  2. By (1), assumption, and Q-Trans: $\tau'' \equiv \forall\alpha{:}\kappa.\tau'$
  3. By (1), (2), and induction: $v = \lambda\alpha{:}\kappa.e$

  – Subcase $\tau = \tau_1\ \tau_2$ and T-Equiv:

  1. By inversion: $\Gamma \vdash v : \tau''$ where $\tau'' \equiv \tau$
  2. By (1), assumption, and Q-Trans: $\tau'' \equiv \forall\alpha{:}\kappa.\tau'$

    3. By (1), (2), and induction: $v = \lambda\alpha{:}\kappa.e$
  - Subcase $\tau = \tau''.1$ and T-Equiv:
    1. By inversion: $\Gamma \vdash v : \tau'''$ where $\tau''' \equiv \tau$
    2. By (1), assumption, and Q-Trans: $\tau''' \equiv \forall\alpha{:}\kappa.\tau'$
    3. By (1), (2), and induction: $v = \lambda\alpha{:}\kappa.e$
  - Subcase $\tau = \tau''.2$ and T-Equiv:
    1. By inversion: $\Gamma \vdash v : \tau'''$ where $\tau''' \equiv \tau$
    2. By (1), assumption, and Q-Trans: $\tau''' \equiv \forall\alpha{:}\kappa.\tau'$
    3. By (1), (2), and induction: $v = \lambda\alpha{:}\kappa.e$

- Let $\tau \equiv \exists\alpha{:}\kappa.\tau'$.
  - By Shape Consistency: $\tau = \exists\alpha{:}\kappa.\tau'$ or $\tau = \tau_1\ \tau_2$ or $\tau = \tau''.1$ or $\tau = \tau''.2$
  - Subcase $\tau = \exists\alpha{:}\kappa.\tau'$ and T-Close: $v = \langle\tau_1, v'\rangle{:}\tau_2$
  - Subcase $\tau = \exists\alpha{:}\kappa.\tau'$ and T-Equiv:
    1. By inversion: $\Gamma \vdash v : \tau''$ where $\tau'' \equiv \tau$
    2. By (1), assumption, and Q-Trans: $\tau'' \equiv \exists\alpha{:}\kappa.\tau'$
    3. By (1), (2), and induction: $v = \langle\tau_1, v'\rangle{:}\tau_2$
  - Subcase $\tau = \tau_1\ \tau_2$ and T-Equiv:
    1. By inversion: $\Gamma \vdash v : \tau''$ where $\tau'' \equiv \tau$
    2. By (1), assumption, and Q-Trans: $\tau'' \equiv \exists\alpha{:}\kappa.\tau'$
    3. By (1), (2), and induction: $v = \langle\tau_1, v'\rangle{:}\tau_2$
  - Subcase $\tau = \tau''.1$ and T-Equiv:
    1. By inversion: $\Gamma \vdash v : \tau'''$ where $\tau''' \equiv \tau$
    2. By (1), assumption, and Q-Trans: $\tau''' \equiv \exists\alpha{:}\kappa.\tau'$
    3. By (1), (2), and induction: $v = \langle\tau_1, v'\rangle{:}\tau_2$
  - Subcase $\tau = \tau''.2$ and T-Equiv:
    1. By inversion: $\Gamma \vdash v : \tau'''$ where $\tau''' \equiv \tau$
    2. By (1), assumption, and Q-Trans: $\tau''' \equiv \exists\alpha{:}\kappa.\tau'$
    3. By (1), (2), and induction: $v = \langle\tau_1, v'\rangle{:}\tau_2$

$\square$

Proposition 11 (Uniqueness of Kinds). *If $\Gamma \vdash \tau : \kappa$ and $\Gamma \vdash \tau : \kappa'$, then $\kappa = \kappa'$.*

Proof. By induction on the structure of τ.

- Case $\tau = \xi$:
  1. By inversion of K-Var (1st derivation): $\kappa = \Gamma(\xi)$
  2. By inversion of K-Var (2nd derivation): $\kappa' = \Gamma(\xi)$
  3. By (1) and (2): $\kappa = \kappa'$
- Case $\tau = \tau_1 \to \tau_2$:
  1. By inversion of K-Arrow (1st derivation): $\kappa = \Omega$
  2. By inversion of K-Arrow (2nd derivation): $\kappa' = \Omega$

3. By (1) and (2): $\kappa = \kappa'$

- Case $\tau = \tau_1 \times \tau_2$:
  1. By inversion of K-Times (1st derivation): $\kappa = \Omega$
  2. By inversion of K-Times (2nd derivation): $\kappa' = \Omega$
  3. By (1) and (2): $\kappa = \kappa'$
- Case $\tau = \forall\alpha{:}\kappa_1.\tau'$ (w.l.o.g. $\alpha \notin \mathit{dom}(\Gamma)$):
  1. By inversion of K-Univ (1st derivation): $\kappa = \Omega$
  2. By inversion of K-Univ (2nd derivation): $\kappa' = \Omega$
  3. By (1) and (2): $\kappa = \kappa'$
- Case $\tau = \exists\alpha{:}\kappa_1.\tau'$ (w.l.o.g. $\alpha \notin \mathit{dom}(\Gamma)$):
  1. By inversion of K-Exist (1st derivation): $\kappa = \Omega$
  2. By inversion of K-Exist (2nd derivation): $\kappa' = \Omega$
  3. By (1) and (2): $\kappa = \kappa'$
- Case $\tau = \lambda\alpha{:}\kappa_1.\tau'$ (w.l.o.g. $\alpha \notin \mathit{dom}(\Gamma)$):
  1. By inversion of K-Abs (1st derivation): $\kappa = \kappa_1 \to \kappa_2$ and $\Gamma, \alpha{:}\kappa_1 \vdash \tau' : \kappa_2$
  2. By inversion of K-Abs (2nd derivation): $\kappa' = \kappa_1 \to \kappa_2'$ and $\Gamma, \alpha{:}\kappa_1 \vdash \tau' : \kappa_2'$
  3. By (1), (2), and induction: $\kappa_2 = \kappa_2'$
  4. By (1), (2), and (3): $\kappa = \kappa'$
- Case $\tau = \tau_1\ \tau_2$:
  1. By inversion of K-App (1st derivation): $\Gamma \vdash \tau_1 : \kappa_1 \to \kappa$
  2. By inversion of K-App (2nd derivation): $\Gamma \vdash \tau_1 : \kappa_1' \to \kappa'$
  3. By (1), (2), and induction: $\kappa_1 \to \kappa = \kappa_1' \to \kappa'$ and hence $\kappa = \kappa'$
- Case $\tau = \langle\tau_1, \tau_2\rangle$:
  1. By inversion of K-Pair (1st derivation): $\kappa = \kappa_1 \times \kappa_2$ and $\Gamma \vdash \tau_1 : \kappa_1$ and $\Gamma \vdash \tau_2 : \kappa_2$
  2. By inversion of K-Pair (2nd derivation): $\kappa' = \kappa_1' \times \kappa_2'$ and $\Gamma \vdash \tau_1 : \kappa_1'$ and $\Gamma \vdash \tau_2 : \kappa_2'$
  3. By (1), (2), and induction: $\kappa_1 = \kappa_1'$ and $\kappa_2 = \kappa_2'$
  4. By (1), (2), and (3): $\kappa = \kappa'$
- Case $\tau = \tau'.1$:
  1. By inversion of K-Proj1 (1st derivation): $\Gamma \vdash \tau' : \kappa \times \kappa_2$
  2. By inversion of K-Proj1 (2nd derivation): $\Gamma \vdash \tau' : \kappa' \times \kappa_2'$
  3. By (1), (2), and induction: $\kappa \times \kappa_2 = \kappa' \times \kappa_2'$ and hence $\kappa = \kappa'$
- Case $\tau = \tau'.2$:
  1. By inversion of K-Proj2 (1st derivation): $\Gamma \vdash \tau' : \kappa_1 \times \kappa$
  2. By inversion of K-Proj2 (2nd derivation): $\Gamma \vdash \tau' : \kappa_1' \times \kappa'$
  3. By (1), (2), and induction: $\kappa_1 \times \kappa = \kappa_1' \times \kappa'$ and hence $\kappa = \kappa'$

□

Proposition 12 (Uniqueness of Types)**.** *If* $\Gamma \vdash e : \tau$ *and* $\Gamma' \vdash e : \tau'$ *with* $\Gamma \equiv \Gamma'$*, then* $\tau \equiv \tau'$*.*

Proof. By induction on the structure of e.

- Case $e = x$:
  1. By Typing Inversion (1st derivation): $\tau = \Gamma(x)$
  2. By Typing Inversion (2nd derivation): $\tau' = \Gamma'(x)$
  3. By (1), (2), and assumption: $\tau \equiv \tau'$
- Case $e = \lambda x{:}\tau_1.e'$ (w.l.o.g. $x \notin \mathit{dom}(\Gamma)$):
  1. By Typing Inversion (1st derivation): $\Gamma, x{:}\tau_1 \vdash e' : \tau_2$ and $\tau \equiv \tau_1 \to \tau_2$
  2. By Typing Inversion (2nd derivation): $\Gamma', x{:}\tau_1 \vdash e' : \tau_2'$ and $\tau' \equiv \tau_1 \to \tau_2'$
  3. By assumption: $\Gamma, x{:}\tau_1 \equiv \Gamma', x{:}\tau_1$
  4. By (1), (2), (3), and induction: $\tau_2 \equiv \tau_2'$
  5. By (1), (2), (4), Q-Symm, Q-Refl, Q-Arrow, and Q-Trans: $\tau \equiv \tau'$
- Case $e = e_1\ e_2$:
  1. By Typing Inversion (1st derivation): $\Gamma \vdash e_1 : \tau_2 \to \tau$
  2. By Typing Inversion (2nd derivation): $\Gamma' \vdash e_1 : \tau_2' \to \tau'$
  3. By (1), (2), assumption, and induction: $\tau_2 \to \tau \equiv \tau_2' \to \tau'$
  4. By (4) and Type Equivalence Inversion: $\tau \equiv \tau'$
- Case $e = \langle e_1, e_2\rangle$:
  1. By Typing Inversion (1st derivation): $\Gamma \vdash e_1 : \tau_1$ and $\Gamma \vdash e_2 : \tau_2$ where $\tau \equiv \tau_1 \times \tau_2$
  2. By Typing Inversion (2nd derivation): $\Gamma' \vdash e_1 : \tau_1'$ and $\Gamma \vdash e_2 : \tau_2'$ where $\tau' \equiv \tau_1' \times \tau_2'$
  3. By (1), (2), assumption, and induction: $\tau_1 \equiv \tau_1'$ and $\tau_2 \equiv \tau_2'$
  4. By (3) and Q-Times: $\tau_1 \times \tau_2 \equiv \tau_1' \times \tau_2'$
  5. By (1), (2), (4), Q-Symm, and Q-Trans: $\tau \equiv \tau'$
- Case $e = \lambda\alpha{:}\kappa.e'$ (w.l.o.g. $\alpha \notin \mathit{dom}(\Gamma)$):
  1. By Typing Inversion (1st derivation): $\Gamma, \alpha{:}\kappa \vdash e' : \tau_1$ and $\tau \equiv \forall\alpha{:}\kappa.\tau_1$
  2. By Typing Inversion (2nd derivation): $\Gamma', \alpha{:}\kappa \vdash e' : \tau_2$ and $\tau' \equiv \forall\alpha{:}\kappa.\tau_2$
  3. By assumption: $\Gamma, \alpha{:}\kappa \equiv \Gamma', \alpha{:}\kappa$
  4. By (1), (2), (3), and induction: $\tau_1 \equiv \tau_2$
  5. By (1), (2), (4), Q-Symm, Q-Trans, and Q-Univ: $\tau \equiv \tau'$
- Case $e = e'\ \tau''$:
  1. By Typing Inversion (1st derivation): $\Gamma \vdash e' : \forall\alpha{:}\kappa_1.\tau_1$ and $\Gamma \vdash \tau'' : \kappa_1$ where $\tau \equiv \tau_1[\alpha := \tau'']$
  2. By Typing Inversion (2nd derivation): $\Gamma' \vdash e' : \forall\alpha{:}\kappa_2.\tau_2$ and $\Gamma' \vdash \tau'' : \kappa_2$ where $\tau' \equiv \tau_2[\alpha := \tau'']$
  3. By (1), (2), assumption, and induction: $\forall\alpha{:}\kappa_1.\tau_1 \equiv \forall\alpha{:}\kappa_2.\tau_2$

4. By (3) and Type Equivalence Inversion: $\kappa_1 = \kappa_2$ and $\tau_1 \equiv \tau_2$
5. By (4) and Type Substitution: $\tau_1[\alpha := \tau''] \equiv \tau_2[\alpha := \tau'']$
6. By (1), (2), Q-Trans, and Q-Symm: $\tau \equiv \tau'$

- Case $e = \langle \tau_1, e' \rangle{:}\tau_2$:
  1. By Typing Inversion (1st derivation): $\tau \equiv \tau_2$
  2. By Typing Inversion (2nd derivation): $\tau' \equiv \tau_2$
  3. By (1), (2), Q-Symm, and Q-Trans: $\tau \equiv \tau'$

- Case $e = \mathsf{let}\ \langle x_1, x_2 \rangle\ \texttt{=}\ e_1\ \mathsf{in}\ e_2$ (w.l.o.g. $\{x_1, x_2\} \not\subseteq dom(\Gamma)$):
  1. By Typing Inversion (1st derivation): $\Gamma \vdash e_1 : \tau_1 \times \tau_2$ and $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_2 : \tau$
  2. By Typing Inversion (2nd derivation): $\Gamma' \vdash e_1 : \tau_1' \times \tau_2'$ and $\Gamma, x_1{:}\tau_1', x_2{:}\tau_2' \vdash e_2 : \tau'$
  3. By (1), (2), assumption, and induction: $\tau_1 \times \tau_2 \equiv \tau_1' \times \tau_2'$
  4. By (3) and Type Equivalence Inversion: $\tau_1 \equiv \tau_1'$ and $\tau_2 \equiv \tau_2'$
  5. By (4) and assumption: $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \equiv \Gamma', x_1{:}\tau_1', x_2{:}\tau_2'$
  6. By (1), (2), (5), and induction: $\tau \equiv \tau'$

- Case $e = \mathsf{let}\ \langle \alpha, x \rangle\ \texttt{=}\ e_1\ \mathsf{in}\ e_2$ (w.l.o.g. $\{\alpha, x\} \not\subseteq dom(\Gamma)$):
  1. By Typing Inversion (1st derivation): $\Gamma \vdash e_1 : \exists\alpha{:}\kappa_1.\tau_1$ and $\Gamma, \alpha{:}\kappa_1, x{:}\tau_1 \vdash e_2 : \tau$
  2. By Typing Inversion (2nd derivation): $\Gamma' \vdash e_1 : \exists\alpha{:}\kappa_2.\tau_2$ and $\Gamma', \alpha{:}\kappa_2, x{:}\tau_2 \vdash e_2 : \tau'$
  3. By (1), (2), assumption, and induction: $\exists\alpha{:}\kappa_1.\tau_1 \equiv \exists\alpha{:}\kappa_2.\tau_2$
  4. By (3) and Type Equivalence Inversion: $\kappa_1 = \kappa_2$ and $\tau_1 \equiv \tau_2$
  5. By (4) and assumption: $\Gamma, \alpha{:}\kappa_1, x{:}\tau_1 \equiv \Gamma', \alpha{:}\kappa_2, x{:}\tau_2$
  6. By (1), (2), (5), and induction: $\tau \equiv \tau'$

- Case $e = \mathsf{lazy}\ \langle \zeta, x \rangle\ \texttt{=}\ e_1\ \mathsf{in}\ e_2$ (w.l.o.g. $\{\zeta, x\} \not\subseteq dom(\Gamma)$):
  1. By Typing Inversion (1st derivation): $\Gamma \vdash e_1 : \exists\alpha{:}\kappa_1.\tau_1$ and $\Gamma, \zeta{:}\kappa_1, x{:}\tau_1[\alpha := \zeta] \vdash e_2 : \tau$
  2. By Typing Inversion (2nd derivation): $\Gamma' \vdash e_1 : \exists\alpha{:}\kappa_2.\tau_2$ and $\Gamma', \zeta{:}\kappa_2, x{:}\tau_2[\alpha := \zeta] \vdash e_2 : \tau'$
  3. By (1), (2), assumption, and induction: $\exists\alpha{:}\kappa_1.\tau_1 \equiv \exists\alpha{:}\kappa_2.\tau_2$
  4. By (3) and Type Equivalence Inversion: $\kappa_1 = \kappa_2$ and $\tau_1 \equiv \tau_2$
  5. By (4) and Type Substitution: $\tau_1[\alpha := \zeta] \equiv \tau_2[\alpha := \zeta]$
  6. By (4), (5), and assumption: $\Gamma, \zeta{:}\kappa_1, x{:}\tau_1[\alpha := \zeta] \equiv \Gamma', \zeta{:}\kappa_2, x{:}\tau_2[\alpha := \zeta]$
  7. By (1), (2), (6), and induction: $\tau \equiv \tau'$

- Case $e = \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$:
  1. By Typing Inversion (1st derivation): $\Gamma \vdash e_2 : \tau$
  2. By Typing Inversion (2nd derivation): $\Gamma' \vdash e_2 : \tau'$
  3. By (1), (2), assumption, and induction: $\tau \equiv \tau'$

□

Lemma 6 (Equivalent Environments)**.**

1. If $\Gamma \vdash \tau : \kappa$ *and* $\Gamma \equiv \Gamma'$ *where* $\Gamma' \vdash \Box$*, then* $\Gamma' \vdash \tau : \kappa$*.*

2. If $\Gamma \vdash e : \tau$ *and* $\Gamma \equiv \Gamma'$ *where* $\Gamma' \vdash \Box$*, then* $\Gamma' \vdash e : \tau$*.*

Proof. (1) by induction on the kinding derivation. (2) by induction on the typing derivation.

1. Let $\Gamma \vdash \tau : \kappa$ and $\Gamma \equiv \Gamma'$ and $\Gamma' \vdash \Box$.
   - Case K-Var: $\tau = \xi$
     (a) By inversion: $\Gamma \vdash \Box$ and $\Gamma(\xi) = \kappa$
     (b) By (a) and assumption: $\kappa = \Gamma'(\xi)$
     (c) By (b), assumption, and K-Var: $\Gamma' \vdash \xi : \kappa$
   - Case K-Arrow: $\tau = \tau_1 \rightarrow \tau_2$ and $\kappa = \Omega$
     (a) By inversion: $\Gamma \vdash \tau_1 : \Omega$ and $\Gamma \vdash \tau_2 : \Omega$
     (b) By (a), assumption, and induction: $\Gamma' \vdash \tau_1 : \Omega$ and $\Gamma' \vdash \tau_2 : \Omega$
     (c) By (b) and K-Arrow: $\Gamma' \vdash \tau_1 \rightarrow \tau_2 : \Omega$
   - Case K-Times: $\tau = \tau_1 \times \tau_2$ and $\kappa = \Omega$
     (a) By inversion: $\Gamma \vdash \tau_1 : \Omega$ and $\Gamma \vdash \tau_2 : \Omega$
     (b) By (a), assumption, and induction: $\Gamma' \vdash \tau_1 : \Omega$ and $\Gamma' \vdash \tau_2 : \Omega$
     (c) By (b) and K-Times: $\Gamma' \vdash \tau_1 \times \tau_2 : \Omega$
   - Case K-Univ: $\tau = \forall\alpha{:}\kappa'.\tau'$ and $\kappa = \Omega$
     (a) By inversion: $\Gamma, \alpha{:}\kappa' \vdash \tau' : \Omega$
     (b) By (a) and Environment Validity: $\Gamma, \alpha{:}\kappa' \vdash \Box$
     (c) By (b) and inversion of E-Type: $\alpha \notin \mathit{dom}(\Gamma)$
     (d) By (c) and assumption: $\alpha \notin \mathit{dom}(\Gamma')$
     (e) By (c), (d), and assumption: $\Gamma, \alpha{:}\kappa' \equiv \Gamma', \alpha{:}\kappa'$
     (f) By (d), assumption, and E-Type: $\Gamma', \alpha{:}\kappa' \vdash \Box$
     (g) By (a), (e), (f), and induction: $\Gamma', \alpha{:}\kappa' \vdash \tau' : \Omega$
     (h) By (g) and K-Univ: $\Gamma' \vdash \exists\alpha{:}\kappa'.\tau' : \Omega$
   - Case K-Exist: $\tau = \exists\alpha{:}\kappa'.\tau'$ and $\kappa = \Omega$ where $\alpha \notin \mathit{dom}(\Gamma)$
     (a) By inversion: $\Gamma, \alpha{:}\kappa' \vdash \tau' : \Omega$
     (b) By (a) and Environment Validity: $\Gamma, \alpha{:}\kappa' \vdash \Box$
     (c) By (b) and inversion of E-Type: $\alpha \notin \mathit{dom}(\Gamma)$
     (d) By (c) and assumption: $\alpha \notin \mathit{dom}(\Gamma')$
     (e) By (c), (d), and assumption: $\Gamma, \alpha{:}\kappa' \equiv \Gamma', \alpha{:}\kappa'$
     (f) By (d), assumption, and E-Type: $\Gamma', \alpha{:}\kappa' \vdash \Box$
     (g) By (a), (e), (f), and induction: $\Gamma', \alpha{:}\kappa' \vdash \tau' : \Omega$
     (h) By (g) and K-Exist: $\Gamma' \vdash \exists\alpha{:}\kappa'.\tau' : \Omega$
   - Case K-Abs: $\tau = \lambda\alpha{:}\kappa'.\tau'$ and $\kappa = \kappa' \rightarrow \kappa''$
     (a) By inversion: $\Gamma, \alpha{:}\kappa' \vdash \tau' : \kappa''$
     (b) By (a) and Environment Validity: $\Gamma, \alpha{:}\kappa' \vdash \Box$
     (c) By (b) and inversion of E-Type: $\alpha \notin \mathit{dom}(\Gamma)$
     (d) By (c) and assumption: $\alpha \notin \mathit{dom}(\Gamma')$
     (e) By (c), (d), and assumption: $\Gamma, \alpha{:}\kappa' \equiv \Gamma', \alpha{:}\kappa'$
     (f) By (d), assumption, and E-Type: $\Gamma', \alpha{:}\kappa' \vdash \Box$

(g) By (a), (e), (f), and induction: $\Gamma', \alpha{:}\kappa' \vdash \tau' : \kappa''$
(h) By (g) and K-Abs: $\Gamma' \vdash \lambda\alpha{:}\kappa'.\tau' : \kappa' \to \kappa''$

- Case K-App: $\tau = \tau_1\ \tau_2$
  (a) By inversion: $\Gamma \vdash \tau_1 : \kappa' \to \kappa$ and $\Gamma \vdash \tau_2 : \kappa'$
  (b) By (a), assumption, and induction: $\Gamma' \vdash \tau_1 : \kappa' \to \kappa$ and $\Gamma' \vdash \tau_2 : \kappa'$
  (c) By (b) and K-App: $\Gamma' \vdash \tau_1\ \tau_2 : \kappa$
- Case K-Pair: $\tau = \langle \tau_1, \tau_2 \rangle$ and $\kappa = \kappa_1 \times \kappa_2$
  (a) By inversion: $\Gamma \vdash \tau_1 : \kappa_1$ and $\Gamma \vdash \tau_2 : \kappa_2$
  (b) By (a), assumption, and induction: $\Gamma' \vdash \tau_1 : \kappa_1$ and $\Gamma' \vdash \tau_2 : \kappa_2$
  (c) By (b) and K-Pair: $\Gamma' \vdash \langle \tau_1, \tau_2 \rangle : \kappa_1 \times \kappa_2$
- Case K-Proj1: $\tau = \tau'.1$
  (a) By inversion: $\Gamma \vdash \tau' : \kappa \times \kappa_2$
  (b) By (a), assumption, and induction: $\Gamma' \vdash \tau' : \kappa \times \kappa_2$
  (c) By (b) and K-Proj1: $\Gamma' \vdash \tau'.1 : \kappa$
- Case K-Proj2: $\tau = \tau'.2$
  (a) By inversion: $\Gamma \vdash \tau' : \kappa_1 \times \kappa$
  (b) By (a), assumption, and induction: $\Gamma' \vdash \tau' : \kappa_1 \times \kappa$
  (c) By (b) and K-Proj2: $\Gamma' \vdash \tau'.2 : \kappa$

2. Let $\Gamma \vdash e : \tau$ and $\Gamma \equiv \Gamma'$ where $\Gamma' \vdash \Box$.

- Case T-Equiv:
  (a) By inversion: $\Gamma \vdash e : \tau'$ and $\tau' \equiv \tau$ where $\Gamma \vdash \tau : \Omega$
  (b) By (a), assumption, and induction: $\Gamma' \vdash e : \tau'$
  (c) By (a), assumption, and (1): $\Gamma' \vdash \tau : \Omega$
  (d) By (a), (b), (c), and T-Equiv: $\Gamma' \vdash e : \tau$
- Case T-Var: $e = x$
  (a) By inversion: $\Gamma \vdash \Box$ and $\tau = \Gamma(x)$
  (b) By assumption: $\Gamma' = \Gamma_1, x{:}\tau', \Gamma_2$ and $\tau' \equiv \tau$
  (c) By (b) and assumption: $\Gamma' \vdash x : \tau'$
  (d) By assumption and Validity: $\Gamma \vdash \tau : \Omega$
  (e) By (d), assumption, and (1): $\Gamma' \vdash \tau : \Omega$
  (f) By (c), (b), (e), and T-Equiv: $\Gamma' \vdash x : \tau$
- Case T-Abs: $e = \lambda x{:}\tau_1.e_1$ and $\tau = \tau_1 \to \tau_2$
  (a) By inversion: $\Gamma, x{:}\tau_1 \vdash e_1 : \tau_2$
  (b) By (a) and Environment Validity: $\Gamma, x{:}\tau_1 \vdash \Box$
  (c) By (b) and inversion of E-Term: $\Gamma \vdash \tau_1 : \Omega$ and $x \notin \mathit{dom}(\Gamma)$
  (d) By (c) and assumption: $x \notin \mathit{dom}(\Gamma')$
  (e) By (c), assumption, and (1): $\Gamma' \vdash \tau_1 : \Omega$
  (f) By (d), (e), and E-Term: $\Gamma', x{:}\tau_1 \vdash \Box$
  (g) By (c), (d), and assumption: $\Gamma, x{:}\tau_1 \equiv \Gamma', x{:}\tau_1$
  (h) By (f), (a), (g), and induction: $\Gamma', x{:}\tau_1 \vdash e_1 : \tau_2$
  (i) By (h) and T-Abs: $\Gamma' \vdash \lambda x{:}\tau_1.e_1 : \tau_1 \to \tau_2$
- Case T-App: $e = e_1\ e_2$

(a) By inversion: $\Gamma \vdash e_1 : \tau' \to \tau$ and $\Gamma \vdash e_2 : \tau'$
(b) By (a), assumption, and induction: $\Gamma' \vdash e_1 : \tau' \to \tau$ and $\Gamma' \vdash e_2 : \tau'$
(c) By (b) and T-App: $\Gamma' \vdash e_1\ e_2 : \tau$

- Case T-Gen: $e = \lambda\alpha{:}\kappa.e_1$ and $\tau = \forall\alpha{:}\kappa.\tau'$

(a) By inversion: $\Gamma, \alpha{:}\kappa \vdash e_1 : \tau'$
(b) By (a) and Environment Validity: $\Gamma, \alpha{:}\kappa \vdash \Box$
(c) By (b) and inversion of E-Type: $\alpha \notin \mathit{dom}(\Gamma)$
(d) By (c) and assumption: $\alpha \notin \mathit{dom}(\Gamma')$
(e) By (d) and assumption: $\Gamma', \alpha{:}\kappa \vdash \Box$
(f) By (c), (d), and assumption: $\Gamma, \alpha{:}\kappa \equiv \Gamma', \alpha{:}\kappa$
(g) By (e), (b), (f), and induction: $\Gamma', \alpha{:}\kappa \vdash e_1 : \tau'$
(h) By (g) and T-Gen: $\Gamma' \vdash \lambda\alpha{:}\kappa.e_1 : \forall\alpha{:}\kappa.\tau'$

- Case T-Inst: $e = e_1\ \tau_2$ and $\tau = \tau'[\alpha := \tau_2]$

(a) By inversion: $\Gamma \vdash e_1 : \forall\alpha{:}\kappa.\tau'$ and $\Gamma \vdash \tau_2 : \kappa$
(b) By (a), assumption, and induction: $\Gamma' \vdash e_1 : \forall\alpha{:}\kappa.\tau'$
(c) By (a), assumption, and (1): $\Gamma' \vdash \tau_2 : \kappa$
(d) By (b), (c), and T-Inst: $\Gamma' \vdash e_1\ \tau_2 : \tau'[\alpha := \tau_2]$

- Case T-Pair: $e = \langle e_1, e_2 \rangle$ and $\tau = \tau_1 \times \tau_2$

(a) By inversion: $\Gamma \vdash e_1 : \tau_1$ and $\Gamma \vdash e_2 : \tau_2$
(b) By (a), assumption, and induction: $\Gamma' \vdash e_1 : \tau_1$ and $\Gamma' \vdash e_2 : \tau_2$
(c) By (b) and T-Pair: $\Gamma' \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2$

- Case T-Proj: $e = \mathsf{let}\ \langle x_1, x_2 \rangle = e_1\ \mathsf{in}\ e_2$

(a) By inversion: $\Gamma \vdash e_1 : \tau_1 \times \tau_2$ and $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_2 : \tau$
(b) By (a) and Environment Validity: $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash \Box$
(c) By (b) and inversion of E-Term: $\Gamma, x_1{:}\tau_1 \vdash \tau_2 : \Omega$ and $x_2 \notin \mathit{dom}(\Gamma, x_1{:}\tau_1)$
(d) By (c) and Environment Validity: $\Gamma, x_1{:}\tau_1 \vdash \Box$
(e) By (d) and inversion of E-Term: $\Gamma \vdash \tau_1 : \Omega$ and $x_1 \notin \mathit{dom}(\Gamma)$
(f) By (e) and assumption: $x_1 \notin \mathit{dom}(\Gamma')$
(g) By (e), assumption, and (1): $\Gamma' \vdash \tau_1 : \Omega$
(h) By (f), (g), and E-Term: $\Gamma', x_1{:}\tau_1 \vdash \Box$
(i) By (h), (c), and induction: $\Gamma', x_1{:}\tau_1 \vdash \tau_2 : \Omega$
(j) By (c) and assumption: $x_2 \notin \mathit{dom}(\Gamma', x_1{:}\tau_1)$
(k) By (i), (j), and E-Term: $\Gamma', x_1{:}\tau_2, x_2{:}\tau_2 \vdash \Box$
(l) By (c), (e), (f), (j), and assumption: $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \equiv \Gamma', x_1{:}\tau_1, x_2{:}\tau_2$
(m) By (k), (a), (l), and induction: $\Gamma', x_1{:}\tau_2, x_2{:}\tau_2 \vdash e_2 : \tau$
(n) By (a), assumption, and induction: $\Gamma' \vdash e_1 : \tau_1 \times \tau_2$
(o) By (m), (n), and T-Proj: $\Gamma' \vdash \mathsf{let}\ \langle x_1, x_2 \rangle = e_1\ \mathsf{in}\ e_2 : \tau$

- Case T-Close: $e = \langle \tau_1, e' \rangle{:}\tau$ where $\tau = \exists\alpha{:}\kappa.\tau_2$

(a) By inversion: $\Gamma \vdash \tau_1 : \kappa$ and $\Gamma \vdash e' : \tau_2[\alpha := \tau_1]$ where $\Gamma \vdash \exists\alpha{:}\kappa.\tau_2 : \Omega$
(b) By (a), assumption, and induction: $\Gamma' \vdash e' : \tau_2[\alpha := \tau_1]$
(c) By (a), assumption, and (1): $\Gamma' \vdash \tau_1 : \kappa$ and $\Gamma' \vdash \exists\alpha{:}\kappa.\tau_2 : \Omega$
(d) By (b), (c), and T-Close: $\Gamma' \vdash \langle \tau_1, e' \rangle{:}\exists\alpha{:}\kappa.\tau_2 : \exists\alpha{:}\kappa.\tau_2$

- Case T-Open: $e = \mathsf{let}\ \langle \alpha, x \rangle = e_1\ \mathsf{in}\ e_2$

(a) By inversion: $\Gamma \vdash e_1 : \exists\alpha{:}\kappa.\tau'$ and $\Gamma, \alpha{:}\kappa, x{:}\tau' \vdash e_2 : \tau$ where $\alpha \notin ftv(\tau)$
(b) By (a) and Environment Validity: $\Gamma, \alpha{:}\kappa, x{:}\tau' \vdash \Box$
(c) By (b) and inversion of E-Term: $\Gamma, \alpha{:}\kappa \vdash \tau' : \Omega$ and $x \notin dom(\Gamma, \alpha{:}\kappa)$
(d) By (c) and Environment Validity: $\Gamma, \alpha{:}\kappa \vdash \Box$
(e) By (d) and inversion of E-Type: $\alpha \notin dom(\Gamma)$
(f) By (e) and assumption: $\alpha \notin dom(\Gamma')$
(g) By (f), assumption, and E-Type: $\Gamma', \alpha{:}\kappa \vdash \Box$
(h) By (g), (c), and induction: $\Gamma', \alpha{:}\kappa \vdash \tau' : \Omega$
(i) By (c) and assumption: $x \notin dom(\Gamma', \alpha{:}\kappa)$
(j) By (h), (i), and E-Term: $\Gamma', \alpha{:}\kappa, x{:}\tau' \vdash \Box$
(k) By (c), (e), (f), (i), and assumption: $\Gamma, \alpha{:}\kappa, x{:}\tau' \equiv \Gamma', \alpha{:}\kappa, x{:}\tau'$
(l) By (j), (a), assumption, and induction: $\Gamma', \alpha{:}\kappa, x{:}\tau' \vdash e_2 : \tau$
(m) By (a), (k), and induction: $\Gamma' \vdash e_1 : \exists\alpha{:}\kappa.\tau'$
(n) By (l), (m), (a), and T-Proj: $\Gamma' \vdash \mathsf{let}\ \langle\alpha, x\rangle = e_1\ \mathsf{in}\ e_2 : \tau$

- Case T-Lazy: $e = \mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ e_2$

(a) By inversion: $\Gamma \vdash e_1 : \exists\alpha{:}\kappa.\tau'$ and $\Gamma, \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta] \vdash e_2 : \tau$ where $\zeta \notin ftv(\tau)$
(b) By (a) and Environment Validity: $\Gamma, \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta] \vdash \Box$
(c) By (b) and inversion of E-Term: $\Gamma, \zeta{:}\kappa \vdash \tau'[\alpha := \zeta] : \Omega$ and $x \notin dom(\Gamma, \zeta{:}\kappa)$
(d) By (c) and Environment Validity: $\Gamma, \zeta{:}\kappa \vdash \Box$
(e) By (d) and inversion of E-Type: $\zeta \notin dom(\Gamma)$
(f) By (e) and assumption: $\zeta \notin dom(\Gamma')$
(g) By (f), assumption, and E-Type: $\Gamma', \zeta{:}\kappa \vdash \Box$
(h) By (g), (c), and induction: $\Gamma', \zeta{:}\kappa \vdash \tau'[\alpha := \zeta] : \Omega$
(i) By (c) and assumption: $x \notin dom(\Gamma', \alpha{:}\kappa)$
(j) By (h), (i), and E-Term: $\Gamma', \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta] \vdash \Box$
(k) By (c), (e), (f), (i), and assumption: $\Gamma, \alpha{:}\kappa, x{:}\tau' \equiv \Gamma', \alpha{:}\kappa, x{:}\tau'$
(l) By (j), (a), assumption, and induction: $\Gamma', \zeta{:}\kappa, x{:}\tau'[\alpha := \zeta] \vdash e_2 : \tau$
(m) By (a), (k), and induction: $\Gamma' \vdash e_1 : \exists\alpha{:}\kappa.\tau'$
(n) By (l), (m), (a), and T-Proj: $\Gamma' \vdash \mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ e_2 : \tau$

- Case T-Case: $e = \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$

(a) By inversion:
   i. $\Gamma \vdash e_0 : \tau_0$
   ii. $\Gamma, x{:}\tau_0' \vdash e_1 : \tau$
   iii. $\Gamma \vdash e_2 : \tau$
(b) By (a-ii) and Environment Validity: $\Gamma, x{:}\tau_0' \vdash \Box$
(c) By (b) and inversion of E-Term: $\Gamma \vdash \tau_0' : \Omega$ and $x \notin dom(\Gamma)$
(d) By (c) and assumption: $x \notin dom(\Gamma')$
(e) By (c), assumption, and (1): $\Gamma' \vdash \tau_0' : \Omega$
(f) By (d), (e), and E-Term: $\Gamma', x{:}\tau_0' \vdash \Box$
(g) By (c), (d), and assumption: $\Gamma, x{:}\tau_0' \equiv \Gamma', x{:}\tau_0'$
(h) By (f), (a-ii), (g), and induction: $\Gamma', x{:}\tau_0' \vdash e_1 : \tau$
(i) By (a-i), assumption, and induction: $\Gamma' \vdash e_0 : \tau_0$
(j) By (a-iii), assumption, and induction: $\Gamma' \vdash e_2 : \tau$
(k) By (h), (i), (j), and T-Case: $\Gamma' \vdash \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$

□

Lemma 7 (Context Elimination)**.**

1. *If* $\Gamma \vdash E[e] : \tau$*, then* $\Gamma \vdash e : \tau'$*.*
2. *If* $\Gamma \vdash LE[e] : \tau$*, then* $\Gamma \vdash L[e] : \tau'$*.*
3. *If* $\Gamma \vdash LE[e] : \tau$*, then* $\Gamma, \Gamma' \vdash e : \tau'$ *with* $\Gamma \vdash L : \Gamma'$*.*

Proof. (1) by structural induction on E. (2) and (3) by structural induction on L.

1. Let $\Gamma \vdash E[e] : \tau$.
   - Case $E = \_$:
     (a) $E[e] = \_[e] = e$
     (b) By (a) and assumption: $\Gamma \vdash e : \tau$
   - Case $E = E'\ e'$:
     (a) $E[e] = (E'\ e')[e] = E'[e]\ e'$
     (b) By (a) and assumption: $\Gamma \vdash E'[e]\ e' : \tau$
     (c) By (b) and Typing Inversion: $\Gamma \vdash E'[e] : \tau' \to \tau$
     (d) By (c) and induction: $\Gamma \vdash e : \tau''$
   - Case $E = (\lambda x{:}\tau'.e')\ E'$:
     (a) $E[e] = ((\lambda x{:}\tau'.e')\ E')[e] = (\lambda x{:}\tau'.e')\ E'[e]$
     (b) By (a) and assumption: $\Gamma \vdash (\lambda x{:}\tau'.e')\ E'[e] : \tau$
     (c) By (b) and Typing Inversion: $\Gamma \vdash E'[e] : \tau''$
     (d) By (c) and induction: $\Gamma \vdash e : \tau'''$
   - Case $E = \langle E', e'\rangle$:
     (a) $E[e] = \langle E', e'\rangle[e] = \langle E'[e], e'\rangle$
     (b) By (a) and assumption: $\Gamma \vdash \langle E'[e], e'\rangle : \tau$
     (c) By (b) and Typing Inversion: $\Gamma \vdash E'[e] : \tau_1$
     (d) By (c) and induction: $\Gamma \vdash e : \tau'$
   - Case $E = \langle v, E'\rangle$:
     (a) $E[e] = \langle v, E'\rangle[e] = \langle v, E'[e]\rangle$
     (b) By (a) and assumption: $\Gamma \vdash \langle v, E'[e]\rangle : \tau$
     (c) By (b) and Typing Inversion: $\Gamma \vdash E'[e] : \tau_2$
     (d) By (c) and induction: $\Gamma \vdash e : \tau'$
   - Case $E = \langle \tau_1, E'\rangle{:}\tau'$:
     (a) $E[e] = (\langle \tau_1, E'\rangle{:}\tau')[e] = \langle \tau_1, E'[e]\rangle{:}\tau'$
     (b) By (a) and assumption: $\Gamma \vdash \langle \tau_1, E'[e]\rangle{:}\tau' : \tau$
     (c) By (b) and Typing Inversion: $\Gamma \vdash E'[e] : \tau_2[\alpha := \tau_1]$
     (d) By (c) and induction: $\Gamma \vdash e : \tau''$
   - Case $E = \mathsf{let}\ \langle x_1, x_2\rangle = E'\ \mathsf{in}\ e'$ (w.l.o.g. $\{x_1, x_2\} \cap dom(\Gamma) = \emptyset$):
     (a) $E[e] = (\mathsf{let}\ \langle x_1, x_2\rangle = E'\ \mathsf{in}\ e')[e] = (\mathsf{let}\ \langle x_1, x_2\rangle = E'[e]\ \mathsf{in}\ e')$
     (b) By (a) and assumption: $\Gamma \vdash \mathsf{let}\ \langle x_1, x_2\rangle = E'[e]\ \mathsf{in}\ e' : \tau$
     (c) By (b) and Typing Inversion: $\Gamma \vdash E'[e] : \tau_1 \times \tau_2$
     (d) By (c) and induction: $\Gamma \vdash e : \tau'$

- Case $E = \mathsf{let}\ \langle\alpha, x\rangle = E'\ \mathsf{in}\ e'$ (w.l.o.g. $\{\alpha, x\} \cap dom(\Gamma) = \emptyset$):
  (a) $E[e] = (\mathsf{let}\ \langle\alpha, x\rangle = E'\ \mathsf{in}\ e')[e] = (\mathsf{let}\ \langle\alpha, x\rangle = E'[e]\ \mathsf{in}\ e')$
  (b) By (a) and assumption: $\Gamma \vdash \mathsf{let}\ \langle\alpha, x\rangle = E'[e]\ \mathsf{in}\ e' : \tau$
  (c) By (b) and Typing Inversion: $\Gamma \vdash E'[e] : \exists\alpha{:}\kappa.\tau'$
  (d) By (c) and induction: $\Gamma \vdash e : \tau''$
- Case $E = E'\ \tau'$:
  (a) $E[e] = (E'\ \tau')[e] = E'[e]\ \tau'$
  (b) By (a) and assumption: $\Gamma \vdash E'[e]\ \tau' : \tau$
  (c) By (b) and Typing Inversion: $\Gamma \vdash E'[e] : \forall\alpha{:}\kappa.\tau_1$
  (d) By (c) and induction: $\Gamma \vdash e : \tau_2$
- Case $E = \mathsf{tcase}\ E'{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$ (w.l.o.g. $x \notin dom(\Gamma)$):
  (a) By (a) and assumption: $\Gamma \vdash \mathsf{tcase}\ E'[e]{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau$
  (b) By (b) and Typing Inversion: $\Gamma \vdash E'[e] : \tau_0$
  (c) By (c) and induction: $\Gamma \vdash e : \tau'$

2. Let $\Gamma \vdash LE[e] : \tau$.

- Case $L = \_$:
  (a) By assumption: $\Gamma \vdash E[e] : \tau$
  (b) By (a) and (1): $\Gamma \vdash e : \tau'$
  (c) By (b): $\Gamma \vdash L[e] : \tau'$
- Case $L = \mathsf{lazy}\ \langle\zeta, x\rangle = e'\ \mathsf{in}\ L'$:
  (a) By assumption: $\Gamma \vdash \mathsf{lazy}\ \langle\zeta, x\rangle = e'\ \mathsf{in}\ L'E[e] : \tau$
  (b) By (a) and Typing Inversion: $\Gamma \vdash e' : \exists\alpha{:}\kappa.\tau_1$ and $\Gamma, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash L'E[e] : \tau''$ where $\tau'' \equiv \tau$
  (c) By (b) and induction: $\Gamma, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash L'[e] : \tau'$
  (d) By (b), (c), and T-Lazy: $\Gamma \vdash \mathsf{lazy}\ \langle\zeta, x\rangle = e'\ \mathsf{in}\ L'[e] : \tau'$
  (e) By (d): $\Gamma \vdash L[e] : \tau'$

3. Let $\Gamma \vdash LE[e] : \tau$.

- Case $L = \_$:
  (a) By assumption: $\Gamma \vdash E[e] : \tau$
  (b) By (a) and (1): $\Gamma \vdash e : \tau'$
  (c) By assumption and Environment Validity: $\Gamma \vdash \Box$
  (d) Let $\Gamma' = \cdot$.
  (e) By (c), (d), and L-Empty: $\Gamma, \Gamma' = \Gamma$ and $\Gamma \vdash L : \Gamma'$
  (f) By (b) and (e): $\Gamma, \Gamma' \vdash e : \tau'$ and $\Gamma \vdash L : \Gamma'$
- Case $L = \mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ L'$:
  (a) By assumption: $\Gamma \vdash \mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ L'E[e] : \tau$
  (b) By (a) and Typing Inversion: $\Gamma \vdash e_1 : \exists\alpha{:}\kappa.\tau_1$ and $\Gamma, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash L'E[e] : \tau''$ where $\tau'' \equiv \tau$
  (c) By (b) and induction: $\Gamma, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta], \Gamma'' \vdash e : \tau'$ where $\Gamma, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash L' : \Gamma''$
  (d) By (b), (c), and L-Lazy: $\Gamma, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta], \Gamma'' \vdash e : \tau'$ and $\Gamma \vdash L : \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta], \Gamma''$

□

Lemma 8 (Exchange)**.**

1. *If* $\Gamma \vdash E[e] : \tau$ *and* $\Gamma \vdash e : \tau'$ *and* $\Gamma, \Gamma' \vdash e' : \tau'$*, then* $\Gamma, \Gamma' \vdash E[e'] : \tau$*.*
2. *If* $\Gamma \vdash LE[e] : \tau$ *and* $\Gamma, \Gamma' \vdash e : \tau'$ *as well as* $\Gamma, \Gamma' \vdash e' : \tau'$ *and* $\Gamma \vdash L : \Gamma'$*, then* $\Gamma \vdash LE[e'] : \tau$*.*

Proof. (1) by structural induction on E, (2) by structural induction on L.

1. Let $\Gamma \vdash E[e] : \tau$, $\Gamma \vdash e : \tau'$, and $\Gamma, \Gamma' \vdash e' : \tau'$.
   - Case $E = \_$:
     (a) By assumption: $\Gamma \vdash e : \tau$
     (b) By (a), assumption and Uniqueness of Types: $\tau \equiv \tau'$
     (c) By assumption and Environment Validity: $\Gamma, \Gamma' \vdash \Box$
     (d) By assumption and Validity: $\Gamma \vdash \tau : \Omega$
     (e) By (c), (d), and Weakening: $\Gamma, \Gamma' \vdash \tau : \Omega$
     (f) By (b), (e), assumption, and T-Equiv: $\Gamma, \Gamma' \vdash e' : \tau$
     (g) By (f): $\Gamma, \Gamma' \vdash E[e'] : \tau$
   - Case $E = E'\ e''$:
     (a) By assumption: $\Gamma \vdash E'[e]\ e'' : \tau$
     (b) By (a) and Typing Inversion: $\Gamma \vdash E'[e] : \tau'' \to \tau$ and $\Gamma \vdash e'' : \tau''$
     (c) By (b), assumption, and induction: $\Gamma, \Gamma' \vdash E'[e'] : \tau'' \to \tau$
     (d) By assumption and Environment Validity: $\Gamma, \Gamma' \vdash \Box$
     (e) By (b), (d), and Weakening: $\Gamma, \Gamma' \vdash e'' : \tau''$
     (f) By (c), (e), and T-App: $\Gamma, \Gamma' \vdash E'[e']\ e'' : \tau$
     (g) By (f): $\Gamma, \Gamma' \vdash E[e'] : \tau$
   - Case $E = (\lambda x{:}\tau_1.e_1)\ E'$:
     (a) By assumption: $\Gamma \vdash (\lambda x{:}\tau_1.e_1)\ E'[e] : \tau$
     (b) By (a) and Typing Inversion: $\Gamma \vdash \lambda x{:}\tau_1.e_1 : \tau'' \to \tau$ and $\Gamma \vdash E'[e] : \tau''$
     (c) By (b), assumption, and induction: $\Gamma, \Gamma' \vdash E'[e'] : \tau''$
     (d) By assumption and Environment Validity: $\Gamma, \Gamma' \vdash \Box$
     (e) By (b), (d), and Weakening: $\Gamma, \Gamma' \vdash \lambda x{:}\tau_1.e_1 : \tau'' \to \tau$
     (f) By (c), (e), and T-App: $\Gamma, \Gamma' \vdash (\lambda x{:}\tau_1.e_1)\ E'[e'] : \tau$
     (g) By (f): $\Gamma, \Gamma' \vdash E[e'] : \tau$
   - Case $E = \langle E', e'' \rangle$:
     (a) By assumption: $\Gamma \vdash \langle E'[e], e'' \rangle : \tau$
     (b) By (a) and Typing Inversion: $\Gamma \vdash E'[e] : \tau_1$ and $\Gamma \vdash e'' : \tau_2$ where $\tau \equiv \tau_1 \times \tau_2$
     (c) By (b), assumption, and induction: $\Gamma, \Gamma' \vdash E'[e'] : \tau_1$
     (d) By assumption and Environment Validity: $\Gamma, \Gamma' \vdash \Box$
     (e) By (b), (d), and Weakening: $\Gamma, \Gamma' \vdash e'' : \tau_2$
     (f) By (c), (e), and T-Pair: $\Gamma, \Gamma' \vdash \langle E'[e'], e'' \rangle : \tau_1 \times \tau_2$
     (g) By assumption and Validity: $\Gamma \vdash \tau : \Omega$
     (h) By (d), (g), and Weakening: $\Gamma, \Gamma' \vdash \tau : \Omega$
     (i) By (b), (f), (h), and T-Equiv: $\Gamma, \Gamma' \vdash \langle E'[e'], e'' \rangle : \tau$

(j) By (i): $\Gamma, \Gamma' \vdash E[e'] : \tau$

- Case $E = \langle v, E' \rangle$:

  (a) By assumption: $\Gamma \vdash \langle v, E'[e] \rangle : \tau$
  (b) By (a) and Typing Inversion: $\Gamma \vdash v : \tau_1$ and $\Gamma \vdash E'[e] : \tau_2$ where $\tau \equiv \tau_1 \times \tau_2$
  (c) By (b), assumption, and induction: $\Gamma, \Gamma' \vdash E'[e'] : \tau_2$
  (d) By assumption and Environment Validity: $\Gamma, \Gamma' \vdash \Box$
  (e) By (b), (d), and Weakening: $\Gamma, \Gamma' \vdash v : \tau_1$
  (f) By (c), (e), and T-Pair: $\Gamma, \Gamma' \vdash \langle v, E'[e'] \rangle : \tau_1 \times \tau_2$
  (g) By assumption and Validity: $\Gamma \vdash \tau : \Omega$
  (h) By (d), (g), and Weakening: $\Gamma, \Gamma' \vdash \tau : \Omega$
  (i) By (b), (f), (h), and T-Equiv: $\Gamma, \Gamma' \vdash \langle v, E'[e'] \rangle : \tau$
  (j) By (i): $\Gamma, \Gamma' \vdash E[e'] : \tau$

- Case $E = \langle \tau_1, E' \rangle{:}\tau_2$:

  (a) By assumption: $\Gamma \vdash \langle \tau_1, E'[e] \rangle{:}\tau_2 : \tau$
  (b) By (a) and Typing Inversion: $\Gamma \vdash \tau_1 : \kappa$ and $\Gamma \vdash E'[e] : \tau_3[\alpha := \tau_1]$ where $\tau_2 = \exists\alpha{:}\kappa.\tau_3$ and $\tau \equiv \tau_2$ and $\Gamma \vdash \tau_2 : \Omega$
  (c) By (b), assumption, and induction: $\Gamma, \Gamma' \vdash E'[e'] : \tau_3[\alpha := \tau_1]$
  (d) By assumption and Environment Validity: $\Gamma, \Gamma' \vdash \Box$
  (e) By (b), (d), and Weakening: $\Gamma, \Gamma' \vdash \tau_1 : \kappa$ and $\Gamma, \Gamma' \vdash \tau_2 : \Omega$
  (f) By (c), (e), and T-Close: $\Gamma, \Gamma' \vdash \langle \tau_1, E'[e'] \rangle{:}\tau_2 : \tau_2$
  (g) By assumption and Validity: $\Gamma \vdash \tau : \Omega$
  (h) By (d), (g), and Weakening: $\Gamma, \Gamma' \vdash \tau : \Omega$
  (i) By (b), (f), (h), and T-Equiv: $\Gamma, \Gamma' \vdash \langle \tau_1, E'[e'] \rangle{:}\tau_2 : \tau$
  (j) By (i): $\Gamma, \Gamma' \vdash E[e'] : \tau$

- Case $E = \mathsf{let}\ \langle x_1, x_2 \rangle = E'\ \mathsf{in}\ e''$ (w.l.o.g. $\{x_1, x_2\} \cap dom(\Gamma, \Gamma') = \emptyset$):

  (a) By assumption: $\Gamma \vdash \mathsf{let}\ \langle x_1, x_2 \rangle = E'[e]\ \mathsf{in}\ e'' : \tau$
  (b) By (a) and Typing Inversion: $\Gamma \vdash E'[e] : \tau_1 \times \tau_2$ and $\Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e'' : \tau$
  (c) By (b), assumption, and induction: $\Gamma, \Gamma' \vdash E'[e'] : \tau_1 \times \tau_2$
  (d) By (c) and Validity: $\Gamma, \Gamma' \vdash \tau_1 \times \tau_2 : \Omega$
  (e) By (d) and inversion of K-Times: $\Gamma, \Gamma' \vdash \tau_1 : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2 : \Omega$
  (f) By (e) and E-Term: $\Gamma, \Gamma', x_1{:}\tau_1 \vdash \Box$
  (g) By (e), (f), and Weakening: $\Gamma, \Gamma', x_1{:}\tau_1 \vdash \tau_2 : \Omega$
  (h) By (g) and E-Term: $\Gamma, \Gamma', x_1{:}\tau_1, x_2{:}\tau_2 \vdash \Box$
  (i) By (b), (h), and Weakening: $\Gamma, \Gamma', x_1{:}\tau_1, x_2{:}\tau_2 \vdash e'' : \tau$
  (j) By (c), (i), and T-Proj: $\Gamma, \Gamma' \vdash \mathsf{let}\ \langle x_1, x_2 \rangle = E'[e']\ \mathsf{in}\ e'' : \tau$
  (k) By (j): $\Gamma, \Gamma' \vdash E[e'] : \tau$

- Case $E = \mathsf{let}\ \langle \alpha, x \rangle = E'\ \mathsf{in}\ e''$ (w.l.o.g. $\{\alpha, x\} \cap dom(\Gamma, \Gamma') = \emptyset$):

  (a) By assumption: $\Gamma \vdash \mathsf{let}\ \langle \alpha, x \rangle = E'[e]\ \mathsf{in}\ e'' : \tau$
  (b) By (a) and Typing Inversion: $\Gamma \vdash E'[e] : \exists\alpha{:}\kappa.\tau''$ and $\Gamma, \alpha{:}\kappa, x{:}\tau'' \vdash e'' : \tau'''$, where $\alpha \notin ftv(\tau''')$ and $\tau''' \equiv \tau$
  (c) By (b), assumption, and induction: $\Gamma, \Gamma' \vdash E'[e'] : \exists\alpha{:}\kappa.\tau''$
  (d) By assumption and Environment Validity: $\Gamma, \Gamma' \vdash \Box$
  (e) By (d) and E-Type: $\Gamma, \Gamma', \alpha{:}\kappa \vdash \Box$
  (f) By (c) and Validity: $\Gamma, \Gamma' \vdash \exists\alpha{:}\kappa.\tau'' : \Omega$

(g) By (f) and inversion of K-Exist: $\Gamma, \Gamma', \alpha{:}\kappa \vdash \tau'' : \Omega$
(h) By (e), (g), and Weakening: $\Gamma, \Gamma', \alpha{:}\kappa \vdash \tau'' : \Omega$
(i) By (h) and E-Term: $\Gamma, \Gamma', \alpha{:}\kappa, x{:}\tau'' \vdash \Box$
(j) By (b), (i), and Weakening: $\Gamma, \Gamma', \alpha{:}\kappa, x{:}\tau'' \vdash e'' : \tau'''$
(k) By (c), (k), and T-Open: $\Gamma, \Gamma' \vdash \mathsf{let}\ \langle \alpha, x\rangle = E'[e']\ \mathsf{in}\ e'' : \tau'''$
(l) By assumption and Validity: $\Gamma \vdash \tau : \Omega$
(m) By (d), (m), and Weakening: $\Gamma, \Gamma' \vdash \tau : \Omega$
(n) By (b), (l), (n), and T-Equiv: $\Gamma, \Gamma' \vdash \mathsf{let}\ \langle \alpha, x\rangle = E'[e']\ \mathsf{in}\ e'' : \tau$
(o) By (o): $\Gamma, \Gamma' \vdash E[e'] : \tau$

- Case $E = E'\ \tau_2$:

(a) By assumption: $\Gamma \vdash E'[e]\ \tau_2 : \tau$
(b) By (a) and Typing Inversion: $\Gamma \vdash E'[e] : \forall \alpha{:}\kappa.\tau_1$ and $\Gamma \vdash \tau_2 : \kappa$ where $\tau \equiv \tau_1[\alpha := \tau_2]$
(c) By (b), assumption, and induction: $\Gamma, \Gamma' \vdash E'[e'] : \forall \alpha{:}\kappa.\tau_1$
(d) By assumption and Environment Validity: $\Gamma, \Gamma' \vdash \Box$
(e) By (b), (d), and Weakening: $\Gamma, \Gamma' \vdash \tau_2 : \kappa$
(f) By (d), (e), and T-Inst: $\Gamma, \Gamma' \vdash E'[e']\ \tau_2 : \tau_1[\alpha := \tau_2]$
(g) By assumption and Validity: $\Gamma \vdash \tau : \Omega$
(h) By (d), (g), and Weakening: $\Gamma, \Gamma' \vdash \tau : \Omega$
(i) By (b), (f), (h), and T-Equiv: $\Gamma, \Gamma' \vdash E'[e']\ \tau_2 : \tau$
(j) By (i): $\Gamma, \Gamma' \vdash E[e'] : \tau$

- Case $E = \mathsf{tcase}\ E'{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$ (w.l.o.g. $x \notin dom(\Gamma, \Gamma')$):

(a) By assumption: $\Gamma \vdash \mathsf{tcase}\ E'[e]{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau$
(b) By (a) and Typing Inversion:
  i. $\Gamma \vdash E'[e] : \tau_0$
  ii. $\Gamma, x{:}\tau_0' \vdash e_1 : \tau$
  iii. $\Gamma \vdash e_2 : \tau$
(c) By (2a), assumption, and induction: $\Gamma, \Gamma' \vdash E'[e'] : \tau_0$
(d) By assumption and Environment Validity: $\Gamma, \Gamma' \vdash \Box$
(e) By (2b) and Environment Validity: $\Gamma, x{:}\tau_0' \vdash \Box$
(f) By (e) and inversion of E-Term: $\Gamma \vdash \tau_0' : \Omega$
(g) By (d), (f), and Weakening: $\Gamma, \Gamma' \vdash \tau_0' : \Omega$
(h) By (g), assumption, and E-Term: $\Gamma, \Gamma', x{:}\tau_0' \vdash \Box$
(i) By (2b), (h), and Weakening: $\Gamma, \Gamma', x{:}\tau_0' \vdash e_1 : \tau$
(j) By (2c), (d), and Weakening: $\Gamma, \Gamma' \vdash e_2 : \tau$
(k) By (c), (i), (j), and T-Case: $\Gamma, \Gamma' \vdash \mathsf{tcase}\ E'[e']{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau$
(l) By (k): $\Gamma, \Gamma' \vdash E[e'] : \tau$

2. Let $\Gamma \vdash LE[e] : \tau$ and $\Gamma, \Gamma' \vdash e : \tau'$ and $\Gamma, \Gamma' \vdash e' : \tau'$ and $\Gamma \vdash L : \Gamma'$.

- Case $L = \_$:

(a) By assumption and inversion of L-Empty: $\Gamma \vdash E[e] : \tau$ and $\Gamma' = \cdot$
(b) By (a) and assumption: $\Gamma \vdash e : \tau'$ and $\Gamma \vdash e' : \tau'$
(c) By (b) and (1): $\Gamma \vdash E[e'] : \tau$
(d) By (c): $\Gamma \vdash LE[e'] : \tau$

- Case $L = \mathsf{lazy}\ \langle\zeta, x\rangle$ = e_1 in L':
  (a) By assumption: $\Gamma \vdash \mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ L'E[e] : \tau$
  (b) By (a) and Typing Inversion: $\Gamma \vdash e_1 : \exists\alpha{:}\kappa.\tau_1$ and $\Gamma, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash L'E[e] : \tau'''$ where $\zeta \notin ftv(\tau''')$ and $\tau''' \equiv \tau$
  (c) We show: $\Gamma, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash L'E[e'] : \tau'''$
    i. By assumption and inversion of L-Lazy: $\Gamma' = \zeta{:}\kappa', x{:}\tau_1'[\alpha := \zeta], \Gamma''$ and $\Gamma \vdash e_1 : \exists\alpha{:}\kappa'.\tau_1'$ and $\Gamma, \zeta{:}\kappa', x{:}\tau_1'[\alpha := \zeta] \vdash L' : \Gamma''$
    ii. By (i), (b), and Uniqueness of Types: $\exists\alpha{:}\kappa.\tau_1 \equiv \exists\alpha{:}\kappa'.\tau_1'$
    iii. By (ii) and Type Equivalence Inversion: $\kappa = \kappa'$ and $\tau_1 \equiv \tau_1'$
    iv. By (iii): $\Gamma, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \equiv \Gamma, \zeta{:}\kappa', x{:}\tau_1'[\alpha := \zeta]$
    v. We show: $\Gamma, \zeta{:}\kappa', x{:}\tau_1'[\alpha := \zeta] \vdash \Box$
      – Subcase $L' = \_$:
        A. By (i) and inversion of L-Empty: $\Gamma, \zeta{:}\kappa', x{:}\tau_1'[\alpha := \zeta] \vdash \Box$
      – Subcase $L' = \mathsf{lazy}\ \langle\zeta', x'\rangle = e_1'\ \mathsf{in}\ L''$:
        A. By (i) and inversion of L-Lazy: $\Gamma, \zeta{:}\kappa', x{:}\tau_1'[\alpha := \zeta] \vdash e_1' : \exists\alpha'{:}\kappa'''.\tau_1''$
        B. By (A) and Environment Validity: $\Gamma, \zeta{:}\kappa', x{:}\tau_1'[\alpha := \zeta] \vdash \Box$
    vi. By (b), (iv), (v), and Equivalent Environments: $\Gamma, \zeta{:}\kappa', x{:}\tau_1'[\alpha := \zeta] \vdash L'E[e] : \tau'''$
    vii. By (vi), (i), assumption, and induction: $\Gamma, \zeta{:}\kappa', x{:}\tau_1'[\alpha := \zeta] \vdash L'E[e'] : \tau'''$
    viii. By (b) and Environment Validity: $\Gamma, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash \Box$
    ix. By (vii), (iv), (viii), and Equivalent Environments: $\Gamma, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash L'E[e'] : \tau'''$
  (d) By (b), (c), and T-Lazy: $\Gamma \vdash \mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ L'E[e'] : \tau'''$
  (e) By assumption and Validity: $\Gamma \vdash \tau : \Omega$
  (f) By (b), (d), (e), and T-Equiv: $\Gamma \vdash \mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ L'E[e'] : \tau$
  (g) By (f): $\Gamma \vdash LE[e'] : \tau$

□

Lemma 9 (Lazy Term Variables). *If $\Gamma \vdash LE[x] : \tau$ and $x \notin dom(\Gamma)$, then $L = L_1[\mathsf{lazy}\ \langle\zeta, x\rangle = e\ \mathsf{in}\ L_2]$ where $x \notin bv(L_2)$.*

Proof. By structural induction on L. W.l.o.g. all bound variables of $LE[x]$ are distinct and $(btv(LE[x]) \cup bv(LE[x])) \cap dom(\Gamma) = \emptyset$.

- Case $L = \_$:
  1. By Context Elimination: $\Gamma \vdash L[x] : \tau'$
  2. By (1) and Context Elimination: $\Gamma \vdash x : \tau''$
  3. By (2): $x \in dom(\Gamma)$
  4. By (2) and Variable Containment: $x \in dom(\Gamma)$
  5. (3) contradicts the assumption $x \notin dom(\Gamma)$, hence this case is not possible.
- Case $L = \mathsf{lazy}\ \langle\zeta, x'\rangle = e\ \mathsf{in}\ L'$:
  – Subcase $x' = x \wedge x \notin bv(L')$: this is what we claimed
  – Subcase $x' = x \wedge x \in bv(L')$: not possible due to our assumption about bound variables
  – Subcase $x' \neq x$:

normal forms	$\nu ::=$	$p \mid \nu_1 \to \nu_2 \mid \nu_1 \times \nu_2 \mid \forall\alpha{:}\kappa.\nu \mid \exists\alpha{:}\kappa.\nu \mid \lambda\alpha{:}\kappa.\nu \mid \langle\nu_1,\nu_2\rangle$
normal paths	$p ::=$	$\alpha \mid p\ \nu \mid p.1 \mid p.2$
type reduction contexts	$T ::=$	$\_ \mid T \to \tau \mid \nu \to T \mid T \times \tau \mid \nu \times T \mid \forall\alpha{:}\kappa.T \mid \exists\alpha{:}\kappa.T \mid \lambda\alpha{:}\kappa.T \mid$ $T\ \tau \mid \nu\ T \mid \langle T,\tau\rangle \mid \langle\nu,T\rangle \mid T.1 \mid T.2$
tcase contexts	$C ::=$	$\mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\tau\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mid \mathsf{tcase}\ v{:}\nu\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$

Reduction $\boxed{e \longrightarrow e'}$

R-App	$LE[(\lambda x{:}\tau.e)\ v]$	$\longrightarrow LE[e[x := v]]$	
R-Inst	$LE[(\lambda\alpha{:}\kappa.e)\ \tau]$	$\longrightarrow LE[e[\alpha := \tau]]$	
R-Proj	$LE[\mathsf{let}\ \langle x_1,x_2\rangle = \langle v_1,v_2\rangle\ \mathsf{in}\ e]$	$\longrightarrow LE[e[x_1 := v_1][x_2 := v_2]]$	
R-Open	$LE[\mathsf{let}\ \langle\alpha,x\rangle = \langle\tau,v\rangle{:}\tau'\ \mathsf{in}\ e]$	$\longrightarrow LE[e[\alpha := \tau][x := v]]$	
R-Suspend	$LE[\mathsf{lazy}\ \langle\zeta,x\rangle = e_1\ \mathsf{in}\ e_2]$	$\longrightarrow L[\mathsf{lazy}\ \langle\zeta,x\rangle = e_1\ \mathsf{in}\ E[e_2]]$	$(E \neq \_ \wedge \zeta \notin ftv(E) \wedge x \notin fv(E))$
R-Trigger	$L_1[\mathsf{lazy}\ \langle\zeta,x\rangle = e\ \mathsf{in}\ L_2ES[x]]$	$\longrightarrow L_1[\mathsf{let}\ \langle\alpha,x\rangle = e\ \mathsf{in}\ (L_2ES[x])[\zeta := \alpha]]$	$(x \notin btv(L_2))$
R-Case1	$LE[\mathsf{tcase}\ v{:}\nu\ \mathsf{of}\ x{:}\nu\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]$	$\longrightarrow LE[e_1[x := v]]$	
R-Case2	$LE[\mathsf{tcase}\ v{:}\nu\ \mathsf{of}\ x{:}\nu'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]$	$\longrightarrow LE[e_2]$	$(\nu \neq \nu')$
RT-App	$LECT[(\lambda\alpha{:}\kappa.\nu)\ \nu']$	$\longrightarrow LECT[\nu[\alpha := \nu']]$	
RT-Proj1	$LECT[\langle\nu_1,\nu_2\rangle.1]$	$\longrightarrow LECT[\nu_1]$	
RT-Proj2	$LECT[\langle\nu_1,\nu_2\rangle.2]$	$\longrightarrow LECT[\nu_2]$	
RT-Trigger	$L_1[\mathsf{lazy}\ \langle\zeta,x\rangle = e_1\ \mathsf{in}\ L_2ECT[\zeta]]$	$\longrightarrow L_1[\mathsf{let}\ \langle\alpha,x\rangle = e_1\ \mathsf{in}\ (L_2ECT[\zeta])[\zeta := \alpha]]$	$(\zeta \notin btv(L_2))$

Figure A.4: The basic calculus + applicative order reduction to normal form on type-level (operational semantics)

1. By assumption and Typing Inversion: $\Gamma,\zeta{:}\kappa,x'{:}\tau'' \vdash L'[x] : \tau'''$
2. By assumption: $x \notin dom(\Gamma,\zeta{:}\kappa,x'{:}\tau'')$
3. By (1), (2), and induction: $L' = L'_1[\mathsf{lazy}\ \langle\zeta',x\rangle = e'\ \mathsf{in}\ L_2]$ where $x \notin bv(L_2)$
4. Let $L_1 = \mathsf{lazy}\ \langle\zeta,x'\rangle = e\ \mathsf{in}\ L'_1$.
5. By (3) and (4): $L = L_1[\mathsf{lazy}\ \langle\zeta',x\rangle = e'\ \mathsf{in}\ L_2]$ where $x \notin bv(L_2)$

□

A.2 Applicative order reduction to normal form

Lemma 10 (Type Context Elimination)**.**

1. If $\Gamma \vdash T[\tau] : \kappa$*, then* $\Gamma,\Gamma' \vdash \tau : \kappa'$ *where* $\Gamma \vdash T : \Gamma'$.

2. If $\Gamma \vdash T : \Gamma'$*, then* $dom(\Gamma') \cap LTVar = \emptyset$.

3. If $\Gamma \vdash C[\tau] : \tau'$*, then* $\Gamma \vdash \tau : \Omega$.

Proof. (1) by induction on the structure of T. (2) follows immediately from the definition of the $\Gamma \vdash T : \Gamma'$ judgement. (3) by case analysis on C.

1. Let $\Gamma \vdash T[\tau] : \kappa$

   - Case $T = \_$:
     (a) By assumption: $\Gamma \vdash \tau : \kappa$
   - Case $T = T' \to \tau'$:
     (a) By assumption: $\Gamma \vdash T'[\tau] \to \tau' : \kappa$
     (b) By (a) and inversion of K-Arrow: $\Gamma \vdash T'[\tau] : \Omega$
     (c) By (b) and induction: $\Gamma, \Gamma' \vdash \tau : \Omega$ where $\Gamma \vdash T' : \Gamma'$
     (d) By (c) and TC-Arrow1: $\Gamma \vdash T' \to \tau' : \Gamma'$
   - Case $T = \nu \to T'$:
     (a) By assumption: $\Gamma \vdash \nu \to T'[\tau] : \kappa$
     (b) By (a) and inversion of K-Arrow: $\Gamma \vdash T'[\tau] : \Omega$
     (c) By (b) and induction: $\Gamma, \Gamma' \vdash \tau : \Omega$ where $\Gamma \vdash T' : \Gamma'$
     (d) By (c) and TC-Arrow2: $\Gamma \vdash \nu \to T' : \Gamma'$
   - Case $T = T' \times \tau'$:
     (a) By assumption: $\Gamma \vdash T'[\tau] \times \tau' : \kappa$
     (b) By (a) and inversion of K-Times: $\Gamma \vdash T'[\tau] : \Omega$
     (c) By (b) and induction: $\Gamma, \Gamma' \vdash \tau : \Omega$ where $\Gamma \vdash T' : \Gamma'$
     (d) By (c) and TC-Times1: $\Gamma \vdash T' \times \tau' : \Gamma'$
   - Case $T = \nu \times T'$:
     (a) By assumption: $\Gamma \vdash \nu \times T'[\tau] : \kappa$
     (b) By (a) and inversion of K-Times: $\Gamma \vdash T'[\tau] : \Omega$
     (c) By (b) and induction: $\Gamma, \Gamma' \vdash \tau : \Omega$ where $\Gamma \vdash T' : \Gamma'$
     (d) By (c) and TC-Times2: $\Gamma \vdash \nu \times T' : \Gamma'$
   - Case $T = T'\ \tau'$:
     (a) By assumption: $\Gamma \vdash T'[\tau]\ \tau' : \kappa$
     (b) By (a) and inversion of K-App: $\Gamma \vdash T'[\tau] : \kappa'' \to \kappa$
     (c) By (b) and induction: $\Gamma, \Gamma' \vdash \tau : \kappa'$ where $\Gamma \vdash T' : \Gamma'$
     (d) By (c) and TC-App1: $\Gamma \vdash T'\ \tau' : \Gamma'$
   - Case $T = \nu\ T'$:
     (a) By assumption: $\Gamma \vdash \nu\ T'[\tau] : \kappa$
     (b) By (a) and inversion of K-App: $\Gamma \vdash T'[\tau] : \kappa''$
     (c) By (b) and induction: $\Gamma, \Gamma' \vdash \tau : \kappa'$ where $\Gamma \vdash T' : \Gamma'$
     (d) By (c) and TC-App2: $\Gamma \vdash \nu\ T' : \Gamma'$
   - Case $T = \forall \alpha{:}\kappa.T'$:
     (a) By assumption: $\Gamma \vdash \forall \alpha{:}\kappa.T'[\tau] : \kappa$
     (b) By (a) and inversion of K-Univ: $\Gamma, \alpha{:}\kappa \vdash T'[\tau] : \Omega$
     (c) By (b) and induction: $\Gamma, \alpha{:}\kappa, \Gamma'' \vdash \tau : \kappa'$ where $\Gamma \vdash T' : \Gamma''$
     (d) Let $\Gamma' = \alpha{:}\kappa, \Gamma''$.
     (e) By (b), (c), (d), and TC-Univ: $\Gamma, \Gamma' \vdash \tau : \kappa'$ where $\Gamma \vdash \forall \alpha{:}\kappa.T' : \Gamma'$
   - Case $T = \exists \alpha{:}\kappa.T'$:
     (a) By assumption: $\Gamma \vdash \exists \alpha{:}\kappa.T'[\tau] : \kappa$
     (b) By (a) and inversion of K-Exist: $\Gamma, \alpha{:}\kappa \vdash T'[\tau] : \Omega$

(c) By (b) and induction: $\Gamma, \alpha{:}\kappa, \Gamma'' \vdash \tau : \kappa'$ where $\Gamma \vdash T' : \Gamma''$

(d) Let $\Gamma' = \alpha{:}\kappa, \Gamma''$.

(e) By (b), (c), (d), and TC-Exist: $\Gamma, \Gamma' \vdash \tau : \kappa'$ where $\Gamma \vdash \exists\alpha{:}\kappa.T' : \Gamma'$

- Case $T = \lambda\alpha{:}\kappa.T'$:

  (a) By assumption: $\Gamma \vdash \lambda\alpha{:}\kappa.T'[\tau] : \kappa$

  (b) By (a) and inversion of K-Abs: $\Gamma, \alpha{:}\kappa \vdash T'[\tau] : \kappa''$

  (c) By (b) and induction: $\Gamma, \alpha{:}\kappa, \Gamma'' \vdash \tau : \kappa'$ where $\Gamma \vdash T' : \Gamma''$

  (d) Let $\Gamma' = \alpha{:}\kappa, \Gamma''$.

  (e) By (b), (c), (d), and TC-Abs: $\Gamma, \Gamma' \vdash \tau : \kappa'$ where $\Gamma \vdash \lambda\alpha{:}\kappa.T' : \Gamma'$

3. Let $\Gamma \vdash C[\tau] : \tau'$.

- Case $C = \mathsf{tcase}\ e_0{:}\_\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$:

  (a) By assumption: $\Gamma \vdash \mathsf{tcase}\ e_0{:}\tau\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau'$

  (b) By (a) and Typing Inversion: $\Gamma \vdash e_0 : \tau$

  (c) By (b) and Validity: $\Gamma \vdash \tau : \Omega$

- Case $C = \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$:

  (a) By assumption: $\Gamma \vdash \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau'$

  (b) By (a) and Typing Inversion: $\Gamma, x{:}\tau \vdash e_1 : \tau'$

  (c) By (b) and Environment Validity: $\Gamma, x{:}\tau \vdash \Box$

  (d) By (c) and inversion of E-Term: $\Gamma \vdash \tau : \Omega$

□

Lemma 11 (Wrapping). *If $\tau \equiv \tau'$, then, for all T, $T[\tau] \equiv T[\tau']$.*

Proof. By structural induction on T.

- Case $T = \_$:

  1. By assumption: $\tau \equiv \tau'$

- Case $T = T' \to \tau''$:

  1. By induction: $T'[\tau] \equiv T'[\tau']$
  2. By Q-Refl: $\tau'' \equiv \tau''$
  3. By (1), (2), and Q-Arrow: $T'[\tau] \to \tau'' \equiv T'[\tau'] \to \tau''$

- Case $T = \nu \to T'$:

  1. By induction: $T'[\tau] \equiv T'[\tau']$
  2. By Q-Refl: $\nu \equiv \nu$
  3. By (1), (2), and Q-Arrow: $\nu \to T'[\tau] \equiv \nu \to T'[\tau']$

- Case $T = T' \times \tau''$:

  1. By induction: $T'[\tau] \equiv T'[\tau']$
  2. By Q-Refl: $\tau'' \equiv \tau''$
  3. By (1), (2), and Q-Times: $T'[\tau] \times \tau'' \equiv T'[\tau'] \times \tau''$

- Case $T = \nu \times T'$:

1. By induction: $T'[\tau] \equiv T'[\tau']$
2. By Q-Refl: $\nu \equiv \nu$
3. By (1), (2), and Q-Times: $\nu \times T'[\tau] \equiv \nu \times T'[\tau']$

- Case $T = \forall\alpha{:}\kappa.T'$:
  1. By induction: $T'[\tau] \equiv T'[\tau']$
  2. By (1) and Q-Univ: $\forall\alpha{:}\kappa.T'[\tau] \equiv \forall\alpha{:}\kappa.T'[\tau']$
- Case $T = \exists\alpha{:}\kappa''.T'$:
  1. By induction: $T'[\tau] \equiv T'[\tau']$
  2. By (1) and Q-Exist: $\exists\alpha{:}\kappa.T'[\tau] \equiv \exists\alpha{:}\kappa.T'[\tau']$
- Case $T = \lambda\alpha{:}\kappa_1.T'$:
  1. By induction: $T'[\tau] \equiv T'[\tau']$
  2. By (1) and Q-Abs: $\lambda\alpha{:}\kappa.T'[\tau] \equiv \lambda\alpha{:}\kappa.T'[\tau']$
- Case $T = T'\ \tau''$:
  1. By induction: $T'[\tau] \equiv T'[\tau']$
  2. By Q-Refl: $\tau'' \equiv \tau''$
  3. By (1), (2), and Q-App: $T'[\tau]\ \tau'' \equiv T'[\tau']\ \tau''$
- Case $T = \nu\ T'$:
  1. By induction: $T'[\tau] \equiv T'[\tau']$
  2. By Q-Refl: $\nu \equiv \nu$
  3. By (1), (2), and Q-App: $\nu\ T'[\tau] \equiv \nu\ T'[\tau']$
- Case $T = \langle T', \tau''\rangle$:
  1. By induction: $T'[\tau] \equiv T'[\tau']$
  2. By Q-Refl: $\tau'' \equiv \tau''$
  3. By (1), (2), and Q-Pair: $\langle T'[\tau], \tau''\rangle \equiv \langle T'[\tau'], \tau''\rangle$
- Case $T = \langle \nu, T'\rangle$:
  1. By induction: $T'[\tau] \equiv T'[\tau']$
  2. By Q-Refl: $\nu \equiv \nu$
  3. By (1), (2), and Q-App: $\langle \nu, T'[\tau]\rangle \equiv \langle \nu, T'[\tau']\rangle$
- Case $T = T'.1$:
  1. By induction: $T'[\tau] \equiv T'[\tau']$
  2. By (1) and Q-Proj1a: $T'[\tau].1 \equiv T'[\tau'].1$
- Case $T = T'.2$:
  1. By induction: $T'[\tau] \equiv T'[\tau']$
  2. By (1) and Q-Proj2a: $T'[\tau].2 \equiv T'[\tau'].2$

□

Lemma 12 (Type Exchange).

1. *If* $\Gamma \vdash T[\tau] : \kappa$ *and* $\Gamma, \Gamma' \vdash \tau : \kappa$ *and* $\Gamma, \Gamma' \vdash \tau' : \kappa$ *where* $\Gamma \vdash T : \Gamma'$, *then* $\Gamma \vdash T[\tau'] : \kappa$.
2. *If* $\Gamma \vdash CT[\tau] : \tau''$ *and* $\Gamma, \Gamma' \vdash \tau : \kappa$ *and* $\Gamma, \Gamma' \vdash \tau' : \kappa$ *and* $\Gamma \vdash T : \Gamma'$ *and* $\tau \equiv \tau'$, *then* $\Gamma \vdash CT[\tau'] : \tau''$.

Proof. (1) by structural induction on T. (2) by case analysis on C.

1. Let $\Gamma \vdash T[\tau] : \kappa$ and $\Gamma, \Gamma' \vdash \tau : \kappa'$ and $\Gamma, \Gamma' \vdash \tau' : \kappa'$ where $\Gamma \vdash T : \Gamma'$ and w.l.o.g. $btv(T[\tau]) \cap dom(\Gamma) = \emptyset$.
   - Case $T = \_$:
     (a) By assumption and inversion of TC-Empty: $\Gamma' = \cdot$
     (b) By assumption and Uniqueness of Kinds: $\kappa = \kappa'$
     (c) By (b) and assumption: $\Gamma \vdash \tau' : \kappa$
   - Case $T = T' \to \tau''$:
     (a) By assumption: $\Gamma \vdash T'[\tau] \to \tau'' : \kappa$
     (b) By (a) and inversion of K-Arrow: $\Gamma \vdash T'[\tau] : \Omega$ and $\Gamma \vdash \tau'' : \Omega$
     (c) By (b) and induction: $\Gamma \vdash T'[\tau'] : \Omega$
     (d) By (c), (b), and K-Arrow: $\Gamma \vdash T'[\tau'] \to \tau'' : \Omega$
   - Case $T = \nu \to T'$:
     (a) By assumption: $\Gamma \vdash \nu \to T'[\tau] : \kappa$
     (b) By (a) and inversion of K-Arrow: $\Gamma \vdash \nu : \Omega$ and $\Gamma \vdash T'[\tau] : \Omega$
     (c) By (b) and induction: $\Gamma \vdash T'[\tau'] : \Omega$
     (d) By (c), (b), and K-Arrow: $\Gamma \vdash \nu \to T'[\tau'] : \Omega$
   - Case $T = T' \times \tau''$:
     (a) By assumption: $\Gamma \vdash T'[\tau] \times \tau'' : \kappa$
     (b) By (a) and inversion of K-Times: $\Gamma \vdash T'[\tau] : \Omega$ and $\Gamma \vdash \tau'' : \Omega$
     (c) By (b) and induction: $\Gamma \vdash T'[\tau'] : \Omega$
     (d) By (c), (b), and K-Times: $\Gamma \vdash T'[\tau'] \times \tau'' : \Omega$
   - Case $T = \nu \times T'$:
     (a) By assumption: $\Gamma \vdash \nu \times T'[\tau] : \kappa$
     (b) By (a) and inversion of K-Times: $\Gamma \vdash \nu : \Omega$ and $\Gamma \vdash T'[\tau] : \Omega$
     (c) By (b) and induction: $\Gamma \vdash T'[\tau'] : \Omega$
     (d) By (c), (b), and K-Times: $\Gamma \vdash \nu \times T'[\tau'] : \Omega$
   - Case $T = \forall \alpha{:}\kappa''.T'$:
     (a) By assumption: $\Gamma \vdash \forall \alpha{:}\kappa''.T'[\tau] : \Omega$
     (b) By (a), assumption, and inversion of K-Univ: $\Gamma, \alpha{:}\kappa'' \vdash T'[\tau] : \Omega$
     (c) By assumption and inversion of TC-Univ: $\Gamma' = \alpha{:}\kappa'', \Gamma''$ and $\Gamma, \alpha{:}\kappa'' \vdash T' : \Gamma''$
     (d) By (b), (c), assumption, and induction: $\Gamma, \alpha{:}\kappa'' \vdash T'[\tau'] : \Omega$
     (e) By (d) and K-Univ: $\Gamma \vdash \forall \alpha{:}\kappa''.T'[\tau'] : \Omega$
   - Case $T = \exists \alpha{:}\kappa''.T'$:
     (a) By assumption: $\Gamma \vdash \exists \alpha{:}\kappa''.T'[\tau] : \Omega$
     (b) By (a), assumption, and inversion of K-Exist: $\Gamma, \alpha{:}\kappa'' \vdash T'[\tau] : \Omega$

(c) By assumption and inversion of TC-Exist: $\Gamma' = \alpha{:}\kappa'', \Gamma''$ and $\Gamma, \alpha{:}\kappa'' \vdash T' : \Gamma''$

(d) By (b), (c), assumption, and induction: $\Gamma, \alpha{:}\kappa'' \vdash T'[\tau'] : \Omega$

(e) By (d) and K-Exist: $\Gamma \vdash \exists\alpha{:}\kappa''.T'[\tau'] : \Omega$

- Case $T = \lambda\alpha{:}\kappa_1.T'$:

  (a) By assumption: $\Gamma \vdash \lambda\alpha{:}\kappa_1.T'[\tau] : \kappa$

  (b) By (a), assumption, and inversion of K-Abs: $\kappa = \kappa_1 \to \kappa_2$ and $\Gamma, \alpha{:}\kappa_1 \vdash T'[\tau] : \kappa_2$

  (c) By assumption and inversion of TC-Abs: $\Gamma' = \alpha{:}\kappa_1, \Gamma''$ and $\Gamma, \alpha{:}\kappa_1 \vdash T' : \Gamma''$

  (d) By (b), (c), assumption, and induction: $\Gamma, \alpha{:}\kappa_1 \vdash T'[\tau'] : \kappa_2$

  (e) By (d) and K-Abs: $\Gamma \vdash \lambda\alpha{:}\kappa_1.T'[\tau'] : \kappa_1 \to \kappa_2$

- Case $T = T'\ \tau''$:

  (a) By assumption: $\Gamma \vdash T'[\tau]\ \tau'' : \kappa$

  (b) By (a) and inversion of K-App: $\Gamma \vdash T'[\tau] : \kappa'' \to \kappa$ and $\Gamma \vdash \tau'' : \kappa''$

  (c) By (b) and induction: $\Gamma \vdash T'[\tau'] : \kappa'' \to \kappa$

  (d) By (c), (b), and K-App: $\Gamma \vdash T'[\tau']\ \tau'' : \kappa$

- Case $T = \nu\ T'$:

  (a) By assumption: $\Gamma \vdash \nu\ T'[\tau] : \kappa$

  (b) By (a) and inversion of K-App: $\Gamma \vdash \nu : \kappa'' \to \kappa$ and $\Gamma \vdash T'[\tau] : \kappa''$

  (c) By (b) and induction: $\Gamma \vdash T'[\tau'] : \kappa''$

  (d) By (c), (b), and K-App: $\Gamma \vdash \nu\ T'[\tau'] : \kappa$

- Case $T = \langle T', \tau'' \rangle$:

  (a) By assumption: $\Gamma \vdash \langle T'[\tau], \tau'' \rangle : \kappa$

  (b) By (a) and inversion of K-Pair: $\kappa = \kappa_1 \times \kappa_2$ and $\Gamma \vdash T'[\tau] : \kappa_1$ and $\Gamma \vdash \tau'' : \kappa_2$

  (c) By (b) and induction: $\Gamma \vdash T'[\tau'] : \kappa_1$

  (d) By (c), (b), and K-Pair: $\Gamma \vdash \langle T'[\tau'], \tau'' \rangle : \kappa_1 \times \kappa_2$

- Case $T = \langle \nu, T' \rangle$:

  (a) By assumption: $\Gamma \vdash \langle \nu, T'[\tau] \rangle : \kappa$

  (b) By (a) and inversion of K-Pair: $\kappa = \kappa_1 \times \kappa_2$ and $\Gamma \vdash \nu : \kappa_1$ and $\Gamma \vdash T'[\tau] : \kappa_2$

  (c) By (b) and induction: $\Gamma \vdash T'[\tau'] : \kappa_2$

  (d) By (c), (b), and K-Pair: $\Gamma \vdash \langle \nu, T'[\tau'] \rangle : \kappa_1 \times \kappa_2$

- Case $T = T'.1$:

  (a) By assumption: $\Gamma \vdash T'[\tau].1 : \kappa$

  (b) By (a) and inversion of K-Proj1: $\Gamma \vdash T'[\tau] : \kappa \times \kappa_2$

  (c) By (b) and induction: $\Gamma \vdash T'[\tau'] : \kappa \times \kappa_2$

  (d) By (c), (b), and K-App: $\Gamma \vdash T'[\tau'].1 : \kappa$

- Case $T = T'.2$:

  (a) By assumption: $\Gamma \vdash T'[\tau].2 : \kappa$

  (b) By (a) and inversion of K-Proj2: $\Gamma \vdash T'[\tau] : \kappa_1 \times \kappa$

  (c) By (b) and induction: $\Gamma \vdash T'[\tau'] : \kappa_1 \times \kappa$

  (d) By (c), (b), and K-App: $\Gamma \vdash T'[\tau'].2 : \kappa$

2. Let $\Gamma \vdash CT[\tau] : \tau''$ and $\Gamma, \Gamma' \vdash \tau : \kappa$ and $\Gamma, \Gamma' \vdash \tau' : \kappa$ and $\Gamma \vdash T : \Gamma'$ and $\tau \equiv \tau'$.

   - Case $C = \mathsf{tcase}\ e_0{:}\_\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$:

     (a) By assumption: $\Gamma \vdash \mathsf{tcase}\ e_0{:}T[\tau]\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau''$

(b) By (a) and Typing Inversion: $\Gamma \vdash e_0 : T[\tau]$ and $\Gamma, x{:}\tau_0' \vdash e_1 : \tau''$ and $\Gamma \vdash e_2 : \tau''$
(c) By (b) and Validity: $\Gamma \vdash T[\tau] : \Omega$
(d) By (c), assumption, and (1): $\Gamma \vdash T[\tau'] : \Omega$
(e) By assumption and Wrapping: $T[\tau] \equiv T[\tau']$
(f) By (b), (d), (e), and T-Equiv: $\Gamma \vdash e_0 : T[\tau']$
(g) By (f), (b), and T-Case: $\Gamma \vdash \mathsf{tcase}\ e_0{:}T[\tau']\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau''$

- Case $C = \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$:

(a) By assumption: $\Gamma \vdash \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}T[\tau]\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau''$
(b) By (a) and Typing Inversion: $\Gamma \vdash e_0 : \tau_0$ and $\Gamma, x{:}T[\tau] \vdash e_1 : \tau''$ and $\Gamma \vdash e_2 : \tau''$
(c) By (b) and Environment Validity: $\Gamma, x{:}T[\tau] \vdash \Box$
(d) By (c) and inversion of E-Term: $\Gamma \vdash T[\tau] : \Omega$ and $x \notin dom(\Gamma)$
(e) By (d), assumption, and (1): $\Gamma \vdash T[\tau'] : \Omega$
(f) By (e), (d), and E-Term: $\Gamma, x{:}T[\tau'] \vdash \Box$
(g) By assumption and Wrapping: $T[\tau] \equiv T[\tau']$
(h) By (b), (f), (g), and Equivalent Environments: $\Gamma, x{:}T[\tau'] \vdash e_1 : \tau''$
(i) By (h), (b), and T-Case: $\Gamma \vdash \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}T[\tau']\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau''$

□

Theorem 1 (Preservation)**.** *If* $\Gamma \vdash e : \tau$ *and* $e \longrightarrow e'$*, then* $\Gamma \vdash e' : \tau$*.*

Proof. By case analysis on the applied reduction rule.

- Case R-App: $e = LE[(\lambda x{:}\tau_1.e_1)\ v]$ and $e' = LE[e_1[x := v]]$:

1. By Context Elimination: $\Gamma, \Gamma' \vdash (\lambda x{:}\tau_1.e_1)\ v : \tau'$ where $\Gamma \vdash L : \Gamma'$
2. We show: $\Gamma, \Gamma' \vdash e_1[x := v] : \tau'$
   (a) By (1) and Typing Inversion: $\Gamma, \Gamma' \vdash \lambda x{:}\tau_1.e_1 : \tau_1' \to \tau'$ and $\Gamma, \Gamma' \vdash v : \tau_1'$
   (b) By (a) and Typing Inversion: $\Gamma, \Gamma', x{:}\tau_1 \vdash e_1 : \tau_2$ and $\tau_1' \to \tau' \equiv \tau_1 \to \tau_2$
   (c) By (b) and Type Equivalence Inversion: $\tau_1' \equiv \tau_1$ and $\tau' \equiv \tau_2$
   (d) By (b) and Environment Validity: $\Gamma, \Gamma', x{:}\tau_1 \vdash \Box$
   (e) By (d) and inversion of E-Term: $\Gamma, \Gamma' \vdash \tau_1 : \Omega$
   (f) By (a), (c), (e), and T-Equiv: $\Gamma, \Gamma' \vdash v : \tau_1$
   (g) By (b), (f), and Term Substitution: $\Gamma, \Gamma' \vdash e_1[x := v] : \tau_2$
   (h) By (1) and Validity: $\Gamma, \Gamma' \vdash \tau' : \Omega$
   (i) By (h), (h), (c), and T-Equiv: $\Gamma, \Gamma' \vdash e_1[x := v] : \tau'$
3. By assumption, (1), (2), and Exchange: $\Gamma \vdash LE[e_1[x := v]] : \tau$

- Case R-Inst: $e = LE[(\lambda\alpha{:}\kappa.e_1)\ \tau']$ and $e' = LE[e_1[\alpha := \tau']]$:

1. By Context Elimination: $\Gamma, \Gamma' \vdash (\lambda\alpha{:}\kappa.e)\ \tau' : \tau''$ where $\Gamma \vdash L : \Gamma'$ and w.l.o.g. $\alpha \notin dom(\Gamma, \Gamma')$
2. We show: $\Gamma, \Gamma' \vdash e[\alpha := \tau'] : \tau''$
   (a) By (1) and Typing Inversion: $\Gamma, \Gamma' \vdash \lambda\alpha{:}\kappa.e_1 : \forall\alpha'{:}\kappa'.\tau_1$ (w.l.o.g. $\alpha' = \alpha$) and $\tau'' \equiv \tau_1[\alpha' := \tau']$ where $\Gamma, \Gamma' \vdash \tau' : \kappa'$
   (b) By (a) and Typing Inversion: $\Gamma, \Gamma', \alpha{:}\kappa \vdash e_1 : \tau_1'$ and $\forall\alpha'{:}\kappa'.\tau_1 \equiv \forall\alpha{:}\kappa.\tau_1'$
   (c) By (b) and Type Equivalence Inversion: $\kappa' = \kappa$ and $\tau_1 \equiv \tau_1'$
   (d) By (b), (c), (a), and Type Substitution: $\Gamma, \Gamma' \vdash e_1[\alpha := \tau'] : \tau_1'[\alpha := \tau']$

(e) By (c) and Type Substitution: $\tau_1[\alpha := \tau'] \equiv \tau'_1[\alpha := \tau']$

(f) By (a), (d), Q-Sym, and Q-Trans: $\tau'_1[\alpha := \tau'] \equiv \tau''$

(g) By (1) and Validity: $\Gamma, \Gamma' \vdash \tau'' : \Omega$

(h) By (d), (f), (g), and T-Equiv: $\Gamma, \Gamma' \vdash e_1[\alpha := \tau'] : \tau''$

3. By assumption, (1), (2), and Exchange: $\Gamma \vdash LE[e_1[\alpha := \tau']] : \tau$

- Case R-Proj: $e = LE[\mathsf{let}\ \langle x_1, x_2\rangle = \langle v_1, v_2\rangle\ \mathsf{in}\ e_1]$ and $e' = LE[e_1[x_1 := v_1][x_2 := v_2]]$:

1. By Context Elimination: $\Gamma, \Gamma' \vdash \mathsf{let}\ \langle x_1, x_2\rangle = \langle v_1, v_2\rangle\ \mathsf{in}\ e_1 : \tau'$ where $\Gamma \vdash L : \Gamma'$

2. We show: $\Gamma, \Gamma' \vdash e_1[x_1 := v_1][x_2 := v_2] : \tau'$

(a) By (1) and Typing Inversion: $\Gamma, \Gamma' \vdash \langle v_1, v_2\rangle : \tau_1 \times \tau_2$ and $\Gamma, \Gamma', x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_1 : \tau'$

(b) By (a) and Typing Inversion: $\Gamma, \Gamma' \vdash v_1 : \tau'_1$ and $\Gamma, \Gamma' \vdash v_2 : \tau'_2$ where $\tau_1 \times \tau_2 \equiv \tau'_1 \times \tau'_2$

(c) By (b) and Type Equivalence Inversion: $\tau_1 \equiv \tau'_1$ and $\tau_2 \equiv \tau'_2$

(d) By (a) and Validity: $\Gamma, \Gamma' \vdash \tau_1 \times \tau_2 : \Omega$

(e) By (d) and inversion of K-Times: $\Gamma, \Gamma' \vdash \tau_1 : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2 : \Omega$

(f) By (b), (c), (e), and T-Equiv: $\Gamma, \Gamma' \vdash v_1 : \tau_1$ and $\Gamma, \Gamma' \vdash v_2 : \tau_2$

(g) By (a), (f), and Term Substitution: $\Gamma, \Gamma', x_2{:}\tau_2 \vdash e_1[x_1 := v_1] : \tau'$

(h) By (g), (f), and Term Substitution: $\Gamma, \Gamma' \vdash e_1[x_1 := v_1][x_2 := v_2] : \tau'$

3. By assumption, (1), (2), and Exchange: $\Gamma \vdash LE[e_1[x_1 := v_1][x_2 := v_2]] : \tau$

- Case R-Open: $e = LE[\mathsf{let}\ \langle \alpha, x\rangle = \langle \tau_1, v\rangle{:}\tau'_1\ \mathsf{in}\ e_1]$ and $e' = LE[e_1[\alpha := \tau_1][x := v]]$:

1. By Context Elimination: $\Gamma, \Gamma' \vdash \mathsf{let}\ \langle \alpha, x\rangle = \langle \tau_1, v\rangle{:}\tau'_1\ \mathsf{in}\ e_1 : \tau'$ where $\Gamma \vdash L : \Gamma'$

2. We show: $\Gamma, \Gamma' \vdash e_1[x_1 := v_1][x_2 := v_2] : \tau'$

(a) By (1) and Typing Inversion: $\Gamma, \Gamma' \vdash \langle \tau_1, v\rangle{:}\tau'_1 : \exists\alpha{:}\kappa.\tau_2$ and $\Gamma, \Gamma', \alpha{:}\kappa, x{:}\tau_2 \vdash e_1 : \tau''$ where $\alpha \notin ftv(\tau'')$ and $\tau'' \equiv \tau'$

(b) By (a) and Typing Inversion: $\Gamma, \Gamma' \vdash \tau_1 : \kappa'$ and $\Gamma, \Gamma' \vdash v : \tau_2[\alpha := \tau_1]$ where $\tau'_1 = \exists\alpha{:}\kappa'.\tau'_2$ and $\exists\alpha{:}\kappa.\tau_2 \equiv \exists\alpha{:}\kappa'.\tau'_2$ and $\Gamma \vdash \tau'_1 : \Omega$

(c) By (b) and Type Equivalence Inversion: $\kappa = \kappa'$ and $\tau_2 \equiv \tau'_2$

(d) By (a), (b), (c), and Type Substitution: $\Gamma, \Gamma', x{:}\tau_2[\alpha := \tau_1] \vdash e_1[\alpha := \tau_1] : \tau''[\alpha := \tau_1]$

(e) By (d), (b), and Term Substitution: $\Gamma, \Gamma' \vdash e_1[\alpha := \tau_1][x := v] : \tau''[\alpha := \tau_1]$

(f) By (a) and (e): $\Gamma, \Gamma' \vdash e_1[\alpha := \tau_1][x := v] : \tau''$

(g) By (1) and Validity: $\Gamma, \Gamma' \vdash \tau' : \Omega$

(h) By (a), (f), (g), and T-Equiv: $\Gamma, \Gamma' \vdash e_1[\alpha := \tau_1][x := v] : \tau'$

3. By assumption, (1), (2), and Exchange: $\Gamma \vdash LE[e_1[\alpha := \kappa][x := v]] : \tau$

- Case R-Suspend: $e = LE[\mathsf{lazy}\ \langle \zeta, x\rangle = e_1\ \mathsf{in}\ e_2]$ and $e' = L[\mathsf{lazy}\ \langle \zeta, x\rangle = e_1\ \mathsf{in}\ E[e_2]]$ where $E \neq \_$:

1. By Context Elimination: $\Gamma, \Gamma' \vdash E[\mathsf{lazy}\ \langle \zeta, x\rangle = e_1\ \mathsf{in}\ e_2] : \tau'$ where $\Gamma \vdash L : \Gamma'$ (w.l.o.g. $\{\zeta, x\} \cap dom(\Gamma, \Gamma') = \emptyset$ and $\zeta \notin ftv(\tau')$)

2. We show: $\Gamma, \Gamma' \vdash \mathsf{lazy}\ \langle \zeta, x\rangle = e_1\ \mathsf{in}\ E[e_2] : \tau'$

(a) By (1) and Context Elimination: $\Gamma, \Gamma' \vdash \mathsf{lazy}\ \langle \zeta, x\rangle = e_1\ \mathsf{in}\ e_2 : \tau_2$

(b) By (a), (1), and Typing Inversion:

i. $\Gamma, \Gamma' \vdash e_1 : \exists\alpha{:}\kappa.\tau_1$

ii. $\Gamma, \Gamma', \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash e_2 : \tau_2'$

iii. $\zeta \notin ftv(\tau_2')$

iv. $\tau_2' \equiv \tau_2$

(c) By (b-ii) and Validity: $\Gamma, \Gamma', \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash \tau_2' : \Omega$

(d) By (d) and Variable Containment: $ftv(\tau_2') \subseteq dom(\Gamma, \Gamma', \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta])$

(e) By (b-iii) and (e): $ftv(\tau_2') \subseteq dom(\Gamma, \Gamma')$

(f) By (1) and Environment Validity: $\Gamma, \Gamma' \vdash \Box$

(g) By (d), (f), (g), and Strengthening: $\Gamma, \Gamma' \vdash \tau_2' : \Omega$

(h) By (a), (b-iv), (h), and T-Equiv: $\Gamma, \Gamma' \vdash \mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ e_2 : \tau_2'$

(i) By (1), (h), (b-ii), and Exchange: $\Gamma, \Gamma', \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash E[e_2] : \tau'$

(j) By (b-i), (i), (1), and T-Lazy: $\Gamma, \Gamma' \vdash \mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ E[e_2] : \tau'$

3. By assumption, (1), (2), and Exchange: $\Gamma \vdash L[\mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ E[e_2]] : \tau$

- Case R-Trigger: $e = L[\mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ L'ES[x]]$ and $e' = L[\mathsf{let}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ (L'ES[x])[\zeta := \alpha]]$

1. By assumption: $\Gamma \vdash L[\mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ L'ES[x]] : \tau$
2. By (1) and Context Elimination: $\Gamma, \Gamma' \vdash \mathsf{lazy}\ \langle\zeta, x\rangle = e_1\ \mathsf{in}\ L'ES[x] : \tau'$ where $\Gamma \vdash L : \Gamma'$
3. By (2) and Typing Inversion: $\Gamma, \Gamma' \vdash e_1 : \exists\alpha{:}\kappa.\tau_1$ and $\Gamma, \Gamma', \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash L'ES[x] : \tau''$ where $\zeta \notin ftv(\tau'')$ and $\tau'' \equiv \tau'$
4. We show: $\Gamma, \Gamma', \alpha{:}\kappa \vdash \alpha : \kappa$

(a) By (3) and Validity: $\Gamma, \Gamma' \vdash \exists\alpha{:}\kappa.\tau_1 : \Omega$

(b) By (a) and inversion of K-Exist: $\Gamma, \Gamma', \alpha{:}\kappa \vdash \tau_1 : \Omega$

(c) By (b) and Environment Validity: $\Gamma, \Gamma', \alpha{:}\kappa \vdash \Box$

(d) By (c): $\Gamma, \Gamma', \alpha{:}\kappa \vdash \alpha : \kappa$

5. We show: $\Gamma, \Gamma', \alpha{:}\kappa, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash L'ES[x] : \tau''$

(a) By (3) and Environment Validity: $\Gamma, \Gamma', \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash \Box$

(b) By (a) and inversion of E-Term: $\Gamma, \Gamma', \zeta{:}\kappa \vdash \tau_1[\alpha := \zeta] : \Omega$ and $x \notin dom(\Gamma, \Gamma', \zeta{:}\kappa)$

(c) By (b) and Environment Validity: $\Gamma, \Gamma', \zeta{:}\kappa \vdash \Box$

(d) By (c) and inversion of E-Type: $\Gamma, \Gamma' \vdash \Box$ and $\zeta \notin dom(\Gamma, \Gamma')$

(e) By (4c) and inversion of E-Type: $\alpha \notin dom(\Gamma, \Gamma')$

(f) By (d), (e), and E-Type: $\Gamma, \Gamma', \alpha{:}\kappa \vdash \Box$

(g) By (d): $\zeta \notin dom(\Gamma, \Gamma', \alpha{:}\kappa)$

(h) By (f), (g), and E-Type: $\Gamma, \Gamma', \alpha{:}\kappa, \zeta{:}\kappa \vdash \Box$

(i) By (b), (h), and Weakening: $\Gamma, \Gamma', \alpha{:}\kappa, \zeta{:}\kappa \vdash \tau_1[\alpha := \zeta] : \Omega$

(j) By (b): $x \notin dom(\Gamma, \Gamma', \alpha{:}\kappa, \zeta{:}\kappa)$

(k) By (i), (j), and E-Term: $\Gamma, \Gamma', \alpha{:}\kappa, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash \Box$

(l) By (3), (k), and Weakening: $\Gamma, \Gamma', \alpha{:}\kappa, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash L'ES[x] : \tau''$

6. By (4), (5), and Type Substitution: $\Gamma, \Gamma', \alpha{:}\kappa, x{:}\tau_1[\alpha := \zeta][\zeta := \alpha] \vdash (L'ES[x])[\zeta := \alpha] : \tau''[\zeta := \alpha]$
7. We show: $\zeta \notin ftv(\tau_1)$

(a) By (4b) and Variable Containment: $ftv(\tau_1) \subseteq dom(\Gamma, \Gamma', \alpha{:}\kappa)$

(b) By (5d): $\zeta \notin dom(\Gamma, \Gamma', \alpha{:}\kappa)$

(c) By (a) and (b): $\zeta \notin ftv(\tau_1)$

8. By (6) and (7): $\Gamma, \Gamma', \alpha{:}\kappa, x{:}\tau_1 \vdash (L'ES[x])[\zeta := \alpha] : \tau''[\zeta := \alpha]$
9. By (3) and (8): $\Gamma, \Gamma', \alpha{:}\kappa, x{:}\tau_1 \vdash (L'ES[x])[\zeta := \alpha] : \tau''$
10. We show: $\alpha \notin ftv(\tau'')$
    (a) By (3) and Variable Containment: $ftv(\tau'') \subseteq dom(\Gamma, \Gamma', \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta])$
    (b) By (5e): $\alpha \notin dom(\Gamma, \Gamma', \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta])$
    (c) By (a) and (b): $\alpha \notin ftv(\tau'')$
11. By (3), (9), (10), and T-Open: $\Gamma, \Gamma' \vdash \mathsf{let}\ \langle \alpha, x\rangle = e_1\ \mathsf{in}\ (L'ES[x])[\zeta := \alpha] : \tau''$
12. By (2) and Validity: $\Gamma, \Gamma' \vdash \tau' : \Omega$
13. By (3), (11), (12), and T-Equiv: $\Gamma, \Gamma' \vdash \mathsf{let}\ \langle \alpha, x\rangle = e_1\ \mathsf{in}\ (L'ES[x])[\zeta := \alpha] : \tau'$
14. By (1), (2), (13), and Exchange: $\Gamma \vdash L[\mathsf{let}\ \langle \alpha, x\rangle = e_1\ \mathsf{in}\ (L'ES[x])[\zeta := \alpha]] : \tau$

- Case R-Case1: $e = LE[\mathsf{tcase}\ v{:}\nu\ \mathsf{of}\ x{:}\nu\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]$ and $e' = LE[e_1[x := v]]$
  1. By Context Elimination: $\Gamma, \Gamma' \vdash \mathsf{tcase}\ v{:}\nu\ \mathsf{of}\ x{:}\nu\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau'$ where $\Gamma \vdash L : \Gamma'$ and w.l.o.g. $x \notin dom(\Gamma, \Gamma')$
  2. By (1) and Typing Inversion: $\Gamma, \Gamma' \vdash v : \nu$ and $\Gamma, \Gamma', x{:}\nu \vdash e_1 : \tau'$
  3. By (2) and Term Substitution: $\Gamma, \Gamma' \vdash e_1[x := v] : \tau'$
  4. By assumption, (1), (2), and Exchange: $\Gamma \vdash LE[e_1[x := v]] : \tau$

- Case R-Case2: $e = LE[\mathsf{tcase}\ v{:}\tau_1\ \mathsf{of}\ x{:}\tau_2\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]$ and $e' = LE[e_2]$
  1. By Context Elimination: $\Gamma, \Gamma' \vdash \mathsf{tcase}\ v{:}\tau_1\ \mathsf{of}\ x{:}\tau_2\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau'$ where $\Gamma \vdash L : \Gamma'$ and w.l.o.g. $x \notin dom(\Gamma, \Gamma')$
  2. By (1) and Typing Inversion: $\Gamma, \Gamma' \vdash e_2 : \tau'$
  3. By assumption, (1), (2), and Exchange: $\Gamma \vdash LE[e_2] : \tau$

- Case RT-App: $e = LECT[(\lambda\alpha{:}\kappa.\nu)\ \nu']$ and $e' = LECT[\nu[\alpha := \nu']]$:
  1. By Context Elimination: $\Gamma, \Gamma' \vdash CT[(\lambda\alpha{:}\kappa.\nu)\ \nu'] : \tau'$ where $\Gamma \vdash L : \Gamma'$ and w.l.o.g. $\alpha \notin dom(\Gamma, \Gamma')$
  2. We show: $\Gamma, \Gamma' \vdash CT[\nu[\alpha := \nu']] : \tau'$
     (a) By (1) and Type Context Elimination: $\Gamma, \Gamma' \vdash T[(\lambda\alpha{:}\kappa.\nu)\ \nu'] : \Omega$
     (b) By (a) and Type Context Elimination: $\Gamma, \Gamma', \Gamma'' \vdash (\lambda\alpha{:}\kappa.\nu)\ \nu' : \kappa'$ where $\Gamma, \Gamma' \vdash T : \Gamma''$
     (c) By (b) and inversion of K-App: $\Gamma, \Gamma', \Gamma'' \vdash \lambda\alpha{:}\kappa.\nu : \kappa'' \to \kappa'$ and $\Gamma, \Gamma', \Gamma'' \vdash \nu' : \kappa''$
     (d) By (c), (1), and inversion of K-Abs: $\Gamma, \Gamma', \Gamma'', \alpha{:}\kappa \vdash \nu : \kappa'$ and $\kappa'' = \kappa$
     (e) By (d), (c), and Type Substitution: $\Gamma, \Gamma', \Gamma'' \vdash \nu[\alpha := \nu'] : \kappa'$
     (f) By Q-Beta: $(\lambda\alpha{:}\kappa.\nu)\ \nu' \equiv \nu[\alpha := \nu']$
     (g) By (1), (b), (d), (e), and Type Exchange: $\Gamma, \Gamma' \vdash CT[\nu[\alpha := \nu']] : \tau'$
  3. By assumption, (1), (2), and Exchange: $\Gamma, \Gamma' \vdash LECT[\nu[\alpha := \nu']] : \tau$

- Case RT-Proj1: $e = LECT[\langle \nu_1, \nu_2\rangle.1]$ and $e' = LECT[\nu_1]$:
  1. By Context Elimination: $\Gamma, \Gamma' \vdash CT[\langle \nu_1, \nu_2\rangle.1] : \tau'$ where $\Gamma \vdash L : \Gamma'$
  2. We show: $\Gamma, \Gamma' \vdash CT[\nu_1] : \tau'$
     (a) By (1) and Type Context Elimination: $\Gamma, \Gamma' \vdash T[\langle \nu_1, \nu_2\rangle.1] : \Omega$

(b) By (a) and Type Context Elimination: $\Gamma, \Gamma', \Gamma'' \vdash \langle \nu_1, \nu_2 \rangle.1 : \kappa$ where $\Gamma, \Gamma' \vdash T : \Gamma''$

(c) By (b) and inversion of K-Proj1: $\Gamma, \Gamma', \Gamma'' \vdash \langle \nu_1, \nu_2 \rangle : \kappa \times \kappa_2$

(d) By (c) and inversion of K-Pair: $\Gamma, \Gamma', \Gamma'' \vdash \nu_1 : \kappa$

(e) By Q-Proj1b: $\langle \nu_1, \nu_2 \rangle.1 \equiv \nu_1$

(f) By (1), (b), (d), (e), and Type Exchange: $\Gamma, \Gamma' \vdash CT[\nu_1] : \tau'$

3. By assumption, (1), (2), and Exchange: $\Gamma \vdash LECT[\nu_1] : \tau$

- Case RT-Proj2: $e = LECT[\langle \nu_1, \nu_2 \rangle.2]$ and $e' = LECT[\nu_2]$:

1. By Context Elimination: $\Gamma, \Gamma' \vdash CT[\langle \nu_1, \nu_2 \rangle.2] : \tau'$ where $\Gamma \vdash L : \Gamma'$
2. We show: $\Gamma, \Gamma' \vdash CT[\nu_2] : \tau'$

(a) By (1) and Type Context Elimination: $\Gamma, \Gamma' \vdash T[\langle \nu_1, \nu_2 \rangle.2] : \Omega$

(b) By (a) and Type Context Elimination: $\Gamma, \Gamma', \Gamma'' \vdash \langle \nu_1, \nu_2 \rangle.2 : \kappa$ where $\Gamma, \Gamma' \vdash T : \Gamma''$

(c) By (b) and inversion of K-Proj2: $\Gamma, \Gamma', \Gamma'' \vdash \langle \nu_1, \nu_2 \rangle : \kappa_1 \times \kappa$

(d) By (c) and inversion of K-Pair: $\Gamma, \Gamma', \Gamma'' \vdash \nu_2 : \kappa$

(e) By Q-Proj2b: $\langle \nu_1, \nu_2 \rangle.2 \equiv \nu_2$

(f) By (1), (b), (d), (e), and Type Exchange: $\Gamma, \Gamma' \vdash CT[\nu_2] : \tau'$

3. By assumption, (1), (2), and Exchange: $\Gamma \vdash LECT[\nu_2] : \tau$

- Case RT-Trigger: $e = L[\mathsf{lazy}\ \langle \zeta, x \rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ L'ECT[\zeta]]$ and $e' = L[\mathsf{let}\ \langle \zeta, x \rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ (L'ECT[\zeta])[\zeta := \alpha]]$

1. By assumption: $\Gamma \vdash L[\mathsf{lazy}\ \langle \zeta, x \rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ L'ECT[\zeta]] : \tau$
2. By (1) and Context Elimination: $\Gamma, \Gamma' \vdash \mathsf{lazy}\ \langle \zeta, x \rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ L'ECT[\zeta] : \tau'$ where $\Gamma \vdash L : \Gamma'$
3. By (2) and Typing Inversion: $\Gamma, \Gamma' \vdash e_1 : \exists \alpha{:}\kappa.\tau_1$ and $\Gamma, \Gamma', \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash L'ECT[\zeta] : \tau''$ where $\zeta \notin ftv(\tau'')$ and $\tau'' \equiv \tau'$
4. We show: $\Gamma, \Gamma', \alpha{:}\kappa \vdash \alpha : \kappa$

(a) By (3) and Validity: $\Gamma, \Gamma' \vdash \exists \alpha{:}\kappa.\tau_1 : \Omega$

(b) By (a) and inversion of K-Exist: $\Gamma, \Gamma', \alpha{:}\kappa \vdash \tau_1 : \Omega$

(c) By (b) and Environment Validity: $\Gamma, \Gamma', \alpha{:}\kappa \vdash \Box$

(d) By (c): $\Gamma, \Gamma', \alpha{:}\kappa \vdash \alpha : \kappa$

5. We show: $\Gamma, \Gamma', \alpha{:}\kappa, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash L'ECT[\zeta] : \tau''$

(a) By (3) and Environment Validity: $\Gamma, \Gamma', \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash \Box$

(b) By (a) and inversion of E-Term: $\Gamma, \Gamma', \zeta{:}\kappa \vdash \tau_1[\alpha := \zeta] : \Omega$ and $x \notin dom(\Gamma, \Gamma', \zeta{:}\kappa)$

(c) By (b) and Environment Validity: $\Gamma, \Gamma', \zeta{:}\kappa \vdash \Box$

(d) By (c) and inversion of E-Type: $\Gamma, \Gamma' \vdash \Box$ and $\zeta \notin dom(\Gamma, \Gamma')$

(e) By (4c) and inversion of E-Type: $\alpha \notin dom(\Gamma, \Gamma')$

(f) By (d), (e), and E-Type: $\Gamma, \Gamma', \alpha{:}\kappa \vdash \Box$

(g) By (d): $\zeta \notin dom(\Gamma, \Gamma', \alpha{:}\kappa)$

(h) By (f), (g), and E-Type: $\Gamma, \Gamma', \alpha{:}\kappa, \zeta{:}\kappa \vdash \Box$

(i) By (b), (h), and Weakening: $\Gamma, \Gamma', \alpha{:}\kappa, \zeta{:}\kappa \vdash \tau_1[\alpha := \zeta] : \Omega$

(j) By (b): $x \notin dom(\Gamma, \Gamma', \alpha{:}\kappa, \zeta{:}\kappa)$

(k) By (i), (j), and E-Term: $\Gamma, \Gamma', \alpha{:}\kappa, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash \Box$

   (l) By (3), (k), and Weakening: $\Gamma, \Gamma', \alpha{:}\kappa, \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta] \vdash L'ECT[\zeta] : \tau''$

6. By (4), (5), and Type Substitution:
   $\Gamma, \Gamma', \alpha{:}\kappa, x{:}\tau_1[\alpha := \zeta][\zeta := \alpha] \vdash (L'ECT[\zeta])[\zeta := \alpha] : \tau''[\zeta := \alpha]$
7. We show: $\zeta \notin ftv(\tau_1)$
   (a) By (4b) and Variable Containment: $ftv(\tau_1) \subseteq dom(\Gamma, \Gamma', \alpha{:}\kappa)$
   (b) By (5d): $\zeta \notin dom(\Gamma, \Gamma', \alpha{:}\kappa)$
   (c) By (a) and (b): $\zeta \notin ftv(\tau_1)$
8. By (6) and (7): $\Gamma, \Gamma', \alpha{:}\kappa, x{:}\tau_1 \vdash (L'ECT[\zeta])[\zeta := \alpha] : \tau''[\zeta := \alpha]$
9. By (3) and (8): $\Gamma, \Gamma', \alpha{:}\kappa, x{:}\tau_1 \vdash (L'ECT[\zeta])[\zeta := \alpha] : \tau''$
10. We show: $\alpha \notin ftv(\tau'')$
   (a) By (3) and Variable Containment: $ftv(\tau'') \subseteq dom(\Gamma, \Gamma', \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta])$
   (b) By (5e): $\alpha \notin dom(\Gamma, \Gamma', \zeta{:}\kappa, x{:}\tau_1[\alpha := \zeta])$
   (c) By (a) and (b): $\alpha \notin ftv(\tau'')$
11. By (3), (9), (10), and T-Open: $\Gamma, \Gamma' \vdash \mathsf{let}\ \langle \alpha, x \rangle = e_1\ \mathsf{in}\ (L'ECT[\zeta])[\zeta := \alpha] : \tau''$
12. By (2) and Validity: $\Gamma, \Gamma' \vdash \tau' : \Omega$
13. By (3), (11), (12), and T-Equiv: $\Gamma, \Gamma' \vdash \mathsf{let}\ \langle \alpha, x \rangle = e_1\ \mathsf{in}\ (L'ECT[\zeta])[\zeta := \alpha] : \tau'$
14. By (1), (2), (13), and Exchange: $\Gamma \vdash L[\mathsf{let}\ \langle \alpha, x \rangle = e_1\ \mathsf{in}\ (L'ECT[\zeta])[\zeta := \alpha]] : \tau$

□

Lemma 13 (Lazy Type Variables). *If* $\Gamma \vdash LCT[\zeta] : \tau$ *and* $\zeta \notin dom(\Gamma)$*, then* $L = L_1[\mathsf{lazy}\ \langle \zeta, x \rangle = e\ \mathsf{in}\ L_2]$ *where* $\zeta \notin btv(L_2)$.

Proof. By structural induction on L. W.l.o.g. all bound type variables of $LCT[\zeta]$ are distinct and $(btv(LCT[\zeta]) \cup bv(LCT[\zeta])) \cap dom(\Gamma) = \emptyset$.

- Case $L = \_$:
  1. By assumption: $\Gamma \vdash CT[\zeta] : \tau'$
  2. By (1) and Type Context Elimination : $\Gamma \vdash T[\zeta] : \Omega$
  3. By (2) and Type Context Elimination: $\Gamma, \Gamma' \vdash \zeta : \Omega$ where $\Gamma \vdash T : \Gamma'$
  4. By (3) and Variable Containment: $\zeta \in dom(\Gamma, \Gamma')$
  5. By (3) and Type Context Elimination: $\zeta \notin dom(\Gamma')$
  6. By (4) and (5): $\zeta \in dom(\Gamma)$
  7. (6) contradicts the assumption $\zeta \notin dom(\Gamma)$, hence this case is not possible.
- Case $L = \mathsf{lazy}\ \langle \zeta', x \rangle = e\ \mathsf{in}\ L'$:
  - Subcase $\zeta' = \zeta \wedge \zeta \notin btv(L')$: this is what we claimed
  - Subcase $\zeta' = \zeta \wedge \zeta \in btv(L')$: not possible due to our assumption about bound type variables
  - Subcase $\zeta' \neq \zeta$:
    1. By assumption and Typing Inversion: $\Gamma, \zeta'{:}\kappa, x{:}\tau' \vdash L'CT[\zeta] : \tau''$
    2. By assumption: $\zeta \notin dom(\Gamma, \zeta'{:}\kappa, x{:}\tau'')$
    3. By (1), (2), and induction: $L' = L'_1[\mathsf{lazy}\ \langle \zeta, x' \rangle = e'\ \mathsf{in}\ L_2]$ where $\zeta \notin btv(L_2)$
    4. Let $L_1 = \mathsf{lazy}\ \langle \zeta', x \rangle = e\ \mathsf{in}\ L'_1$.

5. By (3) and (4): $L = L_1[\mathsf{lazy}\ \langle\zeta, x'\rangle\ \mathsf{=}\ e'\ \mathsf{in}\ L_2]$ where $\zeta \notin btv(L_2)$

□

Proposition 13 (Canonical Normal Forms). *Let $\Gamma \vdash \nu : \kappa$ where ν is not a path.*

- *If $\kappa = \kappa_1 \to \kappa_2$, then $\nu = \lambda\alpha{:}\kappa.\nu'$.*
- *If $\kappa = \kappa_1 \times \kappa_2$, then $\nu = \langle\nu_1, \nu_2\rangle$.*

Proof. Follows from the definition of the kinding relation.

- If $\kappa = \kappa_1 \to \kappa_2$, then the last rule of the derivation must be K-Abs. Hence: $\nu = \lambda\alpha{:}\kappa_1.\nu'$
- If $\kappa = \kappa_1 \to \kappa_2$, then the last rule of the derivation must be K-Pair. Hence: $\nu = \langle\nu_1, \nu_2\rangle$

□

Proposition 14 (Type Progress). *If $\cdot \vdash LCT[\tau] : \tau'$ and τ is not a normal form, then $LCT[\tau] \longrightarrow e$.*

Proof. By structural induction on τ.

- Case $\tau = \alpha$: not possible because α is a normal form
- Case $\tau = \zeta$:
  1. By Lazy Type Variables: $L = L_1[\mathsf{lazy}\ \langle\zeta, x\rangle\ \mathsf{=}\ e\ \mathsf{in}\ L_2]$ where $\zeta \notin btv(L_2)$
  2. By (1) and RT-Trigger: $LCT[\zeta] \longrightarrow L_1[\mathsf{let}\ \langle\alpha, x\rangle\ \mathsf{=}\ e\ \mathsf{in}\ (L_2CT[\zeta])[\zeta := \alpha]]$
- Case $\tau = \tau_1 \to \tau_2$:
  - Subcase τ_1 is not a normal form:
    1. Let $T' = T[\_ \to \tau_2]$. Then: $LCT[\tau] = LCT'[\tau_1]$
    2. By (1) and induction: $LCT[\tau] \longrightarrow e$
  - Subcase $\tau_1 = \nu_1$:
    - Subcase τ_2 is not a normal form:
      1. Let $T' = T[\nu_1 \to \_]$. Then: $LCT[\tau] = LCT'[\tau_2]$
      2. By (1) and induction: $LCT[\tau] \longrightarrow e$
    - Subcase $\tau_2 = \nu_2$: not possible since $\nu_1 \to \nu_2$ is a normal form
- Case $\tau = \tau_1 \times \tau_2$:
  - Subcase τ_1 is not a normal form:
    1. Let $T' = T[\_ \times \tau_2]$. Then: $LCT[\tau] = LCT'[\tau_1]$
    2. By (1) and induction: $LCT[\tau] \longrightarrow e$
  - Subcase $\tau_1 = \nu_1$:
    - Subcase τ_2 is not a normal form:
      1. Let $T' = T[\nu_1 \times \_]$. Then: $LCT[\tau] = LCT'[\tau_2]$
      2. By (1) and induction: $LCT[\tau] \longrightarrow e$
    - Subcase $\tau_2 = \nu_2$: not possible since $\nu_1 \times \nu_2$ is a normal form
- Case $\tau = \exists\alpha{:}\kappa.\tau_1$:
  - Subcase τ_1 is not a normal form:

   1. Let $T' = T[\exists\alpha{:}\kappa.\_]$. Then: $LCT[\tau] = LCT'[\tau_1]$
   2. By (1) and induction: $LCT[\tau] \longrightarrow e$
  - Subcase $\tau_1 = \nu_1$: not possible since $\exists\alpha{:}\kappa.\nu_1$ is a normal form

- Case $\tau = \forall\alpha{:}\kappa.\tau_1$:
  - Subcase τ_1 is not a normal form:
    1. Let $T' = T[\forall\alpha{:}\kappa.\_]$. Then: $LCT[\tau] = LCT'[\tau_1]$
    2. By (1) and induction: $LCT[\tau] \longrightarrow e$
  - Subcase $\tau_1 = \nu_1$: not possible since $\forall\alpha{:}\kappa.\nu_1$ is a normal form

- Case $\tau = \lambda\alpha{:}\kappa.\tau_1$:
  - Subcase τ_1 is not a normal form:
    1. Let $T' = T[\lambda\alpha{:}\kappa.\_]$. Then: $LCT[\tau] = LCT'[\tau_1]$
    2. By (1) and induction: $LCT[\tau] \longrightarrow e$
  - Subcase $\tau_1 = \nu_1$: not possible since $\lambda\alpha{:}\kappa.\nu_1$ is a normal form

- Case $\tau = \tau_1\ \tau_2$:
  - Subcase τ_1 is not a normal form:
    1. Let $T' = T[\_\ \tau_2]$. Then: $LCT[\tau] = LCT'[\tau_1]$
    2. By (1) and induction: $LCT[\tau] \longrightarrow e$
  - Subcase $\tau_1 = \nu_1$:
    * Subsubcase τ_2 is not a normal form:
      1. Let $T' = T[\nu_1\ \_]$. Then: $LCT[\tau] = LCT'[\tau_2]$
      2. By (1) and induction: $LCT[\tau] \longrightarrow e$
    * Subsubcase $\tau_2 = \nu_2$:
      · Subsubsubcase ν_1 is not a path:
        1. By assumption and Context Elimination: $\Gamma \vdash CT[\nu_1\ \nu_2] : \tau''$
        2. By (1) and Type Context Elimination: $\Gamma' \vdash T[\nu_1\ \nu_2] : \Omega$
        3. By (2) and Type Context Elimination: $\Gamma, \Gamma' \vdash \nu_1\ \nu_2 : \kappa$
        4. By (3) and inversion of K-App: $\Gamma, \Gamma' \vdash \nu_1 : \kappa' \to \kappa$
        5. By (4) and Canonical Normal Forms: $\nu_1 = \lambda\alpha{:}\kappa'.\nu$
        6. By (5) and RT-App: $LCT[\tau] = LCT[(\lambda\alpha{:}\kappa'.\nu)\ \nu_2] \longrightarrow LCT[\nu[\alpha := \nu_2]]$

      · Subsubsubcase $\nu_1 = p$: not possible since $p\ \nu_2$ is a normal form

- Case $\tau = \langle\tau_1, \tau_2\rangle$:
  - Subcase τ_1 is not a normal form:
    1. Let $T' = T[\langle\_, \tau_2\rangle]$. Then: $LCT[\tau] = LCT'[\tau_1]$
    2. By (1) and induction: $LCT[\tau] \longrightarrow e$
  - Subcase $\tau_1 = \nu_1$:
    * Subcase τ_2 is not a normal form:
      1. Let $T' = T[\langle\nu_1, \_\rangle]$. Then: $LCT[\tau] = LCT'[\tau_2]$
      2. By (1) and induction: $LCT[\tau] \longrightarrow e$
    * Subcase $\tau_2 = \nu_2$: not possible since $\langle\nu_1, \nu_2\rangle$ is a normal form

- Case $\tau = \tau_1.1$:
  - Subcase τ_1 is not a normal form:
    1. Let $T' = T[\_.1]$. Then: $LCT[\tau] = LCT'[\tau_1]$
    2. By (1) and induction: $LCT[\tau] \longrightarrow e$
  - Subcase $\tau_1 = \nu_1$:
    - Subsubcase ν_1 is not a path:
      1. By assumption and Context Elimination: $\Gamma \vdash CT[\nu_1.1] : \tau''$
      2. By (1) and Type Context Elimination: $\Gamma' \vdash T[\nu_1.1] : \Omega$
      3. By (2) and Type Context Elimination: $\Gamma, \Gamma' \vdash \nu_1.1 : \kappa$
      4. By (3) and inversion of K-Proj1: $\Gamma, \Gamma' \vdash \nu_1 : \kappa \times \kappa_2$
      5. By (4) and Canonical Normal Forms: $\nu_1 = \langle \nu_{11}, \nu_{12} \rangle$
      6. By (5) and RT-Proj1: $LCT[\tau] = LCT[\langle \nu_{11}, \nu_{12} \rangle.1] \longrightarrow LCT[\nu_{11}]$
    - Subsubcase $\nu_1 = p$: not possible since $p.1$ is a normal form
- Case $\tau = \tau_1.2$:
  - Subcase τ_1 is not a normal form:
    1. Let $T' = T[\_.2]$. Then: $LCT[\tau] = LCT'[\tau_1]$
    2. By (1) and induction: $LCT[\tau] \longrightarrow e$
  - Subcase $\tau_1 = \nu_1$:
    - Subsubcase ν_1 is not a path:
      1. By assumption and Context Elimination: $\Gamma \vdash CT[\nu_1.2] : \tau''$
      2. By (1) and Type Context Elimination: $\Gamma' \vdash T[\nu_1.2] : \Omega$
      3. By (2) and Type Context Elimination: $\Gamma, \Gamma' \vdash \nu_1.2 : \kappa$
      4. By (3) and inversion of K-Proj2: $\Gamma, \Gamma' \vdash \nu_1 : \kappa_1 \times \kappa$
      5. By (4) and Canonical Normal Forms: $\nu_1 = \langle \nu_{11}, \nu_{12} \rangle$
      6. By (5) and RT-Proj2: $LCT[\tau] = LCT[\langle \nu_{11}, \nu_{12} \rangle.2] \longrightarrow LCT[\nu_{12}]$
    - Subsubcase $\nu_1 = p$: not possible since $p.2$ is a normal form

□

Lemma 14 (Context Extension)**.** *If $L[e] \longrightarrow e'$ and e is not a* lazy *expression, then for all E exists an e'' such that $LE[e] \longrightarrow e''$.*

Proof. By case analysis on the applied reduction rule.

- Case R-App: $e = E'[(\lambda x{:}\tau.e_1)v]$ and $e' = LE'[e_1[x := v]]$
  1. Let $E'' = E[E']$. Then: $LE[e] = LE''[(\lambda x{:}\tau.e_1)v]$
  2. By (1) and R-App: $LE[e] \longrightarrow LE''[e_1[x := v]]$
- Case R-Inst: $e = E'[(\lambda \alpha{:}\kappa.e_1)\tau]$ and $e' = LE'[e_1[\alpha := \tau]]$
  1. Let $E'' = E[E']$. Then: $LE[e] = LE''[(\lambda \alpha{:}\kappa.e_1)v]$
  2. By (1) and R-Inst: $LE[e] \longrightarrow LE''[e_1[\alpha := \tau]]$
- Case R-Proj: $e = E'[\mathsf{let}\ \langle x_1, x_2 \rangle = \langle v_1, v_2 \rangle\ \mathsf{in}\ e_1]$ and $e' = LE'[e_1[x_1 := v_1][x_2 := v_2]]$
  1. Let $E'' = E[E']$. Then: $LE[e] = LE''[\mathsf{let}\ \langle x_1, x_2 \rangle = \langle v_1, v_2 \rangle\ \mathsf{in}\ e_1]$

2. By (1) and R-Proj: $LE[e] \longrightarrow LE''[e_1[x_1 := v_1][x_2 := v_2]]$

- Case R-Open: $e = E'[\mathsf{let}\ \langle\alpha, x\rangle \texttt{ = } \langle\tau, v\rangle{:}\tau'\ \mathsf{in}\ e_1]$ and $e' = LE'[e_1[\alpha := \tau][x := v]]$
  1. Let $E'' = E[E']$. Then: $LE[e] = LE''[\mathsf{let}\ \langle\alpha, x\rangle \texttt{ = } \langle\tau, v\rangle\ \mathsf{in}\ e_1]$
  2. By (1) and R-Open: $LE[e] \longrightarrow LE''[e_1[\alpha := \tau][x := v]]$

- Case R-Suspend: $e = E'[\mathsf{lazy}\ \langle\zeta, x\rangle \texttt{ = } e_1\ \mathsf{in}\ e_2]$ and $e' = L[\mathsf{lazy}\ \langle\zeta, x\rangle \texttt{ = } e_1\ \mathsf{in}\ E'[e_2]]$ where $E' \neq \_$ and $\zeta \notin ftv(E')$ and $x \notin fv(E')$ and w.l.o.g. $\zeta \notin ftv(E)$ and $x \notin fv(E)$
  1. Let $E'' = E[E']$. Then: $LE[e] = LE''[\mathsf{lazy}\ \langle\zeta, x\rangle \texttt{ = } e_1\ \mathsf{in}\ e_2]$ and $E'' \neq \_$
  2. By assumption: $E'' \neq \_$ and $\zeta \notin ftv(E'')$ and $x \notin fv(E'')$
  3. By (1), (2), and R-Suspend: $LE[e] \longrightarrow L[\mathsf{lazy}\ \langle\zeta, x\rangle \texttt{ = } e_1\ \mathsf{in}\ E''[e_2]]$

- Case R-Trigger: $L = L_1[\mathsf{lazy}\ \langle\zeta, x\rangle \texttt{ = } e_1\ \mathsf{in}\ L_2]$ and $e = E'S[x]$ and $e' = L_1[\mathsf{let}\ \langle\alpha, x\rangle \texttt{ = } e\ \mathsf{in}\ (L_2E'S[x])[\zeta := \alpha]]$ where $x \notin bv(L_2)$
  1. Let $E'' = E[E']$. Then: $LE[e] = LE''S[x]$
  2. By (1), (2), and R-Trigger: $LE[e] \longrightarrow L_1[\mathsf{let}\ \langle\beta, x\rangle \texttt{ = } e\ \mathsf{in}\ (L_2E''S[x])[\zeta := \beta]]$

- Case R-Case1: $e = E'[\mathsf{tcase}\ v{:}\nu\ \mathsf{of}\ x{:}\nu\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]$ and $e' = LE'[e_1[x := v]]$
  1. Let $E'' = E[E']$. Then: $LE[e] = LE''[\mathsf{tcase}\ v{:}\nu\ \mathsf{of}\ x{:}\nu\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]$
  2. By (1) and R-Case1: $LE[e] \longrightarrow LE''[e_1[x := v]]$

- Case R-Case2: $e = E'[\mathsf{tcase}\ v{:}\nu\ \mathsf{of}\ x{:}\nu'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]$ and $e' = LE'[e_2]$ where $\nu \neq \nu'$
  1. Let $E'' = E[E']$. Then: $LE[e] = LE''[\mathsf{tcase}\ v{:}\nu\ \mathsf{of}\ x{:}\nu\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]$
  2. By (1) and R-Case2: $LE[e] \longrightarrow LE''[e_1[x := v]]$

- Case RT-App: $e = E'CT[(\lambda\alpha{:}\kappa.\nu)\ \nu']$ and $e' = E'CT[\nu[\alpha := \nu']]$
  1. Let $E'' = E[E']$. Then: $LE[e] = LE''CT[(\lambda\alpha{:}\kappa.\nu)\ \nu']$
  2. By (1) and RT-App: $LE[e] \longrightarrow LE''CT[\nu[\alpha := \nu']]$

- Case RT-Proj1: $e = E'CT[\langle\nu_1, \nu_2\rangle.1]$ and $e' = E'CT[\nu_1]$
  1. Let $E'' = E[E']$. Then: $LE[e] = LE''CT[\langle\nu_1, \nu_2\rangle.1]$
  2. By (1) and RT-Proj1: $LE[e] \longrightarrow LE''CT[\nu_1]$

- Case RT-Proj2: $e = E'CT[\langle\nu_1, \nu_2\rangle.2]$ and $e' = E'CT[\nu_2]$
  1. Let $E'' = E[E']$. Then: $LE[e] = LE''CT[\langle\nu_1, \nu_2\rangle.2]$
  2. By (1) and RT-Proj2: $LE[e] \longrightarrow LE''CT[\nu_2]$

- Case RT-Trigger: $L = L_1[\mathsf{lazy}\ \langle\zeta, x\rangle \texttt{ = } e_1\ \mathsf{in}\ L_2]$ and $e = E'CT[\zeta]$, $e' = L_1[\mathsf{let}\ \langle\zeta, x\rangle \texttt{ = } e_1\ \mathsf{in}\ (L_2E'CT[\zeta])[\zeta := \alpha]]$ where $\zeta \notin btv(L_2)$
  1. Let $E'' = E[E']$. Then: $LE[e] = LE''CT[\zeta]$
  2. By (1) and RT-Trigger: $LE[e] \longrightarrow L_1[\mathsf{let}\ \langle\zeta, x\rangle \texttt{ = } e_1\ \mathsf{in}\ (L_2E''CT[\zeta])[\zeta := \alpha]]$

□

Theorem 2 (Progress). *If $\cdot \vdash L[e] : \tau$ where e is neither a value nor a* lazy *expression, then $L[e] \longrightarrow e'$.*

Proof. By structural induction on e.

- Case $e = E[\mathsf{lazy}\ \langle\zeta, x\rangle \texttt{=} e_1\ \mathsf{in}\ e_2]$ (w.l.o.g. $\zeta \notin ftv(E)$ and $x \notin fv(E)$):
  1. By assumption: $E \neq \_$
  2. By (1) and R-Suspend: $L[e] \longrightarrow L[\mathsf{lazy}\ \langle\zeta, x\rangle \texttt{=} e_1\ \mathsf{in}\ E[e_2]]$
- Case $e \neq E[\mathsf{lazy} \ldots] \wedge e = x$: not possible (variables are values)
- Case $e \neq E[\mathsf{lazy} \ldots] \wedge e = \lambda x{:}\tau'.e'$: not possible (abstractions are values)
- Case $e \neq E[\mathsf{lazy} \ldots] \wedge e = e_1\ e_2$:
  - Subcase e_1 is not a value:
    1. Let $E_1 = \_\ e_2$.
    2. By (1) and assumption: $\cdot \vdash LE_1[e_1] : \tau$
    3. By (2) and Context Elimination: $\cdot \vdash L[e_1] : \tau_1'$
    4. By assumption: e_1 is not a lazy expression
    5. By (3), (4), and induction: $L[e_1] \longrightarrow e_1'$
    6. By (4), (5), and Context Extension: $L[e] = LE_1[e_1] \longrightarrow e'$
  - Subcase $e_1 = v_1$:
    - ∗ Subsubcase v_1 is not a variable:
      1. By assumption and Context Elimination: $\Gamma \vdash v_1\ e_2 : \tau'$ where $\cdot \vdash L : \Gamma$
      2. By (1) and Typing Inversion: $\Gamma \vdash v_1 : \tau_2 \to \tau'$
      3. By (2) and Canonical Values: $v_1 = \lambda x{:}\tau_2'.e'$
      4. By (3) and R-App: $L[e] = L[(\lambda x{:}\tau_2.e')\ v_2] \longrightarrow L[e'[x := v_2]]$
    - ∗ Subsubcase $v_1 = x$:
      1. Let $S = \_\ e_2$ and $C = S$. Then: $e = S[x] = C[x]$
      2. By (1) and assumption: $\cdot \vdash LC[x] : \tau$
      3. By (2) and Lazy Term Variables: $L = L_1[\mathsf{lazy}\ \langle\zeta, x\rangle \texttt{=} e'\ \mathsf{in}\ L_2]$ where $x \notin bv(L_2)$
      4. By (1), (3), and R-Trigger: $L[e] = L_1[\mathsf{lazy}\ \langle\zeta, x\rangle \texttt{=} e'\ \mathsf{in}\ L_2S[x]] \longrightarrow L_1[\mathsf{let}\ \langle\alpha, x\rangle \texttt{=} e'\ \mathsf{in}\ (L_2S[x])[\zeta := \alpha]]$
- Case $e \neq E[\mathsf{lazy} \ldots] \wedge e = \langle e_1, e_2\rangle$:
  - Subcase e_1 is not a value:
    1. Let $E_1 = \langle\_, e_2\rangle$.
    2. By (1) and assumption: $\cdot \vdash LE_1[e_1] : \tau$
    3. By (2) and Context Elimination: $\cdot \vdash L[e_1] : \tau_1'$
    4. By assumption: e_1 is not a lazy expression
    5. By (3), (4), and induction: $L[e_1] \longrightarrow e_1'$
    6. By (4), (5), and Context Extension: $L[e] = LE_1[e_1] \longrightarrow e'$
  - Subcase $e_1 = v_1$:
    - ∗ Subsubcase e_2 is not a value:
      1. Let $E_2 = \langle v_1, \_\rangle$.
      2. By (1) and assumption: $\cdot \vdash LE_2[e_2] : \tau$
      3. By (2) and Context Elimination: $\cdot \vdash L[e_2] : \tau_2'$

4. By assumption: e_2 is not a lazy expression
5. By (3), (4), and induction: $L[e_2] \longrightarrow e_2'$
6. By (4), (5), and Context Extension: $L[e] = LE_2[e_2] \longrightarrow e'$

∗ Subsubcase $e_2 = v_2$: not possible ($\langle v_1, v_2\rangle$ is a value)

- Case $e \neq E[\mathsf{lazy}\ \ldots] \wedge e = \lambda\alpha{:}\kappa.e'$: Type abstractions are values.
- Case $e \neq E[\mathsf{lazy}\ \ldots] \wedge e = e_1\ \tau'$:
  - Subcase e_1 is not a value:
    1. Let $E_1 = \_\ \tau'$.
    2. By (1) and assumption: $\cdot \vdash LE_1[e_1] : \tau$
    3. By (2) and Context Elimination: $\cdot \vdash L[e_1] : \tau_1'$
    4. By assumption: e' is not a lazy expression
    5. By (3), (4), and induction: $L[e_1] \longrightarrow e_1'$
    6. By (4), (5), and Context Extension: $L[e] = LE_1[e_1] \longrightarrow e'$
  - Subcase $e_1 = v_1$:
    - ∗ Subsubcase e_1 is not a variable:
      1. By assumption and Context Elimination: $\Gamma \vdash e_1\ \tau' : \tau''$
      2. By (1) and Typing Inversion: $\Gamma \vdash e_1 : \forall\alpha{:}\kappa.\tau_1$
      3. By (2) and Canonical Values: $e_1 = \lambda\alpha{:}\kappa.e_2$
      4. By (3) and R-INST: $L[e] = L[(\lambda\alpha{:}\kappa.e_2)\ \tau')] \longrightarrow L[e_2[\alpha := \tau']]$
    - ∗ Subsubcase $e_1 = x$:
      1. Let $S = \_\ \tau'$ and $C = S$. Then: $e = S[x] = C[x]$
      2. By (1) and assumption: $\cdot \vdash LC[x] : \tau$
      3. By (2) and Lazy Term Variables: $L = L_1[\mathsf{lazy}\ \langle\zeta, x\rangle\ \mathsf{=}\ e'\ \mathsf{in}\ L_2]$ where $x \notin bv(L_2)$
      4. By (1), (3), and R-TRIGGER: $L[e] = L_1[\mathsf{lazy}\ \langle\zeta, x\rangle\ \mathsf{=}\ e'\ \mathsf{in}\ L_2S[x]] \longrightarrow L_1[\mathsf{let}\ \langle\alpha, x\rangle\ \mathsf{=}\ e'\ \mathsf{in}\ (L_2S[x])[\zeta := \alpha]]$
- Case $e \neq E[\mathsf{lazy}\ \ldots] \wedge e = \langle\tau_1, e_1\rangle{:}\tau_2$
  - Subcase e_1 is not a value:
    1. Let $E_1 = \langle\tau_1, \_\rangle{:}\tau_2$.
    2. By (1) and assumption: $\cdot \vdash LE_1[e_1] : \tau$
    3. By (2) and Context Elimination: $\cdot \vdash L[e_1] : \tau_1'$
    4. By assumption: e_1 is not a lazy expression
    5. By (3), (4), and induction: $L[e_1] \longrightarrow e_1'$
    6. By (4), (5), and Context Extension: $L[e] = LE_1[e_1] \longrightarrow e'$
  - Subcase $e_1 = v_1$: not possible ($\langle\tau_1, v_1\rangle{:}\tau_2$ is a value)
  - $e = \langle\tau_1, e_1\rangle{:}\tau_2$
- Case $e \neq E[\mathsf{lazy}\ \ldots] \wedge e = \mathsf{let}\ \langle x_1, x_2\rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ e_2$
  - Subcase e_1 is not a value:
    1. Let $E_1 = \mathsf{let}\ \langle x_1, x_2\rangle\ \mathsf{=}\ \_\ \mathsf{in}\ e_2$.
    2. By (1) and assumption: $\cdot \vdash LE_1[e_1] : \tau$
    3. By (2) and Context Elimination: $\cdot \vdash L[e_1] : \tau_1'$

4. By assumption: e_1 is not a lazy expression
5. By (3), (4), and induction: $L[e_1] \longrightarrow e_1'$
6. By (4), (5), and Context Extension: $L[e] = LE_1[e_1] \longrightarrow e'$

- Subcase $e_1 = v$:
  - Subsubcase v is not a variable:
    1. By assumption and Context Elimination: $\Gamma \vdash \mathsf{let}\ \langle x_1, x_2\rangle\ \mathsf{=}\ v\ \mathsf{in}\ e_2 : \tau'$
    2. By (1) and Typing Inversion: $\Gamma \vdash v : \tau_1 \times \tau_2$
    3. By (2) and Canonical Values: $v = \langle v_1, v_2\rangle$
    4. By (3) and R-Proj: $L[e] = L[\mathsf{let}\ \langle x_1, x_2\rangle\ \mathsf{=}\ \langle v_1, v_2\rangle\ \mathsf{in}\ e_2] \longrightarrow L[e_2[x_1 := v_1][x_2 := v_2]]$
  - Subsubcase $v = x$:
    1. Let $S = \mathsf{let}\ \langle x_1, x_2\rangle\ \mathsf{=}\ \_\ \mathsf{in}\ e_2$ and $C = S$. Then: $e = S[x] = C[x]$
    2. By (1) and assumption: $\cdot \vdash LC[x] : \tau$
    3. By (2) and Lazy Term Variables: $L = L_1[\mathsf{lazy}\ \langle \zeta, x\rangle\ \mathsf{=}\ e'\ \mathsf{in}\ L_2]$ where $x \notin bv(L_2)$
    4. By (1), (3), and R-Trigger: $L[e] = L_1[\mathsf{lazy}\ \langle \zeta, x\rangle\ \mathsf{=}\ e'\ \mathsf{in}\ L_2S[x]] \longrightarrow L_1[\mathsf{let}\ \langle \alpha, x\rangle\ \mathsf{=}\ e'\ \mathsf{in}\ (L_2S[x])[\zeta := \alpha]]$

- Case $e \neq E[\mathsf{lazy}\ \ldots] \wedge e = \mathsf{let}\ \langle \alpha, x\rangle\ \mathsf{=}\ e_1\ \mathsf{in}\ e_2$
  - Subcase e_1 is not a value:
    1. Let $E_1 = \mathsf{let}\ \langle \alpha, x\rangle\ \mathsf{=}\ \_\ \mathsf{in}\ e_2$.
    2. By (1) and assumption: $\cdot \vdash LE_1[e_1] : \tau$
    3. By (2) and Context Elimination: $\cdot \vdash L[e_1] : \tau_1'$
    4. By assumption: e_1 is not a lazy expression
    5. By (3), (4), and induction: $L[e_1] \longrightarrow e_1'$
    6. By (4), (5), and Context Extension: $L[e] = LE_1[e_1] \longrightarrow e'$
  - Subcase $e_1 = v$:
    - Subsubcase v is not a variable:
      1. By assumption and Context Elimination: $\Gamma \vdash \mathsf{let}\ \langle \alpha, x\rangle\ \mathsf{=}\ v\ \mathsf{in}\ e_2 : \tau'$
      2. By (1) and Typing Inversion: $\Gamma \vdash v : \exists\alpha{:}\kappa.\tau_1$
      3. By (2) and Canonical Values: $v = \langle \tau_2, v'\rangle{:}\tau_2'$
      4. By (3) and R-Open: $L[e] = L[\mathsf{let}\ \langle \alpha, x\rangle\ \mathsf{=}\ \langle \tau_2, v'\rangle{:}\tau_2'\ \mathsf{in}\ e_2] \longrightarrow L[e_2[\alpha := \tau_2][x := v']]$
    - Subsubcase $v = x'$:
      1. Let $S = \mathsf{let}\ \langle \alpha, x\rangle\ \mathsf{=}\ \_\ \mathsf{in}\ e_2$ and $C = S$. Then: $e = S[x'] = C[x']$
      2. By (1) and assumption: $\cdot \vdash LC[x'] : \tau$
      3. By (2) and Lazy Term Variables: $L = L_1[\mathsf{lazy}\ \langle \zeta, x'\rangle\ \mathsf{=}\ e'\ \mathsf{in}\ L_2]$ where $x' \notin bv(L_2)$
      4. By (1), (3), and R-Trigger: $L[e] = L_1[\mathsf{lazy}\ \langle \zeta, x'\rangle\ \mathsf{=}\ e'\ \mathsf{in}\ L_2S[x']] \longrightarrow L_1[\mathsf{let}\ \langle \alpha, x\rangle\ \mathsf{=}\ e'\ \mathsf{in}\ (L_2S[x'])[\zeta := \alpha]]$

- Case $e \neq E[\mathsf{lazy}\ \ldots] \wedge e = \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$:
  - Subcase e_0 is not a value:
    1. Let $E_0 = \mathsf{tcase}\ \_{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$.
    2. By (1) and assumption: $\cdot \vdash LE_0[e_0] : \tau$

3. By (2) and Context Elimination: $\cdot \vdash L[e_0] : \tau_0''$
4. By assumption: e_0 is not a lazy expression
5. By (3), (4), and induction: $L[e_0] \longrightarrow e_0'$
6. By (4), (5), and Context Extension: $L[e] = LE_0[e_0] \longrightarrow e'$

– Subcase $e_0 = v$ and τ_0 is not a normal form:
  1. Let $C = \mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$. Then: $L[e] = LC[\tau_0]$
  2. By (1) and Type Progress: $L[e] \longrightarrow e'$

– Subcase $e_0 = v$ and $\tau_0 = \nu$:
  * Subsubcase τ_0' is not a normal form:
    1. Let $C = \mathsf{tcase}\ v{:}\tau_0\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$. Then: $L[e] = LC[\tau_0']$
    2. By (1) and Type Progress: $L[e] \longrightarrow e'$
  * Subsubcase $\tau_0' = \nu$:
    1. By R-Case1: $L[e] = L[\mathsf{tcase}\ v{:}\nu\ \mathsf{of}\ x{:}\nu\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2] \longrightarrow L[e_1[x := v]]$
  * Subsubcase $\tau_0' = \nu' \neq \nu$:
    1. By R-Case2: $L[e] = L[\mathsf{tcase}\ v{:}\nu\ \mathsf{of}\ x{:}\nu'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2] \longrightarrow L[e_2]$

□

A.3 Interleaved call-by-name reduction to weak head normal form

Proposition 15 (Uniqueness of Environments)**.**

1. If $\Gamma \vdash B : \Gamma_1$ *and* $\Gamma \vdash B : \Gamma_2$*, then* $\Gamma_1 = \Gamma_2$*.*

2. If $\Gamma \vdash P : \Gamma_1$ *and* $\Gamma \vdash P : \Gamma_2$*, then* $\Gamma_1 = \Gamma_2$*.*

Proof. Simultaneous, by induction on the generation of B and P.

1. Let $\Gamma \vdash B : \Gamma_1$ and $\Gamma \vdash B : \Gamma_2$.
   - Case $B = \mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$:
     (a) By inversion of B-Case (1st derivation): $\Gamma_1 = \cdot$
     (b) By inversion of B-Case (2nd derivation): $\Gamma_2 = \cdot$
   - Case $B = B'[\_ \to \tau_1][\_ \to \tau_2]$:
     (a) By inversion of B-Arrow1 (1st derivation): $\Gamma \vdash B' : \Gamma_1$
     (b) By inversion of B-Arrow1 (2nd derivation): $\Gamma \vdash B' : \Gamma_2$
     (c) By (a), (b), and induction: $\Gamma_1 = \Gamma_2$
   - Case $B = B'[\nu \to \_][\nu \to \_]$:
     (a) By inversion of B-Arrow2 (1st derivation): $\Gamma \vdash B' : \Gamma_1$
     (b) By inversion of B-Arrow2 (2nd derivation): $\Gamma \vdash B' : \Gamma_2$
     (c) By (a), (b), and induction: $\Gamma_1 = \Gamma_2$
   - Case $B = B'[\_ \times \tau_1][\_ \times \tau_2]$:
     (a) By inversion of B-Times1 (1st derivation): $\Gamma \vdash B' : \Gamma_1$
     (b) By inversion of B-Times1 (2nd derivation): $\Gamma \vdash B' : \Gamma_2$
     (c) By (a), (b), and induction: $\Gamma_1 = \Gamma_2$

$$
\begin{array}{lrl}
\text{weak head normal forms} & \omega ::= & q \mid \tau_1 \to \tau_2 \mid \tau_1 \times \tau_2 \mid \forall\alpha{:}\kappa.\tau \mid \exists\alpha{:}\kappa.\tau \mid \lambda\alpha{:}\kappa.\tau \mid \langle\tau_1,\tau_2\rangle \\
\text{weak head normal paths} & q ::= & \alpha \mid q\ \tau \mid q.1 \mid q.2 \\
\\
\text{regular comparison contexts} & B ::= & \mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mid \\
 & & B[\_ \to \tau_1][\_ \to \tau_2] \mid B[\nu \to \_][\nu \to \_] \mid \\
 & & B[\_ \times \tau_1][\_ \times \tau_2] \mid B[\nu \times \_][\nu \times \_] \mid \\
 & & B[\forall\alpha{:}\kappa.\_][\forall\alpha{:}\kappa.\_] \mid B[\exists\alpha{:}\kappa.\_][\exists\alpha{:}\kappa.\_] \mid \\
 & & B[\lambda\alpha{:}\kappa.\_][\lambda\alpha{:}\kappa.\_] \mid P[p\ \_][p\ \_] \mid \\
 & & B[\langle\_,\tau_1\rangle][\langle\_,\tau_2\rangle] \mid B[\langle\nu,\_\rangle][\langle\nu,\_\rangle] \\
\text{path comparison contexts} & P ::= & B \mid P[\_\ \tau_1][\_\ \tau_2] \mid P[\_.1][\_.1] \mid P[\_.2][\_.2] \\
\text{tcase contexts} & C ::= & B[\_][\tau] \mid B[\omega][\_] \\
\text{type reduction contexts} & T ::= & \_ \mid T\ \tau \mid T.1 \mid T.2
\end{array}
$$

Reduction $\boxed{e \longrightarrow e'}$

$$
\begin{array}{lrl}
\text{R-App} & LE[(\lambda x{:}\tau.e)\ v] & \longrightarrow LE[e[x := v]] \\
\text{R-Inst} & LE[(\lambda\alpha{:}\kappa.e)\ \tau] & \longrightarrow LE[e[\alpha := \tau]] \\
\text{R-Proj} & LE[\mathsf{let}\ \langle x_1,x_2\rangle = \langle v_1,v_2\rangle\ \mathsf{in}\ e] & \longrightarrow LE[e[x_1 := v_1][x_2 := v_2]] \\
\text{R-Open} & LE[\mathsf{let}\ \langle\alpha,x\rangle = \langle\tau,v\rangle{:}\tau'\ \mathsf{in}\ e] & \longrightarrow LE[e[\alpha := \tau][x := v]] \\
\text{R-Suspend} & LE[\mathsf{lazy}\ \langle\zeta,x\rangle = e_1\ \mathsf{in}\ e_2] & \longrightarrow L[\mathsf{lazy}\ \langle\zeta,x\rangle = e_1\ \mathsf{in}\ E[e_2]] \\
 & & \hfill (E \neq \_ \wedge \zeta \notin ftv(E) \wedge x \notin fv(E)) \\
\text{R-Trigger} & L_1[\mathsf{lazy}\ \langle\zeta,x\rangle = e\ \mathsf{in}\ L_2ES[x]] & \longrightarrow L_1[\mathsf{let}\ \langle\alpha,x\rangle = e\ \mathsf{in}\ (L_2ES[x])[\zeta := \alpha]] \\
 & & \hfill (x \notin btv(L_2)) \\
\text{R-Case1} & LE[\mathsf{tcase}\ v{:}\nu\ \mathsf{of}\ x{:}\nu\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2] & \longrightarrow LE[e_1[x := v]] \\
\text{R-Case2} & LE[\mathsf{tcase}\ v{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2] & \longrightarrow LE[e_2] \\
 & & \hfill ((\mathsf{tcase}\ v{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) = B[\omega][\omega']\ \text{and}\ \omega \not\sim \omega' \\
 & & \hfill \text{or}\ (\mathsf{tcase}\ v{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) = P[q][q']\ \text{and}\ q \not\sim q') \\
\text{RT-App} & LECT[(\lambda\alpha{:}\kappa.\tau_1)\ \tau_2] & \longrightarrow LECT[\tau_1[\alpha := \tau_2]] \\
\text{RT-Proj1} & LECT[\langle\tau_1,\tau_2\rangle.1] & \longrightarrow LECT[\tau_1] \\
\text{RT-Proj2} & LECT[\langle\tau_1,\tau_2\rangle.2] & \longrightarrow LECT[\tau_2] \\
\text{RT-Trigger} & L_1[\mathsf{lazy}\ \langle\zeta,x\rangle = e_1\ \mathsf{in}\ L_2ECT[\zeta]] & \longrightarrow L_1[\mathsf{let}\ \langle\alpha,x\rangle = e_1\ \mathsf{in}\ (L_2ECT[\zeta])[\zeta := \alpha]] \\
 & & \hfill (\zeta \notin btv(L_2))
\end{array}
$$

Figure A.5: The basic calculus + interleaved call-by-name reduction to weak head normal form on type-level (operational semantics)

- Case $B = B'[\nu \times \_][\nu \times \_]$:
  (a) By inversion of B-Times2 (1st derivation): $\Gamma \vdash B' : \Gamma_1$
  (b) By inversion of B-Times2 (2nd derivation): $\Gamma \vdash B' : \Gamma_2$
  (c) By (a), (b), and induction: $\Gamma_1 = \Gamma_2$
- Case $B = B'[\forall\alpha{:}\kappa.\_][\forall\alpha{:}\kappa.\_]$:
  (a) By inversion of B-Univ (1st derivation): $\Gamma \vdash B' : \Gamma_1'$ with $\Gamma_1 = \Gamma_1', \alpha{:}\kappa$
  (b) By inversion of B-Univ (2nd derivation): $\Gamma \vdash B' : \Gamma_2'$ with $\Gamma_2 = \Gamma_2', \alpha{:}\kappa$
  (c) By (a), (b), and induction: $\Gamma_1' = \Gamma_2'$
  (d) By (a), (b), and (c): $\Gamma_1 = \Gamma_2$
- Case $B = B'[\exists\alpha{:}\kappa.\_][\exists\alpha{:}\kappa.\_]$:
  (a) By inversion of B-Exist (1st derivation): $\Gamma \vdash B' : \Gamma_1'$ with $\Gamma_1 = \Gamma_1', \alpha{:}\kappa$
  (b) By inversion of B-Exist (2nd derivation): $\Gamma \vdash B' : \Gamma_2'$ with $\Gamma_2 = \Gamma_2', \alpha{:}\kappa$
  (c) By (a), (b), and induction: $\Gamma_1' = \Gamma_2'$
  (d) By (a), (b), and (c): $\Gamma_1 = \Gamma_2$
- Case $B = B'[\lambda\alpha{:}\kappa.\_][\lambda\alpha{:}\kappa.\_]$:
  (a) By inversion of B-Abs (1st derivation): $\Gamma \vdash B' : \Gamma_1'$ with $\Gamma_1 = \Gamma_1', \alpha{:}\kappa$
  (b) By inversion of B-Abs (2nd derivation): $\Gamma \vdash B' : \Gamma_2'$ with $\Gamma_2 = \Gamma_2', \alpha{:}\kappa$
  (c) By (a), (b), and induction: $\Gamma_1' = \Gamma_2'$
  (d) By (a), (b), and (c): $\Gamma_1 = \Gamma_2$
- Case $B = P'[p\ \_][p\ \_]$:
  (a) By inversion of B-Path (1st derivation): $\Gamma \vdash P' : \Gamma_1$
  (b) By inversion of B-Path (2nd derivation): $\Gamma \vdash P' : \Gamma_2$
  (c) By (a), (b), and induction: $\Gamma_1 = \Gamma_2$
- Case $B = B'[\langle \_, \tau_1\rangle][\langle \_, \tau_2\rangle]$:
  (a) By inversion of B-Pair1 (1st derivation): $\Gamma \vdash B' : \Gamma_1$
  (b) By inversion of B-Pair1 (2nd derivation): $\Gamma \vdash B' : \Gamma_2$
  (c) By (a), (b), and induction: $\Gamma_1 = \Gamma_2$
- Case $B = B'[\langle \nu, \_\rangle][\langle \nu, \_\rangle]$:
  (a) By inversion of B-Pair2 (1st derivation): $\Gamma \vdash B' : \Gamma_1$
  (b) By inversion of B-Pair2 (2nd derivation): $\Gamma \vdash B' : \Gamma_2$
  (c) By (a), (b), and induction: $\Gamma_1 = \Gamma_2$

2. Let $\Gamma \vdash P : \Gamma_1$ and $\Gamma \vdash P : \Gamma_2$.

- Case $P = B$:
  (a) By inversion of P-B (1st derivation): $\Gamma \vdash B : \Gamma_1$
  (b) By inversion of P-B (2nd derivation): $\Gamma \vdash B : \Gamma_2$
  (c) By (a), (b), and induction: $\Gamma_1 = \Gamma_2$
- Case $P = P'[\_\ \tau_1][\_\ \tau_2]$:
  (a) By inversion of P-App (1st derivation): $\Gamma \vdash P' : \Gamma_1$
  (b) By inversion of P-App (2nd derivation): $\Gamma \vdash P' : \Gamma_2$
  (c) By (a), (b), and induction: $\Gamma_1 = \Gamma_2$
- Case $P = P'[\_.1][\_.1]$:

(a) By inversion of P-Proj1 (1st derivation): $\Gamma \vdash P' : \Gamma_1$
(b) By inversion of P-Proj1 (2nd derivation): $\Gamma \vdash P' : \Gamma_2$
(c) By (a), (b), and induction: $\Gamma_1 = \Gamma_2$

- Case $P = P'[\_.2][\_.2]$:

(a) By inversion of P-Proj2 (1st derivation): $\Gamma \vdash P' : \Gamma_1$
(b) By inversion of P-Proj2 (2nd derivation): $\Gamma \vdash P' : \Gamma_2$
(c) By (a), (b), and induction: $\Gamma_1 = \Gamma_2$

□

Lemma 15 (Type Context Elimination)**.**

1. If $\Gamma \vdash B[\tau_1][\tau_2] : \tau$, then $\Gamma, \Gamma' \vdash \tau_1 : \kappa$ and $\Gamma, \Gamma' \vdash \tau_2 : \kappa$ where $\Gamma \vdash B : \Gamma'$.

2. If $\Gamma \vdash P[\tau_1][\tau_2] : \tau$, then $\Gamma, \Gamma' \vdash \tau_1 : \kappa_1$ and $\Gamma, \Gamma' \vdash \tau_2 : \kappa_2$ where $\Gamma \vdash P : \Gamma'$.

3. If $\Gamma \vdash B : \Gamma'$, then $dom(\Gamma') \cap LTVar = \emptyset$.

4. If $\Gamma \vdash C[\tau] : \tau'$, then $\Gamma, \Gamma' \vdash \tau : \kappa$ and $dom(\Gamma') \cap LTVar = \emptyset$.

5. If $\Gamma \vdash T[\tau] : \kappa$, then $\Gamma \vdash \tau : \kappa'$.

Proof. (1) and (2) simultaneous, by induction on the generation of B and P. (3) follows immediately from the definition of the $\Gamma \vdash B : \Gamma'$ judgement. (4) by case analysis on C, using (1). (5) by structural induction on T.

1. Let $\Gamma \vdash B[\tau_1][\tau_2] : \tau$.

- Case $B = \mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$:

(a) By Typing Inversion: $\Gamma \vdash v : \tau_1$ and $\Gamma, x{:}\tau_2 \vdash e_1 : \tau$
(b) By (a) and Validity: $\Gamma \vdash \tau_1 : \Omega$
(c) By (a) and Environment Validity: $\Gamma, x{:}\tau_2 \vdash \Box$
(d) By (c) and inversion of E-Term: $\Gamma \vdash \tau_2 : \Omega$
(e) By B-Case: $\Gamma \vdash \mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \cdot$

- Case $B = B'[\_ \to \tau_1'][\_ \to \tau_2']$:

(a) By induction: $\Gamma, \Gamma' \vdash \tau_1 \to \tau_1' : \kappa$ and $\Gamma, \Gamma' \vdash \tau_2 \to \tau_2' : \kappa$ where $\Gamma \vdash B' : \Gamma'$
(b) By (a) and inversion of K-Arrow: $\Gamma, \Gamma' \vdash \tau_1 : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2 : \Omega$
(c) By (a) and B-Arrow1: $\Gamma \vdash B'[\_ \to \tau_1'][\_ \to \tau_2'] : \Gamma'$

- Case $B = B'[\nu \to \_][\nu \to \_]$:

(a) By induction: $\Gamma, \Gamma' \vdash \nu \to \tau_1 : \kappa$ and $\Gamma, \Gamma' \vdash \nu \to \tau_2 : \kappa$ where $\Gamma \vdash B' : \Gamma'$
(b) By (a) and inversion of K-Arrow: $\Gamma, \Gamma' \vdash \tau_1 : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2 : \Omega$
(c) By (a) and B-Arrow2: $\Gamma \vdash B'[\nu \to \_][\nu \to \_] : \Gamma'$

- Case $B = B'[\_ \times \tau_1'][\_ \times \tau_2']$:

(a) By induction: $\Gamma, \Gamma' \vdash \tau_1 \times \tau_1' : \kappa$ and $\Gamma, \Gamma' \vdash \tau_2 \times \tau_2' : \kappa$ where $\Gamma \vdash B' : \Gamma'$
(b) By (a) and inversion of K-Times: $\Gamma, \Gamma' \vdash \tau_1 : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2 : \Omega$
(c) By (a) and B-Times1: $\Gamma \vdash B'[\_ \times \tau_1'][\_ \times \tau_2'] : \Gamma'$

- Case $B = B'[\nu \times \_][\nu \times \_]$:

(a) By induction: $\Gamma, \Gamma' \vdash \nu \times \tau_1 : \kappa$ and $\Gamma, \Gamma' \vdash \nu \times \tau_2 : \kappa$ where $\Gamma \vdash B' : \Gamma'$
(b) By (a) and inversion of K-Times: $\Gamma, \Gamma' \vdash \tau_1 : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2 : \Omega$

   (c) By (a) and B-Times2: $\Gamma \vdash B'[\nu \times \_][\nu \times \_] : \Gamma'$

- Case $B = B'[\forall\alpha{:}\kappa.\_][\forall\alpha{:}\kappa.\_]$:
   (a) By induction: $\Gamma, \Gamma' \vdash \forall\alpha{:}\kappa.\tau_1 : \kappa'$ and $\Gamma, \Gamma' \vdash \forall\alpha{:}\kappa.\tau_2 : \kappa'$ where $\Gamma \vdash B' : \Gamma'$
   (b) By (a) and inversion of K-Univ: $\Gamma, \Gamma', \alpha{:}\kappa \vdash \tau_1 : \Omega$ and $\Gamma, \Gamma', \alpha{:}\kappa \vdash \tau_2 : \Omega$
   (c) By (a) and B-Univ: $\Gamma \vdash B'[\forall\alpha{:}\kappa.\_][\forall\alpha{:}\kappa.\_] : \Gamma', \alpha{:}\kappa$
- Case $B = B'[\exists\alpha{:}\kappa.\_][\exists\alpha{:}\kappa.\_]$:
   (a) By induction: $\Gamma, \Gamma' \vdash \exists\alpha{:}\kappa.\tau_1 : \kappa'$ and $\Gamma, \Gamma' \vdash \exists\alpha{:}\kappa.\tau_2 : \kappa'$ where $\Gamma \vdash B' : \Gamma'$
   (b) By (a) and inversion of K-Exist: $\Gamma, \Gamma', \alpha{:}\kappa \vdash \tau_1 : \Omega$ and $\Gamma, \Gamma', \alpha{:}\kappa \vdash \tau_2 : \Omega$
   (c) By (a) and B-Exist: $\Gamma \vdash B'[\exists\alpha{:}\kappa.\_][\exists\alpha{:}\kappa.\_] : \Gamma', \alpha{:}\kappa$
- Case $B = B'[\lambda\alpha{:}\kappa.\_][\lambda\alpha{:}\kappa.\_]$:
   (a) By induction: $\Gamma, \Gamma' \vdash \lambda\alpha{:}\kappa.\tau_1 : \kappa'$ and $\Gamma, \Gamma' \vdash \lambda\alpha{:}\kappa.\tau_2 : \kappa'$ where $\Gamma \vdash B' : \Gamma'$
   (b) By (a) and inversion of K-Abs: $\kappa' = \kappa \to \kappa''$ and $\Gamma, \Gamma', \alpha{:}\kappa \vdash \tau_1 : \kappa''$ and $\Gamma, \Gamma', \alpha{:}\kappa \vdash \tau_2 : \kappa''$
   (c) By (a) and B-Abs: $\Gamma \vdash B'[\lambda\alpha{:}\kappa.\_][\lambda\alpha{:}\kappa.\_] : \Gamma', \alpha{:}\kappa$
- Case $B = P[p\ \_][p\ \_]$:
   (a) By induction: $\Gamma, \Gamma' \vdash p\ \tau_1 : \kappa_1$ and $\Gamma, \Gamma' \vdash p\ \tau_2 : \kappa_2$ where $\Gamma \vdash P : \Gamma'$
   (b) By (a) and inversion of K-App: $\Gamma, \Gamma' \vdash p : \kappa_1' \to \kappa_1$ and $\Gamma, \Gamma' \vdash \tau_1 : \kappa_1'$ and $\Gamma, \Gamma' \vdash p : \kappa_2' \to \kappa_2$ and $\Gamma, \Gamma' \vdash \tau_2 : \kappa_2'$
   (c) By (b) and Uniqueness of Kinds: $\kappa_1' \to \kappa_1 = \kappa_2' \to \kappa_2$ and hence $\kappa_1' = \kappa_2'$
   (d) By (a) and B-Path: $\Gamma \vdash P[p\ \_][p\ \_] : \Gamma'$
- Case $B = B'[\langle\_, \tau_1'\rangle][\langle\_, \tau_2'\rangle]$:
   (a) By induction: $\Gamma, \Gamma' \vdash \langle\tau_1, \tau_1'\rangle : \kappa$ and $\Gamma, \Gamma' \vdash \langle\tau_2, \tau_2'\rangle : \kappa$ where $\Gamma \vdash B' : \Gamma'$
   (b) By (a) and inversion of K-Pair: $\kappa = \kappa' \times \kappa''$ and $\Gamma, \Gamma' \vdash \tau_1 : \kappa'$ and $\Gamma, \Gamma' \vdash \tau_2 : \kappa'$
   (c) By (a) and B-Pair1: $\Gamma \vdash B'[\langle\_, \tau_1'\rangle][\langle\_, \tau_2'\rangle] : \Gamma'$
- Case $B = B'[\langle\nu, \_\rangle][\langle\nu, \_\rangle]$:
   (a) By induction: $\Gamma, \Gamma' \vdash \langle\nu, \tau_1\rangle : \kappa$ and $\Gamma, \Gamma' \vdash \langle\nu, \tau_2\rangle : \kappa$ where $\Gamma \vdash B' : \Gamma'$
   (b) By (a) and inversion of K-Pair: $\kappa = \kappa'' \times \kappa'$ and $\Gamma, \Gamma' \vdash \tau_1 : \kappa'$ and $\Gamma, \Gamma' \vdash \tau_2 : \kappa'$
   (c) By (a) and B-Pair2: $\Gamma \vdash B'[\langle\nu, \_\rangle][\langle\nu, \_\rangle] : \Gamma'$

2. Let $\Gamma \vdash P[\tau_1][\tau_2] : \tau$.

- Case $P = B$:
   (a) By induction: $\Gamma, \Gamma' \vdash \tau_1 : \kappa$ and $\Gamma, \Gamma' \vdash \tau_2 : \kappa$ where $\Gamma \vdash B : \Gamma'$
- Case $P = P'[\_\ \tau_1'][\_\ \tau_2']$:
   (a) By induction: $\Gamma, \Gamma' \vdash \tau_1\ \tau_1' : \kappa_1$ and $\Gamma, \Gamma' \vdash \tau_2\ \tau_2' : \kappa_2$ where $\Gamma \vdash P' : \Gamma'$
   (b) By (a) and inversion of K-App: $\Gamma, \Gamma' \vdash \tau_1' : \kappa_1'$ and $\Gamma, \Gamma' \vdash \tau_2' : \kappa_2'$
   (c) By (a) and P-App: $\Gamma \vdash P'[\_\ \tau_1'][\_\ \tau_2'] : \Gamma'$
- Case $P = P'[\_.1][\_.1]$:
   (a) By induction: $\Gamma, \Gamma' \vdash \tau_1.1 : \kappa_1$ and $\Gamma, \Gamma' \vdash \tau_2.1 : \kappa_2$ where $\Gamma \vdash P' : \Gamma'$
   (b) By (a) and inversion of K-Pair1: $\Gamma, \Gamma' \vdash \tau_1 : \kappa_1 \times \kappa_1'$ and $\Gamma, \Gamma' \vdash \tau_2 : \kappa_2 \times \kappa_2'$
   (c) By (a) and P-Proj1: $\Gamma \vdash P'[\_.1][\_.1] : \Gamma'$
- Case $P = P'[\_.2][\_.2]$:
   (a) By induction: $\Gamma, \Gamma' \vdash \tau_1.2 : \kappa_1$ and $\Gamma, \Gamma' \vdash \tau_2.2 : \kappa_2$ where $\Gamma \vdash P' : \Gamma'$

(b) By (a) and inversion of K-Pair2: $\Gamma, \Gamma' \vdash \tau_1 : \kappa_1' \times \kappa_1$ and $\Gamma, \Gamma' \vdash \tau_2 : \kappa_2' \times \kappa_2$

(c) By (a) and P-Proj2: $\Gamma \vdash P'[\_.2][\_.2] : \Gamma'$

4. Let $\Gamma \vdash C[\tau] : \tau'$.

   - Case $C = B[\_][\tau'']$:

     (a) By assumption: $\Gamma \vdash B[\tau][\tau''] : \tau'$

     (b) By (a): $\Gamma, \Gamma' \vdash \tau : \kappa$ where $\Gamma \vdash B : \Gamma'$.

     (c) By (b) and (3): $\mathit{dom}(\Gamma') \cap \mathit{LTVar} = \emptyset$

   - Case $C = B[\omega][\_]$: analogous

5. Let $\Gamma \vdash T[\tau] : \kappa$.

   - Case $T = \_$:

     (a) By assumption: $\Gamma \vdash \tau : \kappa$

   - Case $T = T'\ \tau'$:

     (a) By inversion of K-App: $\Gamma \vdash T'[\tau] : \kappa' \to \kappa$

     (b) By (a) and induction: $\Gamma \vdash \tau : \kappa''$

   - Case $T = T'.1$:

     (a) By inversion of K-Proj1: $\Gamma \vdash T'[\tau] : \kappa \times \kappa_2$

     (b) By (a) and induction: $\Gamma \vdash \tau : \kappa'$

   - Case $T = T'.2$:

     (a) By inversion of K-Proj2: $\Gamma \vdash T'[\tau] : \kappa_1 \times \kappa$

     (b) By (a) and induction: $\Gamma \vdash \tau : \kappa'$

□

Lemma 16 (Type Exchange)**.**

1. *If* $\Gamma \vdash T[\tau] : \kappa$ *and* $\Gamma \vdash \tau : \kappa'$ *as well as* $\Gamma \vdash \tau' : \kappa'$*, then* $\Gamma \vdash T[\tau'] : \kappa$*.*

2. *If* $\Gamma \vdash B[\tau_1][\tau_2] : \tau$ *and* $\Gamma, \Gamma' \vdash \tau_1 : \kappa$ *and* $\Gamma, \Gamma' \vdash \tau_1' : \kappa$ *and* $\tau_1 \equiv \tau_1'$ *and* $\Gamma, \Gamma' \vdash \tau_2 : \kappa$ *and* $\Gamma, \Gamma' \vdash \tau_2' : \kappa$ *and* $\tau_2 \equiv \tau_2'$ *and* $\Gamma \vdash B : \Gamma'$*, then* $\Gamma \vdash B[\tau_1'][\tau_2']$*.*

3. *If* $\Gamma \vdash P[\tau_1][\tau_2] : \tau$ *and* $\Gamma, \Gamma' \vdash \tau_1 : \kappa$ *and* $\Gamma, \Gamma' \vdash \tau_1' : \kappa$ *and* $\tau_1 \equiv \tau_1'$ *and* $\Gamma, \Gamma' \vdash \tau_2 : \kappa$ *and* $\Gamma, \Gamma' \vdash \tau_2' : \kappa$ *and* $\tau_2 \equiv \tau_2'$ *and* $\Gamma \vdash P : \Gamma'$*, then* $\Gamma \vdash P[\tau_1'][\tau_2']$*.*

Proof. (1) by structural induction on T. (2) and (3) simultaneous by induction on the generation of B and P.

1. Let $\Gamma \vdash T[\tau] : \kappa$ and $\Gamma \vdash \tau : \kappa'$ as well as $\Gamma \vdash \tau' : \kappa'$.

   - Case $T = \_$:

     (a) By assumption and Uniqueness of Kinds: $\kappa = \kappa'$

     (b) By (a) and assumption: $\Gamma \vdash \tau' : \kappa$

   - Case $T = T'\ \tau''$:

     (a) By inversion of K-App: $\Gamma \vdash T'[\tau] : \kappa'' \to \kappa$ and $\Gamma \vdash \tau'' : \kappa''$

     (b) By (a), assumption, and induction: $\Gamma \vdash T'[\tau'] : \kappa'' \to \kappa$

     (c) By (b), (a), and K-App: $\Gamma \vdash T'[\tau']\ \tau'' : \kappa$

     (d) By (c): $\Gamma \vdash (T'\ \tau'')[\tau'] : \kappa$

- Case $T = T'.1$:
  (a) By inversion of K-PROJ1: $\Gamma \vdash T'[\tau] : \kappa \times \kappa_2$
  (b) By (a), assumption, and induction: $\Gamma \vdash T'[\tau'] : \kappa \times \kappa_2$
  (c) By (b) and KPROJ1: $\Gamma \vdash T'[\tau'].1 : \kappa$
  (d) By (c): $\Gamma \vdash T'.1[\tau'] : \kappa$
- Case $T = T'.2$:
  (a) By inversion of K-PROJ2: $\Gamma \vdash T'[\tau] : \kappa_1 \times \kappa$
  (b) By (a), assumption, and induction: $\Gamma \vdash T'[\tau'] : \kappa_1 \times \kappa$
  (c) By (b) and KPROJ2: $\Gamma \vdash T'[\tau'].2 : \kappa$
  (d) By (c): $\Gamma \vdash T'.2[\tau'] : \kappa$

2. Let $\Gamma \vdash B[\tau_1][\tau_2] : \tau$ and $\Gamma, \Gamma' \vdash \tau_1 : \kappa$ and $\Gamma, \Gamma' \vdash \tau_1' : \kappa$ and $\tau_1 \equiv \tau_1'$ and $\Gamma, \Gamma' \vdash \tau_2 : \kappa$ and $\Gamma, \Gamma' \vdash \tau_2' : \kappa$ and $\tau_2 \equiv \tau_2'$ and $\Gamma \vdash B : \Gamma'$.

- Case $B = \mathsf{tcase}\ v : \_\ \mathsf{of}\ x : \_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$ (w.l.o.g. $x \notin dom(\Gamma)$):
  (a) By Typing Inversion:
    i. $\Gamma \vdash v : \tau_1$
    ii. $\Gamma, x{:}\tau_2 \vdash e_1 : \tau$
    iii. $\Gamma \vdash e_2 : \tau$
  (b) By inversion of B-CASE: $\Gamma' = \cdot$
  (c) We show: $\Gamma \vdash v : \tau_1'$
    i. By (a-i) and Validity: $\Gamma \vdash \tau_1 : \Omega$
    ii. By (i), (b), assumption, and Uniqueness of Kinds: $\kappa = \Omega$
    iii. By (ii), (b), and assumption: $\Gamma \vdash \tau_1' : \Omega$
    iv. By (a-i), (iii), assumption, and TEQUIV: $\Gamma \vdash v : \tau_1'$
  (d) We show: $\Gamma, x{:}\tau_2' \vdash e_1 : \tau$
    i. By (c-ii), (b), and assumption: $\Gamma \vdash \tau_2' : \Omega$
    ii. By (i) and ETERM: $\Gamma, x{:}\tau_2' \vdash \square$
    iii. By assumption: $\Gamma, x{:}\tau_2 \equiv \Gamma, x{:}\tau_2'$
    iv. By (a-ii), (ii), (iv), and Equivalent Environments: $\Gamma, x{:}\tau_2' \vdash e_1 : \tau$
  (e) By (c), (d), (a-iii), and TCASE: $\Gamma \vdash \mathsf{tcase}\ v : \tau_1'\ \mathsf{of}\ x : \tau_2'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau$
- Case $B = B'[\_ \to \tau_1''][\_ \to \tau_2'']$:
  (a) By inversion of B-ARROW1: $\Gamma \vdash B' : \Gamma'$
  (b) By Type Context Elimination: $\Gamma, \Gamma'' \vdash \tau_1 \to \tau_1'' : \kappa'$ and $\Gamma, \Gamma'' \vdash \tau_2 \to \tau_2'' : \kappa'$ where $\Gamma \vdash B' : \Gamma''$
  (c) By (a), (b), and Uniqueness of Environments: $\Gamma' = \Gamma''$
  (d) By (b), (c), and inversion of K-ARROW: $\kappa' = \Omega$ and $\Gamma, \Gamma' \vdash \tau_1 : \Omega$ and $\Gamma, \Gamma' \vdash \tau_1'' : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2 : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2'' : \Omega$
  (e) By (d), assumption, and Uniqueness of Kinds: $\kappa = \Omega$
  (f) By (d), (e), assumption, and K-ARROW: $\Gamma, \Gamma' \vdash \tau_1' \to \tau_1'' : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2' \to \tau_2'' : \Omega$
  (g) By assumption, Q-REFL, and Q-ARROW: $\tau_1 \to \tau_1'' \equiv \tau_1' \to \tau_1''$ and $\tau_2 \to \tau_2'' \equiv \tau_2' \to \tau_2''$
  (h) By assumption, (a), (d), (f), (g), and induction: $\Gamma \vdash B'[\tau_1' \to \tau_1''][\tau_2' \to \tau_2''] : \tau$
  (i) By (h): $\Gamma \vdash B[\tau_1'][\tau_2'] : \tau$

- Case $B = B'[\nu \to \_][\nu \to \_]$:
  (a) By inversion of B-Arrow2: $\Gamma \vdash B' : \Gamma'$
  (b) By Type Context Elimination: $\Gamma, \Gamma'' \vdash \nu \to \tau_1 : \kappa'$ and $\Gamma, \Gamma'' \vdash \nu \to \tau_2 : \kappa'$ where $\Gamma \vdash B' : \Gamma''$
  (c) By (a), (b), and Uniqueness of Environments: $\Gamma' = \Gamma''$
  (d) By (b), (c), and inversion of K-Arrow: $\kappa' = \Omega$ and $\Gamma, \Gamma' \vdash \nu : \Omega$ and $\Gamma, \Gamma' \vdash \tau_1 : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2 : \Omega$
  (e) By (d), assumption, and Uniqueness of Kinds: $\kappa = \Omega$
  (f) By (d), (e), assumption, and K-Arrow: $\Gamma, \Gamma' \vdash \nu \to \tau_1' : \Omega$ and $\Gamma, \Gamma' \vdash \nu \to \tau_2' : \Omega$
  (g) By assumption, Q-Refl, and Q-Arrow: $\nu \to \tau_1' \equiv \nu \to \tau_1'$ and $\nu \to \tau_2 \equiv \nu \to \tau_2'$
  (h) By assumption, (a), (d), (f), (g), and induction: $\Gamma \vdash B'[\tau_1' \to \tau_1''][\tau_2' \to \tau_2''] : \tau$
  (i) By (h): $\Gamma \vdash B[\tau_1'][\tau_2'] : \tau$
- Case $B = B'[\_ \times \tau_1''][\_ \times \tau_2'']$:
  (a) By inversion of B-Times1: $\Gamma \vdash B' : \Gamma'$
  (b) By Type Context Elimination: $\Gamma, \Gamma'' \vdash \tau_1 \times \tau_1'' : \kappa'$ and $\Gamma, \Gamma'' \vdash \tau_2 \times \tau_2'' : \kappa'$ where $\Gamma \vdash B' : \Gamma''$
  (c) By (a), (b), and Uniqueness of Environments: $\Gamma' = \Gamma''$
  (d) By (b), (c), and inversion of K-Times: $\kappa' = \Omega$ and $\Gamma, \Gamma' \vdash \tau_1 : \Omega$ and $\Gamma, \Gamma' \vdash \tau_1'' : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2 : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2'' : \Omega$
  (e) By (d), assumption, and Uniqueness of Kinds: $\kappa = \Omega$
  (f) By (d), (e), assumption, and K-Times: $\Gamma, \Gamma' \vdash \tau_1' \times \tau_1'' : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2' \times \tau_2'' : \Omega$
  (g) By assumption, Q-Refl, and Q-Times: $\tau_1 \times \tau_1'' \equiv \tau_1' \times \tau_1''$ and $\tau_2 \times \tau_2'' \equiv \tau_2' \times \tau_2''$
  (h) By assumption, (a), (d), (f), (g), and induction: $\Gamma \vdash B'[\tau_1' \times \tau_1''][\tau_2' \times \tau_2''] : \tau$
  (i) By (h): $\Gamma \vdash B[\tau_1'][\tau_2'] : \tau$
- Case $B = B'[\nu \times \_][\nu \times \_]$:
  (a) By inversion of B-Times2: $\Gamma \vdash B' : \Gamma'$
  (b) By Type Context Elimination: $\Gamma, \Gamma'' \vdash \nu \times \tau_1 : \kappa'$ and $\Gamma, \Gamma'' \vdash \nu \times \tau_2 : \kappa'$ where $\Gamma \vdash B' : \Gamma''$
  (c) By (a), (b), and Uniqueness of Environments: $\Gamma' = \Gamma''$
  (d) By (b), (c), and inversion of K-Times: $\kappa' = \Omega$ and $\Gamma, \Gamma' \vdash \nu : \Omega$ and $\Gamma, \Gamma' \vdash \tau_1 : \Omega$ and $\Gamma, \Gamma' \vdash \tau_2 : \Omega$
  (e) By (d), assumption, and Uniqueness of Kinds: $\kappa = \Omega$
  (f) By (d), (e), assumption, and K-Times: $\Gamma, \Gamma' \vdash \nu \times \tau_1' : \Omega$ and $\Gamma, \Gamma' \vdash \nu \times \tau_2' : \Omega$
  (g) By assumption, Q-Refl, and Q-Times: $\nu \times \tau_1 \equiv \nu \times \tau_1'$ and $\nu \times \tau_2 \equiv \nu \times \tau_2'$
  (h) By assumption, (a), (b), (d), (f), (g), and induction: $\Gamma \vdash B'[\nu \times \tau_1'][\nu \times \tau_2'] : \tau$
  (i) By (h): $\Gamma \vdash B[\tau_1'][\tau_2'] : \tau$
- Case $B = B'[\forall\alpha{:}\kappa'.\_][\forall\alpha{:}\kappa'.\_]$:
  (a) By inversion of B-Univ: $\Gamma \vdash B' : \Gamma''$ and $\Gamma' = \Gamma'', \alpha{:}\kappa'$
  (b) By Type Context Elimination: $\Gamma, \Gamma''' \vdash \forall\alpha'{:}\kappa'.\tau_1 : \kappa''$ and $\Gamma, \Gamma''' \vdash \forall\alpha{:}\kappa'.\tau_2 : \kappa''$ where $\Gamma \vdash B' : \Gamma'''$
  (c) By (a), (b), and Uniqueness of Environments: $\Gamma'' = \Gamma'''$
  (d) By (c), (b), and inversion of K-Univ: $\kappa'' = \Omega$ and $\Gamma, \Gamma'', \alpha'{:}\kappa' \vdash \tau_1 : \Omega$ and $\Gamma, \Gamma'', \alpha'{:}\kappa' \vdash \tau_2 : \Omega$
  (e) By (d), (a), assumption, and Uniqeness of Kinds: $\kappa = \Omega$

(f) By (a), (e), assumption, and K-Univ: $\Gamma, \Gamma'' \vdash \forall\alpha{:}\kappa'.\tau_1' : \Omega$ and $\Gamma, \Gamma'' \vdash \forall\alpha{:}\kappa'.\tau_2' : \Omega$

(g) By assumption and Q-Univ: $\forall\alpha{:}\kappa'.\tau_1 \equiv \forall\alpha{:}\kappa'.\tau_1'$ and $\forall\alpha{:}\kappa'.\tau_2 \equiv \forall\alpha{:}\kappa'.\tau_2'$

(h) By assumption, (a), (b), (c), (d), (f), (g), and induction: $\Gamma \vdash B'[\forall\alpha{:}\kappa'.\tau_1'][\forall\alpha{:}\kappa'.\tau_2'] : \tau$

(i) By (h): $\Gamma \vdash B[\tau_1'][\tau_2'] : \tau$

- Case $B = B'[\exists\alpha{:}\kappa'.\_][\exists\alpha{:}\kappa'.\_]$:

  (a) By inversion of B-Exist: $\Gamma \vdash B' : \Gamma''$ and $\Gamma' = \Gamma'', \alpha{:}\kappa'$

  (b) By Type Context Elimination: $\Gamma, \Gamma''' \vdash \exists\alpha'{:}\kappa.\tau_1 : \kappa'$ and $\Gamma, \Gamma''' \vdash \exists\alpha{:}\alpha'.\tau_2 : \kappa'$ where $\Gamma \vdash B' : \Gamma'''$

  (c) By (a), (b), and Uniqueness of Environments: $\Gamma'' = \Gamma'''$

  (d) By (c), (b), and inversion of K-Exist: $\kappa'' = \Omega$ and $\Gamma, \Gamma'', \alpha'{:}\kappa' \vdash \tau_1 : \Omega$ and $\Gamma, \Gamma'', \alpha'{:}\kappa' \vdash \tau_2 : \Omega$

  (e) By (d), (a), assumption, and Uniqeness of Kinds: $\kappa = \Omega$

  (f) By (a), (e), assumption, and K-Exist: $\Gamma, \Gamma'' \vdash \exists\alpha{:}\kappa'.\tau_1' : \Omega$ and $\Gamma, \Gamma'' \vdash \exists\alpha{:}\kappa'.\tau_2' : \Omega$

  (g) By assumption and Q-Exist: $\exists\alpha{:}\kappa'.\tau_1 \equiv \exists\alpha{:}\kappa'.\tau_1'$ and $\exists\alpha{:}\kappa'.\tau_2 \equiv \exists\alpha{:}\kappa'.\tau_2'$

  (h) By assumption, (a), (b), (c), (d), (f), (g), and induction: $\Gamma \vdash B'[\exists\alpha{:}\kappa'.\tau_1'][\exists\alpha{:}\kappa'.\tau_2'] : \tau$

  (i) By (h): $\Gamma \vdash B[\tau_1'][\tau_2'] : \tau$

- Case $B = B'[\lambda\alpha{:}\kappa'.\_][\lambda\alpha{:}\kappa'.\_]$:

  (a) By inversion of B-Abs: $\Gamma \vdash B' : \Gamma''$ and $\Gamma' = \Gamma'', \alpha{:}\kappa'$

  (b) By Type Context Elimination: $\Gamma, \Gamma''' \vdash \lambda\alpha'{:}\kappa.\tau_1 : \kappa'$ and $\Gamma, \Gamma''' \vdash \lambda\alpha{:}\alpha'.\tau_2 : \kappa'$ where $\Gamma \vdash B' : \Gamma'''$

  (c) By (a), (b), and Uniqueness of Environments: $\Gamma'' = \Gamma'''$

  (d) By (c), (b), and inversion of K-Univ: $\kappa'' = \kappa' \to \kappa'''$ and $\Gamma, \Gamma'', \alpha'{:}\kappa' \vdash \tau_1 : \kappa'''$ and $\Gamma, \Gamma'', \alpha'{:}\kappa' \vdash \tau_2 : \kappa'''$

  (e) By (d), (a), assumption, and Uniqeness of Kinds: $\kappa = \kappa'''$

  (f) By (a), (e), assumption, and K-Abs: $\Gamma, \Gamma'' \vdash \lambda\alpha{:}\kappa'.\tau_1' : \kappa' \to \kappa'''$ and $\Gamma, \Gamma'' \vdash \lambda\alpha{:}\kappa'.\tau_2' : \kappa' \to \kappa'''$

  (g) By assumption and Q-Abs: $\lambda\alpha{:}\kappa'.\tau_1 \equiv \lambda\alpha{:}\kappa'.\tau_1'$ and $\lambda\alpha{:}\kappa'.\tau_2 \equiv \lambda\alpha{:}\kappa'.\tau_2'$

  (h) By assumption, (a), (b), (c), (d), (f), (g), and induction: $\Gamma \vdash B'[\lambda\alpha{:}\kappa'.\tau_1'][\lambda\alpha{:}\kappa'.\tau_2'] : \tau$

  (i) By (h): $\Gamma \vdash B[\tau_1'][\tau_2'] : \tau$

- Case $B = P[p\ \_][p\ \_]$:

  (a) By inversion of B-Path: $\Gamma \vdash P : \Gamma'$

  (b) By Type Context Elimination: $\Gamma, \Gamma'' \vdash p\ \tau_1 : \kappa_1$ and $\Gamma, \Gamma'' \vdash p\ \tau_2 : \kappa_2$ where $\Gamma \vdash B' : \Gamma''$

  (c) By (a), (b), and Uniqueness of Environments: $\Gamma' = \Gamma''$

  (d) By (b), (c), and inversion of K-App: $\Gamma, \Gamma' \vdash p : \kappa_1' \to \kappa_1$ and $\Gamma, \Gamma' \vdash \tau_1 : \kappa_1'$ and $\Gamma, \Gamma' \vdash p : \kappa_2' \to \kappa_2$ and $\Gamma, \Gamma' \vdash \tau_2 : \kappa_2'$

  (e) By (d), assumption, and Uniqueness of Kinds: $\kappa_1' = \kappa_2' = \kappa$ and $\kappa_1 = \kappa_2$

  (f) By (d), (e), assumption, and K-App: $\Gamma, \Gamma' \vdash p\ \tau_1' : \kappa_1$ and $\Gamma, \Gamma' \vdash p\ \tau_2' : \kappa_1$

  (g) By assumption, Q-Refl, and Q-App: $p\ \tau_1 \equiv p\ \tau_1'$ and $p\ \tau_2 \equiv p\ \tau_2'$

  (h) By assumption, (a), (b), (e), (f), (g), and induction: $\Gamma \vdash P[p\ \tau_1'][p\ \tau_2'] : \tau$

(i) By (h): $\Gamma \vdash B[\tau_1'][\tau_2'] : \tau$

- Case $B = B'[\langle \_, \tau_1'' \rangle][\langle \_, \tau_2'' \rangle]$:
  (a) By inversion of B-Pair1: $\Gamma \vdash B' : \Gamma'$
  (b) By Type Context Elimination: $\Gamma, \Gamma'' \vdash \langle \tau_1, \tau_1'' \rangle : \kappa'$ and $\Gamma, \Gamma'' \vdash \langle \tau_2, \tau_2'' \rangle : \kappa'$ where $\Gamma \vdash B' : \Gamma''$
  (c) By (a), (b), and Uniqueness of Environments: $\Gamma' = \Gamma''$
  (d) By (b), (c), inversion of K-Pair, and Uniqueness of Kinds: $\kappa' = \kappa_1 \times \kappa_1'$ and $\Gamma, \Gamma' \vdash \tau_1 : \kappa_1$ and $\Gamma, \Gamma' \vdash \tau_1'' : \kappa_1'$ and $\kappa' = \kappa_2 \times \kappa_2'$ and $\Gamma, \Gamma' \vdash \tau_2 : \kappa_2$ and $\Gamma, \Gamma' \vdash \tau_2'' : \kappa_2'$
  (e) By (d), assumption, and Uniqueness of Kinds: $\kappa_1 = \kappa_2 = \kappa$ and $\kappa_1' = \kappa_2'$
  (f) By (d), (e), assumption, and K-Pair: $\Gamma, \Gamma' \vdash \langle \tau_1', \tau_1'' \rangle : \kappa \times \kappa_1'$ and $\Gamma, \Gamma' \vdash \langle \tau_2', \tau_2'' \rangle : \kappa \times \kappa_1'$
  (g) By assumption, Q-Refl, and Q-Pair: $\langle \tau_1, \tau_1'' \rangle \equiv \langle \tau_1', \tau_1'' \rangle$ and $\langle \tau_2, \tau_2'' \rangle \equiv \langle \tau_2', \tau_2'' \rangle$
  (h) By assumption, (a), (b), (d), (f), (g), and induction: $\Gamma \vdash B'[\langle \tau_1', \tau_1'' \rangle][\langle \tau_2', \tau_2'' \rangle] : \tau$
  (i) By (h): $\Gamma \vdash B[\tau_1'][\tau_2'] : \tau$
- Case $B = B'[\langle \nu, \_ \rangle][\langle \nu, \_ \rangle]$:
  (a) By inversion of B-Pair2: $\Gamma \vdash B' : \Gamma'$
  (b) By Type Context Elimination: $\Gamma, \Gamma'' \vdash \langle \nu, \tau_1 \rangle : \kappa'$ and $\Gamma, \Gamma'' \vdash \langle \nu, \tau_2 \rangle : \kappa'$ where $\Gamma \vdash B' : \Gamma''$
  (c) By (a), (b), and Uniqueness of Environments: $\Gamma' = \Gamma''$
  (d) By (b), (c), inversion of K-Pair, and Uniqueness of Kinds: $\kappa' = \kappa_1 \times \kappa_1'$ and $\Gamma, \Gamma' \vdash \nu : \kappa_1$ and $\Gamma, \Gamma' \vdash \tau_1 : \kappa_1'$ and $\kappa' = \kappa_2 \times \kappa_2'$ and $\Gamma, \Gamma' \vdash \nu : \kappa_2$ and $\Gamma, \Gamma' \vdash \tau_2 : \kappa_2'$
  (e) By (d), assumption, and Uniqueness of Kinds: $\kappa_1 = \kappa_2$ and $\kappa_1' = \kappa_2' = \kappa$
  (f) By (d), (e), assumption, and K-Pair: $\Gamma, \Gamma' \vdash \langle \nu, \tau_1' \rangle : \kappa_1 \times \kappa$ and $\Gamma, \Gamma' \vdash \langle \nu, \tau_2' \rangle : \kappa_1 \times \kappa$
  (g) By assumption, Q-Refl, and Q-Pair: $\langle \nu, \tau_1 \rangle \equiv \langle \nu, \tau_1' \rangle$ and $\langle \nu, \tau_2 \rangle \equiv \langle \nu, \tau_2' \rangle$
  (h) By assumption, (a), (b), (d), (f), (g), and induction: $\Gamma \vdash B'[\langle \nu, \tau_1' \rangle][\langle \nu, \tau_2' \rangle] : \tau$
  (i) By (h): $\Gamma \vdash B[\tau_1'][\tau_2'] : \tau$

3. Let $\Gamma \vdash P[\tau_1][\tau_2] : \tau$ and $\Gamma, \Gamma' \vdash \tau_1 : \kappa$ and $\Gamma, \Gamma' \vdash \tau_1' : \kappa$ and $\tau_1 \equiv \tau_1'$ and $\Gamma, \Gamma' \vdash \tau_2 : \kappa$ and $\Gamma, \Gamma' \vdash \tau_2' : \kappa$ and $\tau_2 \equiv \tau_2'$ and $\Gamma \vdash P : \Gamma'$.
   - Case $P = B$:
     (a) By assumption and P-B: $\Gamma \vdash B : \Gamma'$
     (b) By assumption, (a), and induction: $\Gamma, \Gamma' \vdash B[\tau_1'][\tau_2'] : \tau$
   - Case $P = P'[\_\ \tau_1''][\_\ \tau_2'']$:
     (a) By inversion of P-App: $\Gamma \vdash P' : \Gamma'$
     (b) By Type Context Elimination: $\Gamma, \Gamma'' \vdash \tau_1\ \tau_1'' : \kappa_1$ and $\Gamma, \Gamma'' \vdash \tau_2\ \tau_2'' : \kappa_2$ where $\Gamma \vdash P' : \Gamma''$
     (c) By (a), (b), and Uniqueness of Environments: $\Gamma' = \Gamma''$
     (d) By (b), (c), and inversion of K-App: $\Gamma, \Gamma' \vdash \tau_1 : \kappa_1' \rightarrow \kappa_1$ and $\Gamma, \Gamma' \vdash \tau_1'' : \kappa_1'$ and $\Gamma, \Gamma' \vdash \tau_2 : \kappa_2' \rightarrow \kappa_2$ and $\Gamma, \Gamma' \vdash \tau_2'' : \kappa_2'$
     (e) By (d), assumption, and Uniqueness of Kinds: $\kappa_1 = \kappa_2$ and $\kappa_1' = \kappa_2'$ and $\kappa = \kappa_1' \rightarrow \kappa_1$
     (f) By (d), (e), assumption, and K-App: $\Gamma, \Gamma' \vdash \tau_1'\ \tau_1'' : \kappa'$ and $\Gamma, \Gamma' \vdash \tau_2'\ \tau_2'' : \kappa'$

(g) By assumption, Q-Refl, and Q-App: $\tau_1\ \tau_1'' \equiv \tau_1'\ \tau_1''$ and $\tau_2\ \tau_2'' \equiv \tau_2'\ \tau_2''$
(h) By assumption, (a), (b), (e), (f), (g), and induction: $\Gamma, \Gamma' \vdash P'[\tau_1'\ \tau_1''][\tau_2'\ \tau_2''] : \tau$
(i) By (h): $\Gamma, \Gamma' \vdash P[\tau_1'][\tau_2'] : \tau$

- Case $P = P'[\_.1][\_.1]$:
  (a) By inversion of P-Proj1: $\Gamma \vdash P' : \Gamma'$
  (b) By Type Context Elimination: $\Gamma, \Gamma'' \vdash \tau_1.1 : \kappa_1$ and $\Gamma, \Gamma'' \vdash \tau_2.1 : \kappa_2$ where $\Gamma \vdash P' : \Gamma''$
  (c) By (a), (b), and Uniqueness of Environments: $\Gamma' = \Gamma''$
  (d) By (b), (c), and inversion of K-Proj1: $\Gamma, \Gamma' \vdash \tau_1 : \kappa_1 \times \kappa_1'$ and $\Gamma, \Gamma' \vdash \tau_2 : \kappa_2 \times \kappa_2'$
  (e) By (d), assumption, and Uniqueness of Kinds: $\kappa_1' = \kappa_2'$ and $\kappa_1 = \kappa_2$ and $\kappa = \kappa_1 \times \kappa_1'$
  (f) By (d), (e), assumption, and KProj1: $\Gamma, \Gamma' \vdash \tau_1'.1 : \kappa_1$ and $\Gamma, \Gamma' \vdash \tau_2'.1 : \kappa_2$
  (g) By assumption and Q-Proj1a: $\tau_1.1 \equiv \tau_1'.1$ and $\tau_2.1 \equiv \tau_2'.1$
  (h) By assumption, (a), (b), (e), (f), (g), and induction: $\Gamma, \Gamma' \vdash P'[\tau_1'.1][\tau_2'.1] : \tau$
  (i) By (h): $\Gamma, \Gamma' \vdash P[\tau_1'][\tau_2'] : \tau$
- Case $P = P'[\_.2][\_.2]$:
  (a) By inversion of P-Proj2: $\Gamma \vdash P' : \Gamma'$
  (b) By Type Context Elimination: $\Gamma, \Gamma'' \vdash \tau_1.2 : \kappa_1$ and $\Gamma, \Gamma'' \vdash \tau_2.2 : \kappa_2$ where $\Gamma \vdash P' : \Gamma''$
  (c) By (a), (b), and Uniqueness of Environments: $\Gamma' = \Gamma''$
  (d) By (b), (c), and inversion of K-Proj2: $\Gamma, \Gamma' \vdash \tau_1 : \kappa_1 \times \kappa_1'$ and $\Gamma, \Gamma' \vdash \tau_2 : \kappa_2 \times \kappa_2'$
  (e) By (d), assumption, and Uniqueness of Kinds: $\kappa_1 = \kappa_2$ and $\kappa_1' = \kappa_2'$ and $\kappa = \kappa_1 \times \kappa_1'$
  (f) By (d), (e), assumption, and KProj2: $\Gamma, \Gamma' \vdash \tau_1'.2 : \kappa_1$ and $\Gamma, \Gamma' \vdash \tau_2'.2 : \kappa_2$
  (g) By assumption and Q-Proj2a: $\tau_1.2 \equiv \tau_1'.2$ and $\tau_2.2 \equiv \tau_2'.2$
  (h) By assumption, (a), (b), (e), (f), (g), and induction: $\Gamma, \Gamma' \vdash P'[\tau_1'.2][\tau_2'.2] : \tau$
  (i) By (h): $\Gamma, \Gamma' \vdash P[\tau_1'][\tau_2'] : \tau$

□

Lemma 17 (Wrapping). *If $\tau \equiv \tau'$, then, for all T, $T[\tau] \equiv T[\tau']$.*

Proof. By structural induction on T.

- Case $T = \_$:
  1. By assumption: $\tau \equiv \tau'$
- Case $T = T'\ \tau''$:
  1. By induction: $T'[\tau] \equiv T'[\tau']$
  2. By (1), Q-Refl, and Q-App: $T'[\tau]\ \tau'' \equiv T'[\tau']\ \tau''$
  3. By (2): $(T'\ \tau'')[\tau] \equiv (T'\ \tau'')[\tau']$
- Case $T = T'.1$:
  1. By induction: $T'[\tau] \equiv T'[\tau']$
  2. By (1) and Q-Proj1a: $T'[\tau].1 \equiv T'[\tau'].1$
  3. By (2): $(T'.1)[\tau] \equiv (T'.1)[\tau']$

- Case $T = T'.2$:

  1. By induction: $T'[\tau] \equiv T'[\tau']$
  2. By (1) and Q-Proj2a: $T'[\tau].2 \equiv T'[\tau'].2$
  3. By (2): $(T'.2)[\tau] \equiv (T'.2)[\tau']$

$\square$

Theorem 3 (Preservation)**.** *If* $\Gamma \vdash e : \tau$ *and* $e \longrightarrow e'$*, then* $\Gamma \vdash e' : \tau$*.*

Proof. By case analysis on the applied reduction rule. We show only the cases that differ from the proof for the applicative order strategy.

- Case RT-App: $e = \mathit{LECT}[(\lambda\alpha{:}\kappa.\tau_1)\ \tau_2)]$ and $e' = \mathit{LECT}[\tau_1[\alpha := \tau_2]]$

  1. By Context Elimination: $\Gamma, \Gamma' \vdash \mathit{CT}[(\lambda\alpha{:}\kappa.\tau_1)\ \tau_2)] : \tau'$ where $\Gamma \vdash L : \Gamma'$
  2. We show: $\Gamma, \Gamma' \vdash \mathit{CT}[\tau_1[\alpha := \tau_2]] : \tau'$
     - Subcase $C = B[\_][\tau_3]$:
       (a) By (1): $\Gamma, \Gamma' \vdash B[T[(\lambda\alpha{:}\kappa.\tau_1)\ \tau_2)]][\tau_3] : \tau'$
       (b) By (a) and Type Context Elimination: $\Gamma, \Gamma', \Gamma'' \vdash T[(\lambda\alpha{:}\kappa.\tau_1)\ \tau_2] : \kappa'$ and $\Gamma, \Gamma', \Gamma'' \vdash \tau_3 : \kappa'$ where $\Gamma, \Gamma' \vdash B : \Gamma''$ (w.l.o.g. $\alpha \notin \mathit{dom}(\Gamma, \Gamma', \Gamma'')$)
       (c) By (b) and Type Context Elimination: $\Gamma, \Gamma', \Gamma'' \vdash (\lambda\alpha{:}\kappa.\tau_1)\ \tau_2 : \kappa''$
       (d) By (c) and inversion of K-App: $\Gamma, \Gamma', \Gamma'' \vdash \lambda\alpha{:}\kappa.\tau_1 : \kappa_2 \to \kappa''$ and $\Gamma, \Gamma', \Gamma'' \vdash \tau_2 : \kappa_2$
       (e) By (d), (b), and inversion of K-Abs: $\Gamma, \Gamma', \Gamma'', \alpha{:}\kappa \vdash \tau_1 : \kappa''$ and $\kappa = \kappa_2$
       (f) By (d), (e), and Type Substitution: $\Gamma, \Gamma', \Gamma'' \vdash \tau_1[\alpha := \tau_2] : \kappa''$
       (g) By (b), (c), (f), and Type Exchange: $\Gamma, \Gamma', \Gamma'' \vdash T[\tau_1[\alpha := \tau_2]] : \kappa'$
       (h) By Q-Beta: $(\lambda\alpha{:}\kappa.\tau_1)\ \tau_2 \equiv \tau_1[\alpha := \tau_2]$
       (i) By (h) and Wrapping: $T[(\lambda\alpha{:}\kappa.\tau_1)\ \tau_2)] \equiv T[\tau_1[\alpha := \tau_2]]$
       (j) By (a), (b), (g), (i), and Type Exchange: $\Gamma, \Gamma' \vdash B[T[\tau_1[\alpha := \tau_2]]][\tau_3] : \tau'$
       (k) By (j): $\Gamma, \Gamma' \vdash \mathit{CT}[\tau_1[\alpha := \tau_2]] : \tau'$
     - Subcase $C = B[\omega][\_]$: analogous
  3. By assumption, (1), (2), and Exchange: $\Gamma \vdash \mathit{LECT}[\tau_1[\alpha := \tau_2]] : \tau$

- Case RT-Proj1: $e = \mathit{LECT}[\langle\tau_1, \tau_2\rangle.1]$ and $e' = \mathit{LECT}[\tau_1]$

  1. By Context Elimination: $\Gamma, \Gamma' \vdash \mathit{CT}[\langle\tau_1, \tau_2\rangle.1] : \tau'$ where $\Gamma \vdash L : \Gamma'$
  2. We show: $\Gamma, \Gamma' \vdash \mathit{CT}[\tau_1] : \tau'$
     - Subcase $C = B[\_][\tau_3]$:
       (a) By (1): $\Gamma, \Gamma' \vdash B[T[\langle\tau_1, \tau_2\rangle.1]][\tau_3] : \tau'$
       (b) By (a) and Type Context Elimination: $\Gamma, \Gamma', \Gamma'' \vdash T[\langle\tau_1, \tau_2\rangle.1] : \kappa'$ and $\Gamma, \Gamma', \Gamma'' \vdash \tau_3 : \kappa'$ where $\Gamma, \Gamma' \vdash B : \Gamma''$
       (c) By (b) and Type Context Elimination: $\Gamma, \Gamma', \Gamma'' \vdash \langle\tau_1, \tau_2\rangle.1 : \kappa''$
       (d) By (c) and inversion of K-Proj1: $\Gamma, \Gamma', \Gamma'' \vdash \langle\tau_1, \tau_2\rangle : \kappa'' \times \kappa'''$
       (e) By (d) and inversion of K-Pair: $\Gamma, \Gamma', \Gamma'' \vdash \tau_1 : \kappa''$
       (f) By (b), (c), (e), and Type Exchange: $\Gamma, \Gamma', \Gamma'' \vdash T[\tau_1] : \kappa'$
       (g) By Q-Proj1b: $\langle\tau_1, \tau_2\rangle.1 \equiv \tau_1$
       (h) By (g) and Wrapping: $T[\langle\tau_1, \tau_2\rangle.1] \equiv T[\tau_1]$
       (i) By (a), (b), (f), (h), and Type Exchange: $\Gamma, \Gamma' \vdash B[T[\tau_1]][\tau_3] : \tau'$

(j) By (i): $\Gamma, \Gamma' \vdash CT[\tau_1] : \tau'$

– Subcase $C = B[\omega][\_]$: analogous

3. By assumption, (1), (2), and Exchange: $\Gamma \vdash LECT[\tau_1] : \tau$

- Case RT-Proj2: $e = LECT[\langle\tau_1, \tau_2\rangle.2]$ and $e' = LECT[\tau_2]$

  1. By Context Elimination: $\Gamma, \Gamma' \vdash CT[\langle\tau_1, \tau_2\rangle.2] : \tau'$ where $\Gamma \vdash L : \Gamma'$
  2. We show: $\Gamma, \Gamma' \vdash CT[\tau_2] : \tau'$
     - Subcase $C = B[\_][\tau_3]$:
       (a) By (1): $\Gamma, \Gamma' \vdash B[T[\langle\tau_1, \tau_2\rangle.2]][\tau_3] : \tau'$
       (b) By (a) and Type Context Elimination: $\Gamma, \Gamma', \Gamma'' \vdash T[\langle\tau_1, \tau_2\rangle.2] : \kappa'$ and $\Gamma, \Gamma', \Gamma'' \vdash \tau_3 : \kappa'$ where $\Gamma, \Gamma' \vdash B : \Gamma''$
       (c) By (b) and Type Context Elimination: $\Gamma, \Gamma', \Gamma'' \vdash \langle\tau_1, \tau_2\rangle.2 : \kappa''$
       (d) By (c) and inversion of K-Proj2: $\Gamma, \Gamma', \Gamma'' \vdash \langle\tau_1, \tau_2\rangle : \kappa''' \times \kappa''$
       (e) By (d) and inversion of K-Pair: $\Gamma, \Gamma', \Gamma'' \vdash \tau_2 : \kappa''$
       (f) By (b), (c), (e), and Type Exchange: $\Gamma, \Gamma', \Gamma'' \vdash T[\tau_2] : \kappa'$
       (g) By Q-Proj1b: $\langle\tau_1, \tau_2\rangle.2 \equiv \tau_2$
       (h) By (g) and Wrapping: $T[\langle\tau_1, \tau_2\rangle.2] \equiv T[\tau_2]$
       (i) By (a), (b), (f), (h), and Type Exchange: $\Gamma, \Gamma' \vdash B[T[\tau_2]][\tau_3] : \tau'$
       (j) By (i): $\Gamma, \Gamma' \vdash CT[\tau_2] : \tau'$
     - Subcase $C = B[\omega][\_]$: analogous
  3. By assumption, (1), (2), and Exchange: $\Gamma \vdash LECT[\tau_2] : \tau$

□

Lemma 18 (Lazy Type Variables). *If* $\Gamma \vdash LCT[\zeta] : \tau$ *and* $\zeta \notin dom(\Gamma)$*, then* $L = L_1[\text{lazy } \langle\zeta, x\rangle = e \text{ in } L_2]$ *where* $\zeta \notin btv(L_2)$*.*

Proof. By structural induction on L. W.l.o.g. all bound type variables of $LCT[\zeta]$ are distinct and $(btv(LCT[\zeta]) \cup bv(LCT[\zeta])) \cap dom(\Gamma) = \emptyset$.

- Case $L = \_$:
  1. By assumption and Type Context Elimination: $\Gamma, \Gamma' \vdash T[\zeta] : \kappa$ and $\zeta \notin dom(\Gamma')$
  2. By (1) and Type Context Elimination: $\Gamma, \Gamma' \vdash \zeta : \kappa'$
  3. By (2) and Variable Containment: $\zeta \in dom(\Gamma, \Gamma')$
  4. By (1) and (3): $\zeta \in dom(\Gamma)$
  5. (4) contradicts the assumption $\zeta \notin dom(\Gamma)$, hence this case is not possible.

- Case $L = \text{lazy } \langle\zeta', x\rangle = e \text{ in } L'$:
  - Subcase $\zeta' = \zeta \wedge \zeta \notin btv(L')$: this is what we claimed
  - Subcase $\zeta' = \zeta \wedge \zeta \in btv(L')$: not possible due to our assumption about bound type variables
  - Subcase $\zeta' \neq \zeta$:
    1. By assumption and Typing Inversion: $\Gamma, \zeta':\kappa, x:\tau' \vdash L'CT[\zeta] : \tau''$
    2. By assumption: $\zeta \notin dom(\Gamma, \zeta':\kappa, x:\tau')$
    3. By (1), (2), and induction: $L' = L'_1[\text{lazy } \langle\zeta, x'\rangle = e' \text{ in } L_2]$ where $\zeta \notin btv(L_2)$
    4. Let $L_1 = \text{lazy } \langle\zeta', x\rangle = e \text{ in } L'_1$.

5. By (3) and (4): $L = L_1[\mathsf{lazy}\ \langle\zeta, x'\rangle\ \mathsf{=}\ e'\ \mathsf{in}\ L_2]$ where $\zeta \notin btv(L_2)$

$\square$

Proposition 16 (Weight property)**.** *Let X range over B and P.*

1. *If (X', τ_1', τ_2') is equal to or a deeper decomposition than (X, τ_1, τ_2), then $weight(X, \tau_1, \tau_2) - weight(X', \tau_1', \tau_2') \leq size(\tau_1) - size(\tau_1')$.*
2. *If (X, τ_1', τ_2') is a decomposition of some term, then $weight(X, \tau_1', \tau_2') > 0$.*
3. *Let (X_1, ν_1, ν_1) be equal to or a deeper decomposition than (B', ν, ν) and similar (X_2, ν_2, ν_2) equal to or a deeper decomposition than (P', p, p).*
   - *If $B' = B[\_ \to \tau_1][\_ \to \tau_2]$, then $weight(B[\nu \to \_][\nu \to \_], \tau_1, \tau_2) < weight(X_1, \nu_1, \nu_1)$.*
   - *If $B' = B[\_ \times \tau_1][\_ \times \tau_2]$, then $weight(B[\nu \times \_][\nu \times \_], \tau_1, \tau_2) < weight(X_1, \nu_1, \nu_1)$.*
   - *If $B' = B[\langle \_, \tau_1\rangle][\langle \_, \tau_2\rangle]$, then $weight(B[\langle \nu, \_\rangle][\langle \nu, \_\rangle], \tau_1, \tau_2) < weight(X_1, \nu_1, \nu_1)$.*
   - *If $B' = B[\langle \_, \tau_1\rangle][\langle \_, \tau_2\rangle]$, then $weight(B[\langle \nu, \_\rangle][\langle \nu, \_\rangle], \tau_1, \tau_2) < weight(X_1, \nu_1, \nu_1)$.*
   - *If $P' = P[\_\ \tau_1][\_\ \tau_2]$, then $weight(P[p\ \_][p\ \_], \tau_1, \tau_2) < weight(X_2, \nu_2, \nu_2)$.*

Proof. (1) by induction on the generation of X. (2) follows from (1). (3) follows from (1) and the definition of the size function.

1. Let (X', τ_1', τ_2') be equal to or a deeper decomposition than (X, τ_1, τ_2).
   - Case $X' = X$:
     (a) Hence: $\tau_1 = \tau_1'$ and $\tau_2 = \tau_2'$
     (b) By (a): $weight(X, \tau_1, \tau_2) - weight(X', \tau_1', \tau_2') = 0 = size(\tau_1) - size(\tau_1')$
   - Case $X' = X''[\_ \to \tau_1''][\_ \to \tau_2'']$ and $X' \neq X$:
     (a) By assumption: $(X'', \tau_1' \to \tau_1'', \tau_2' \to \tau_2'')$ is equal to or a deeper decomposition than (X, τ_1, τ_2)
     (b) By (a) and induction: $weight(X, \tau_1, \tau_2) - weight(X'', \tau_1' \to \tau_1'', \tau_2' \to \tau_2'') \leq size(\tau_1) - size(\tau_1' \to \tau_1'')$
     (c) By definition of *weight* and *size*: $weight(X'', \tau_1' \to \tau_1'', \tau_2' \to \tau_2'') - weight(X''[\_ \to \tau_1''][\_ \to \tau_2''], \tau_1', \tau_2') = size(\tau_1'') < size(\tau_1') + 2 \cdot size(\tau_1'') = size(\tau_1' \to \tau_1'') - size(\tau_1')$
     (d) By (b) and (c): $weight(X, \tau_1, \tau_2) - weight(X''[\_ \to \tau_1''][\_ \to \tau_2''], \tau_1', \tau_2') < size(\tau_1) - size(\tau_1')$
   - Case $X' = X''[\nu \to \_][\nu \to \_]$ and $X' \neq X$:
     (a) By assumption: $(X'', \nu \to \tau_1', \nu \to \tau_2')$ is equal to or a deeper decomposition than (X, τ_1, τ_2):
     (b) By (a) and induction: $weight(X, \tau_1, \tau_2) - weight(X'', \nu \to \tau_1', \nu \to \tau_2') \leq size(\tau_1) - size(\nu \to \tau_1')$
     (c) By definition of *weight* and *size*: $weight(X'', \nu \to \tau_1', \nu \to \tau_2') - weight(X''[\nu \to \_][\nu \to \_], \tau_1', \tau_2') = size(\nu) + size(\tau_1') < 2 \cdot size(\nu) + size(\tau_1') = size(\nu \to \tau_1') - size(\tau_1')$
     (d) By (b) and (c): $weight(X, \tau_1, \tau_2) - weight(X''[\nu \to \_][\nu \to \_], \tau_1', \tau_2') \leq size(\tau_1) - size(\tau_1')$
   - Case $X' = X''[\_ \times \tau_1''][\_ \times \tau_2'']$ and $X' \neq X$:
     (a) By assumption: $(X'', \tau_1' \times \tau_1'', \tau_2' \times \tau_2'')$ is equal to or a deeper decomposition than (X, τ_1, τ_2):

(b) By (a) and induction: $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X'', \tau_1' \times \tau_1'', \tau_2' \times \tau_2'') \leq \mathit{size}(\tau_1) - \mathit{size}(\tau_1' \times \tau_1'')$

(c) By definition of *weight* and *size*: $\mathit{weight}(X'', \tau_1' \times \tau_1'', \tau_2' \times \tau_2'') - \mathit{weight}(X''[\_ \times \tau_1''][\_ \times \tau_2''], \tau_1', \tau_2') = \mathit{size}(\tau_1'') < \mathit{size}(\tau_1') + 2 \cdot \mathit{size}(\tau_1'') = \mathit{size}(\tau_1' \times \tau_1'') - \mathit{size}(\tau_1')$

(d) By (b) and (c): $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X''[\_ \times \tau_1''][\_ \times \tau_2''], \tau_1', \tau_2') < \mathit{size}(\tau_1) - \mathit{size}(\tau_1')$

- Case $X' = X''[\nu \times \_][\nu \times \_]$ and $(X'', \nu \times \tau_1', \nu \times \tau_2')$ is equal to or a deeper decomposition than (X, τ_1, τ_2):

  (a) By induction: $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X'', \nu \times \tau_1', \nu \times \tau_2') \leq \mathit{size}(\tau_1) - \mathit{size}(\nu \times \tau_1')$

  (b) By definition of *weight* and *size*: $\mathit{weight}(X'', \nu \times \tau_1', \nu \times \tau_2') - \mathit{weight}(X''[\nu \times \_][\nu \times \_], \tau_1', \tau_2') = \mathit{size}(\nu) + \mathit{size}(\tau_1') < 2 \cdot \mathit{size}(\nu) + \mathit{size}(\tau_1') = \mathit{size}(\nu \times \tau_1') - \mathit{size}(\tau_1')$

  (c) By (a) and (b): $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X''[\nu \times \_][\nu \times \_], \tau_1', \tau_2') \leq \mathit{size}(\tau_1) - \mathit{size}(\tau_1')$

- Case $X' = X''[\forall\alpha{:}\kappa.\_][\forall\alpha{:}\kappa.\_]$ and $(X'', \forall\alpha{:}\kappa.\tau_1', \forall\alpha{:}\kappa.\tau_2')$ is equal to or a deeper decomposition than (X, τ, τ_2):

  (a) By induction: $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X'', \forall\alpha{:}\kappa.\tau_1', \forall\alpha{:}\kappa.\tau_2') \leq \mathit{size}(\tau_1) - \mathit{size}(\forall\alpha{:}\kappa.\tau_1')$

  (b) By definition of *weight* and *size*: $\mathit{weight}(X'', \forall\alpha{:}\kappa.\tau_1', \forall\alpha{:}\kappa.\tau_2') - \mathit{weight}(X''[\forall\alpha{:}\kappa.\_][\forall\alpha{:}\kappa.\_], \tau_1', \tau_2') = 1 < 2 = \mathit{size}(\forall\alpha{:}\kappa.\tau_1') - \mathit{size}(\tau_1')$

  (c) By (a) and (b): $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X''[\forall\alpha{:}\kappa.\_][\forall\alpha{:}\kappa.\_], \tau_1', \tau_2') < \mathit{size}(\tau_1) - \mathit{size}(\tau_1')$

- Case $X' = X''[\exists\alpha{:}\kappa.\_][\exists\alpha{:}\kappa.\_]$ and $(X'', \exists\alpha{:}\kappa.\tau_1', \exists\alpha{:}\kappa.\tau_2')$ is equal to or a deeper decomposition than (X, τ, τ_2):

  (a) By induction: $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X'', \exists\alpha{:}\kappa.\tau_1', \exists\alpha{:}\kappa.\tau_2') \leq \mathit{size}(\tau_1) - \mathit{size}(\exists\alpha{:}\kappa.\tau_1')$

  (b) By definition of *weight* and *size*: $\mathit{weight}(X'', \exists\alpha{:}\kappa.\tau_1', \exists\alpha{:}\kappa.\tau_2') - \mathit{weight}(X''[\exists\alpha{:}\kappa.\_][\exists\alpha{:}\kappa.\_], \tau_1', \tau_2') = 1 < 2 = \mathit{size}(\exists\alpha{:}\kappa.\tau_1') - \mathit{size}(\tau_1')$

  (c) By (a) and (b): $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X''[\exists\alpha{:}\kappa.\_][\exists\alpha{:}\kappa.\_], \tau_1', \tau_2') < \mathit{size}(\tau_1) - \mathit{size}(\tau_1')$

- Case $X' = X''[\lambda\alpha{:}\kappa.\_][\lambda\alpha{:}\kappa.\_]$ and $(X'', \lambda\alpha{:}\kappa.\tau_1', \lambda\alpha{:}\kappa.\tau_2')$ is equal to or a deeper decomposition than (X, τ, τ_2):

  (a) By induction: $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X'', \lambda\alpha{:}\kappa.\tau_1', \lambda\alpha{:}\kappa.\tau_2') \leq \mathit{size}(\tau_1) - \mathit{size}(\lambda\alpha{:}\kappa.\tau_1')$

  (b) By definition of *weight* and *size*: $\mathit{weight}(X'', \lambda\alpha{:}\kappa.\tau_1', \lambda\alpha{:}\kappa.\tau_2') - \mathit{weight}(X''[\lambda\alpha{:}\kappa.\_][\lambda\alpha{:}\kappa.\_], \tau_1', \tau_2') = 1 < 2 = \mathit{size}(\lambda\alpha{:}\kappa.\tau_1') - \mathit{size}(\tau_1')$

  (c) By (a) and (b): $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X''[\lambda\alpha{:}\kappa.\_][\lambda\alpha{:}\kappa.\_], \tau_1', \tau_2') < \mathit{size}(\tau_1) - \mathit{size}(\tau_1')$

- Case $X' = X''[p\ \_][p\ \_]$:

  (a) By induction: $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X'', p\ \tau_1', p\ \tau_2') \leq \mathit{size}(\tau_1) - \mathit{size}(p\ \tau_1')$

  (b) By definition of *weight* and *size*: $\mathit{weight}(X'', p\ \tau_1', p\ \tau_2') - \mathit{weight}(X''[p\ \_][p\ \_], \tau_1', \tau_2') = \mathit{size}(p) + \mathit{size}(\tau_1') < 2 \cdot \mathit{size}(p) + \mathit{size}(\tau_1') = \mathit{size}(p\ \tau_1') - \mathit{size}(\tau_1')$

  (c) By (a) and (b): $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X''[p\ \_][p\ \_], \tau_1', \tau_2') \leq \mathit{size}(\tau_1) - \mathit{size}(\tau_1')$

- Case $X' = X''[\langle \_, \tau_1'' \rangle][\langle \_, \tau_2'' \rangle]$:

  (a) By induction: $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X'', \langle \tau_1', \tau_1'' \rangle, \langle \tau_2', \tau_2'' \rangle) \leq \mathit{size}(\tau_1) - \mathit{size}(\langle \tau_1', \tau_1'' \rangle)$

(b) By definition of *weight* and *size*: $\mathit{weight}(X'', \langle \tau_1', \tau_1'' \rangle, \langle \tau_2', \tau_2'' \rangle) - \mathit{weight}(X''[\langle \_, \tau_1'' \rangle][\langle \_, \tau_2'' \rangle], \tau_1', \tau_2') = \mathit{size}(\tau_1'') < \mathit{size}(\tau_1') + 2 \cdot \mathit{size}(\tau_1'') = \mathit{size}(\langle \tau_1', \tau_1'' \rangle) - \mathit{size}(\tau_1')$

(c) By (a) and (b): $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X''[\langle \_, \tau_1'' \rangle][\langle \_, \tau_2'' \rangle], \tau_1', \tau_2') < \mathit{size}(\tau_1) - \mathit{size}(\tau_1')$

- Case $X' = X''[\langle \nu, \_ \rangle][\langle \nu, \_ \rangle]$:

  (a) By induction: $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X'', \langle \nu, \tau_1' \rangle, \langle \nu, \tau_2' \rangle) \leq \mathit{size}(\tau_1) - \mathit{size}(\langle \nu, \tau_1' \rangle)$

  (b) By definition of *weight* and *size*: $\mathit{weight}(X'', \langle \nu, \tau_1' \rangle, \langle \nu, \tau_2' \rangle) - \mathit{weight}(X''[\langle \nu, \_ \rangle][\langle \nu, \_ \rangle], \tau_1', \tau_2') = \mathit{size}(\nu) + \mathit{size}(\tau_1') < 2 \cdot \mathit{size}(\nu) + \mathit{size}(\tau_1') = \mathit{size}(\langle \nu, \tau_1' \rangle) - \mathit{size}(\tau_1')$

  (c) By (a) and (b): $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X''[\langle \nu, \_ \rangle][\langle \nu, \_ \rangle], \tau_1', \tau_2') \leq \mathit{size}(\tau_1) - \mathit{size}(\tau_1')$

- Case $X' = X''[\_.1][\_.1]$:

  (a) By induction: $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X'', \tau_1'.1, \tau_2'.1) \leq \mathit{size}(\tau_1) - \mathit{size}(\tau_1'.1)$

  (b) By definition of *weight* and *size*: $\mathit{weight}(X'', \tau_1'.1, \tau_2'.1) - \mathit{weight}(X''[\_.1][\_.1], \tau_1', \tau_2') = 1 < 2 = \mathit{size}(\tau_1'.1) - \mathit{size}(\tau_1')$

  (c) By (a) and (b): $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X''[\_.1][\_.1], \tau_1', \tau_2') < \mathit{size}(\tau_1) - \mathit{size}(\tau_1')$

- Case $X' = X''[\_.2][\_.2]$:

  (a) By induction: $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X'', \tau_1'.1, \tau_2'.1) \leq \mathit{size}(\tau_1) - \mathit{size}(\tau_1'.1)$

  (b) By definition of *weight* and *size*: $\mathit{weight}(X'', \tau_1'.1, \tau_2'.1) - \mathit{weight}(X''[\_.1][\_.1], \tau_1', \tau_2') = 1 < 2 = \mathit{size}(\tau_1'.1) - \mathit{size}(\tau_1')$

  (c) By (a) and (b): $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X''[\_.1][\_.1], \tau_1', \tau_2') < \mathit{size}(\tau_1) - \mathit{size}(\tau_1')$

- Case $X' = X''[\_\ \tau_1''][\_\ \tau_2'']$:

  (a) By induction: $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X'', \tau_1'\ \tau_1'', \tau_2'\ \tau_2'') \leq \mathit{size}(\tau_1) - \mathit{size}(\tau_1'\ \tau_1'')$

  (b) By definition of *weight* and *size*: $\mathit{weight}(X'', \tau_1'\ \tau_1'', \tau_2'\ \tau_2'') - \mathit{weight}(X''[\_ \to \tau_1''][\_ \to \tau_2''], \tau_1', \tau_2') = \mathit{size}(\tau_1'') < \mathit{size}(\tau_1') + 2 \cdot \mathit{size}(\tau_1'') = \mathit{size}(\tau_1'\ \tau_1'') - \mathit{size}(\tau_1')$

  (c) By (a) and (b): $\mathit{weight}(X, \tau_1, \tau_2) - \mathit{weight}(X''[\_\ \tau_1''][\_\ \tau_2''], \tau_1', \tau_2') < \mathit{size}(\tau_1) - \mathit{size}(\tau_1')$

2. Let (X, τ_1', τ_2') be a decomposition of e.

   (a) By definition of binary contexts: $e = \mathsf{tcase}\ v{:}\tau_1\ \mathsf{of}\ x{:}\tau_2\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$

   (b) By (a) and (1): $\mathit{weight}(\mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2, \tau_1, \tau_2) - \mathit{weight}(X, \tau_1', \tau_2') \leq \mathit{size}(\tau_1) - \mathit{size}(\tau_1')$

   (c) By (b): $\mathit{weight}(X, \tau_1', \tau_2') \geq \mathit{weight}(\mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2, \tau_1, \tau_2) - \mathit{size}(\tau_1) + \mathit{size}(\tau_1')$

   (d) By (c) and definition: $\mathit{weight}(X, \tau_1', \tau_2') \geq \mathit{size}(\tau_1')$

   (e) By (d) and definition: $\mathit{weight}(X, \tau_1', \tau_2') > 0$

3. Let (X_1, ν_1, ν_1) be equal to or a deeper decomposition than (B', ν, ν) and similar (X_2, p', p') equal to or a deeper decomposition than (P', p, p).

   - Assume $B' = B[\_ \to \tau_1][\_ \to \tau_2]$.

     (a) By (1): $\mathit{weight}(B[\_ \to \tau_1][\_ \to \tau_2], \nu, \nu) - \mathit{weight}(X_1, \nu_1, \nu_1) \leq \mathit{size}(\nu) - \mathit{size}(\nu_1) < \mathit{size}(\nu)$

     (b) By definition: $\mathit{weight}(B[\_ \to \tau_1][\_ \to \tau_2], \nu, \nu) - \mathit{weight}(B[\nu \to \_][\nu \to \_], \tau_1, \tau_2) = \mathit{size}(\nu)$

(c) By (a) and (b): $\mathit{weight}(B[\nu \to \_][\nu \to \_], \tau_1, \tau_2) < \mathit{weight}(X_1, \nu_1, \nu_1)$.

- Assume $B' = B[\_ \times \tau_1][\_ \times \tau_2]$.
  (a) By (1): $\mathit{weight}(B[\_ \times \tau_1][\_ \times \tau_2], \nu, \nu) - \mathit{weight}(X_1, \nu_1, \nu_1) \leq \mathit{size}(\nu) - \mathit{size}(\nu_1) < \mathit{size}(\nu)$
  (b) By definition: $\mathit{weight}(B[\_ \times \tau_1][\_ \times \tau_2], \nu, \nu) - \mathit{weight}(B[\nu \times \_][\nu \times \_], \tau_1, \tau_2) = \mathit{size}(\nu)$
  (c) By (a) and (b): $\mathit{weight}(B[\nu \times \_][\nu \times \_], \tau_1, \tau_2) < \mathit{weight}(X_1, \nu_1, \nu_1)$.
- Assume $B' = B[\langle \_, \tau_1\rangle][\langle \_, \tau_2\rangle]$.
  (a) By (1): $\mathit{weight}(B[\langle \_, \tau_1\rangle][\langle \_, \tau_2\rangle], \nu, \nu) - \mathit{weight}(X_1, \nu_1, \nu_1) \leq \mathit{size}(\nu) - \mathit{size}(\nu_1) < \mathit{size}(\nu)$
  (b) By definition: $\mathit{weight}(B[\langle \_, \tau_1\rangle][\langle \_, \tau_2\rangle], \nu, \nu) - \mathit{weight}(B[\langle \nu, \_\rangle][\langle \nu, \_\rangle], \tau_1, \tau_2) = \mathit{size}(\nu)$
  (c) By (a) and (b): $\mathit{weight}(B[\langle \nu, \_\rangle][\langle \nu, \_\rangle], \tau_1, \tau_2) < \mathit{weight}(X_1, \nu_1, \nu_1)$.
- Assume $P' = P[\_\ \tau_1][\_\ \tau_2]$.
  (a) By (1): $\mathit{weight}(P[\_\ \tau_1][\_\ \tau_2], p, p) - \mathit{weight}(X_2, \nu_2, \nu_2) \leq \mathit{size}(p) - \mathit{size}(\nu) < \mathit{size}(p)$
  (b) By definition: $\mathit{weight}(P[\_\ \tau_1][\_\ \tau_2], p, p) - \mathit{weight}(P[p\ \_][p\ \_], \tau_1, \tau_2) = \mathit{size}(p)$
  (c) By (a) and (b): $\mathit{weight}(P[p\ \_][p\ \_], \tau_1, \tau_2) < \mathit{weight}(X_2, \nu_2, \nu_2)$.

□

Proposition 17 (Type Progress)**.**

1. *If $\cdot \vdash LCT[\tau_0] : \tau$ and τ_0 is not in weak head normal form, then $LCT[\tau_0] \longrightarrow e$.*
2. *If $e = LP[p][p]$, then there exists a decomposition $(B, \tau_1, \tau_2) \ll (P, p, p)$ or (P, p, p) is equal to or deeper than some decomposition (B, p', p').*
3. *If $e = LB[\nu][\nu]$, then $e \longrightarrow e'$ or there exists a decomposition $(B', \tau_1, \tau_2) \ll (B, \nu, \nu)$.*
4. *If $e = LP[q_1][q_2]$, then $e \longrightarrow e'$ or there exists a decomposition (B, τ_1, τ_2) of $P[q_1][q_2]$ with $\mathit{weight}(B, \tau_1, \tau_2) < \mathit{weight}(P, q_1, q_2)$.*
5. *If $\cdot \vdash LB[\tau_1][\tau_2] : \tau$, then $LB[\tau_1][\tau_2] \longrightarrow e$.*

Proof. (1) by structural induction on τ_0. (2) by induction on the generation of P. (3) by induction on the generation of B. (4) by structural induction on q_1. (5) by induction on $\mathit{weight}(B, \tau_1, \tau_2)$.

1. Let $\cdot \vdash LCT[\tau_0] : \tau$ where τ_0 is not in weak head normal form.
   - Case $\tau_0 = \alpha$: not possible
   - Case $\tau_0 = \zeta$:
     (a) By Lazy Type Variables: $L = L_1[\mathsf{lazy}\ \langle \zeta, x\rangle\ \mathsf{=}\ e\ \mathsf{in}\ L_2]$ where $\zeta \notin \mathit{btv}(L_2)$
     (b) By (a) and RT-Trigger: $LCT[\tau_0] = L_1[\mathsf{lazy}\ \langle \zeta, x\rangle\ \mathsf{=}\ e\ \mathsf{in}\ L_2CT[\zeta]] \longrightarrow L_1[\mathsf{let}\ \langle \alpha, x\rangle\ \mathsf{=}\ e\ \mathsf{in}\ (L_2CT[\tau_0])[\zeta := \alpha]]$
   - Case $\tau_0 = \tau_1 \to \tau_2$: not possible
   - Case $\tau_0 = \tau_1 \times \tau_2$: not possible
   - Case $\tau_0 = \forall \alpha{:}\kappa.\tau_0'$: not possible
   - Case $\tau_0 = \exists \alpha{:}\kappa.\tau_0'$: not possible
   - Case $\tau_0 = \lambda \alpha{:}\kappa.\tau_0'$: not possible
   - Case $\tau_0 = \tau_1\ \tau_2$:
     – Subcase $\tau_1 = \lambda \alpha{:}\kappa.\tau_1'$:

(a) By RT-App: $LCT[\tau_0] = LCT[(\lambda\alpha{:}\kappa.\tau_1')\ \tau_2] \longrightarrow LCT[\tau_1'[\alpha := \tau_2]]$

- Subcase τ_1 is not an abstraction:
  (a) By assumption and Context Elimination: $\Gamma \vdash CT[\tau_1\ \tau_2] : \tau'$
  (b) By (a) and Type Context Elimination: $\Gamma, \Gamma' \vdash T[\tau_1\ \tau_2] : \kappa$
  (c) By (b) and Type Context Elimination: $\Gamma, \Gamma' \vdash \tau_1\ \tau_2 : \kappa''$
  (d) By (c) and inversion of K-App: $\Gamma, \Gamma' \vdash \tau_1 : \kappa' \to \kappa''$
  (e) By (d) and definition of the kinding relation: $\tau_1 = \alpha$ or $\tau_1 = \lambda\alpha{:}\kappa'.\tau_1'$ or $\tau_1 = \tau_1'\ \tau_2'$ or $\tau_1 = \tau_1'.1$ or $\tau_1 = \tau_1'.2$
  (f) $\tau_1 = \alpha$ is not possible since $\alpha\ \tau_2$ is in weak head normal form
  (g) By (e), (f), and subcase assumption: τ_1 is not in weak head normal form
  (h) Let $T' = T[\_\ \tau_2]$. Then: $LCT[\tau_0] = LCT'[\tau_1]$
  (i) By (g), (h), assumption, and induction: $LCT[\tau_0] = LCT'[\tau_1] \longrightarrow e$

- Case $\tau_0 = \langle\tau_1, \tau_2\rangle$: not possible
- Case $\tau_0 = \tau_0'.1$:
  - Subcase $\tau_0' = \langle\tau_1, \tau_2\rangle$:
    (a) By RT-Proj1: $LCT[\tau_0] = LCT[\langle\tau_1, \tau_2\rangle.1] \longrightarrow LCT[\tau_1]$
  - Subcase τ_0' is not a type pair:
    (a) By assumption and Context Elimination: $\Gamma \vdash CT[\tau_0'.1] : \tau'$
    (b) By (a) and Type Context Elimination: $\Gamma, \Gamma' \vdash T[\tau_0'.1] : \kappa$
    (c) By (b) and Type Context Elimination: $\Gamma, \Gamma' \vdash \tau_0'.1 : \kappa'$
    (d) By (c) and inversion of K-Proj1: $\Gamma \vdash \tau_0' : \kappa' \times \kappa_2$
    (e) By (d) and definition of the kinding relation: $\tau_0' = \alpha$ or $\tau_0' = \langle\tau_1, \tau_2\rangle$ or $\tau_0' = \tau_1'\ \tau_2'$ or $\tau_0' = \tau_1'.1$ or $\tau_0' = \tau_1'.2$
    (f) $\tau_0' = \alpha$ is not possible since $\alpha.1$ is in weak head normal form
    (g) By (e), (f), and subcase assumption: τ_0' is not in weak head normal form
    (h) Let $T' = T[\_.1]$. Then: $LCT[\tau_0] = LCT'[\tau_0']$
    (i) By (g), (h), assumption, and induction: $LCT[\tau_0] = LCT'[\tau_0'] \longrightarrow e$
- Case $\tau_0 = \tau_0'.2$:
  - Subcase $\tau_0' = \langle\tau_1, \tau_2\rangle$:
    (a) By RT-Proj1: $LCT[\tau_0] = LCT[\langle\tau_1, \tau_2\rangle.2] \longrightarrow LCT[\tau_2]$
  - Subcase τ_0' is not a type pair:
    (a) By assumption and Context Elimination: $\Gamma \vdash CT[\tau_0'.2] : \tau'$
    (b) By (a) and Type Context Elimination: $\Gamma, \Gamma' \vdash T[\tau_0'.2] : \kappa$
    (c) By (b) and Type Context Elimination: $\Gamma, \Gamma' \vdash \tau_0'.2 : \kappa'$
    (d) By (c) and inversion of K-Proj2: $\Gamma \vdash \tau_0' : \kappa_1 \times \kappa'$
    (e) By (d) and definition of the kinding relation: $\tau_0' = \alpha$ or $\tau_0' = \langle\tau_1, \tau_2\rangle$ or $\tau_0' = \tau_1'\ \tau_2'$ or $\tau_0' = \tau_1'.1$ or $\tau_0' = \tau_1'.2$
    (f) $\tau_0' = \alpha$ is not possible since $\alpha.2$ is in weak head normal form
    (g) By (e), (f), and subcase assumption: τ_0' is not in weak head normal form
    (h) Let $T' = T[\_.2]$. Then: $LCT[\tau_0] = LCT'[\tau_0']$
    (i) By (g), (h), assumption, and induction: $LCT[\tau_0] = LCT'[\tau_0'] \longrightarrow e$

2. Let $e = LP[p][p]$.

- Case $P = B$:

(a) Then: $(P, p, p) = (B, p, p)$

- Case $P = P'[\_.1][\_.1]$:

  (a) By induction: there exists a decomposition $(B, \tau_1, \tau_2) \ll (P', p.1, p.1)$ or $(P', p.1, p.1)$ is equal to or deeper than some decomposition (B, p', p')

    – If $(B, \tau_1, \tau_2) \ll (P', p.1, p.1)$, then also $(B, \tau_1, \tau_2) \ll (P, p, p)$, since (P, p, p) is deeper than $(P', p.1, p.1)$

    – If $(P', p.1, p.1)$ is equal to or deeper than some decomposition (B, p', p'), then (P, p, p) is deeper than (B, p', p'), since (P, p, p) is deeper than $(P', p.1, p.1)$

- Case $P = P'[\_.2][\_.2]$: analogous
- Case $P = P'[\_\ \tau_1][\_\ \tau_2]$:

  (a) Let $B = P'[p\ \_][p\ \_]$. Then: $B[\tau_1][\tau_2] = P[p][p]$

  (b) By (a) and Weight Property: $(B, \tau_1, \tau_2) \ll (P, p, p)$

3. Let $e = LB[\nu][\nu]$.

- Case $B = \mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$:

  (a) By R-Case1: $e \longrightarrow L[e_1[x := v]]$

- Case $B = B'[\_ \to \tau_1][\_ \to \tau_2]$:

  (a) Let $B'' = B'[\nu \to \_][\nu \to \_]$. Then: $LB[\nu][\nu] = LB''[\tau_1][\tau_2]$

  (b) By (a) and Weight Property: $(B'', \tau_1, \tau_2) \ll (B, \nu, \nu)$

- Case $B = B'[\nu' \to \_][\nu' \to \_]$:

  (a) By induction: $LB'[\nu' \to \nu][\nu' \to \nu] \longrightarrow e'$ or there exists a decomposition $(B'', \tau_1, \tau_2) \ll (B', \nu' \to \nu, \nu' \to \nu)$

    – If $LB'[\nu' \to \nu][\nu' \to \nu] \longrightarrow e'$, then: $LB[\nu][\nu] \longrightarrow e'$

    – If $(B'', \tau_1, \tau_2) \ll (B', \nu' \to \nu, \nu' \to \nu)$, then also $(B'', \tau_1, \tau_2) \ll (B, \nu, \nu)$, since (B, ν, ν) is deeper than $(B', \nu' \to \nu, \nu' \to \nu)$

- Case $B = B'[\_ \times \tau_1][\_ \times \tau_2]$:

  (a) Let $B'' = B'[\nu \times \_][\nu \times \_]$. Then: $LB[\nu][\nu] = LB''[\tau_1][\tau_2]$

  (b) By (a) and Weight Property: $(B'', \tau_1, \tau_2) \ll (B, \nu, \nu)$

- Case $B = B'[\nu' \times \_][\nu' \times \_]$:

  (a) By induction: $LB'[\nu' \times \nu][\nu' \times \nu] \longrightarrow e'$ or there exists a decomposition $(B'', \tau_1, \tau_2) \ll (B', \nu' \times \nu, \nu' \times \nu)$

    – If $LB'[\nu' \times \nu][\nu' \times \nu] \longrightarrow e'$, then: $LB[\nu][\nu] \longrightarrow e'$

    – If $(B'', \tau_1, \tau_2) \ll (B', \nu' \times \nu, \nu' \times \nu)$, then also $(B'', \tau_1, \tau_2) \ll (B, \nu, \nu)$, since (B, ν, ν) is deeper than $(B', \nu' \times \nu, \nu' \times \nu)$

- Case $B = B'[\forall\alpha{:}\kappa.\_][\forall\alpha{:}\kappa.\_]$:

  (a) By induction: $LB'[\forall\alpha{:}\kappa.\nu][\forall\alpha{:}\kappa.\nu] \longrightarrow e'$ or there exists a decomposition $(B'', \tau_1, \tau_2) \ll (B', \forall\alpha{:}\kappa.\nu, \forall\alpha{:}\kappa.\nu)$

    – If $LB'[\forall\alpha{:}\kappa.\nu][\forall\alpha{:}\kappa.\nu] \longrightarrow e'$, then: $LB[\nu][\nu] \longrightarrow e'$

    – If $(B'', \tau_1, \tau_2) \ll (B', \forall\alpha{:}\kappa.\nu, \forall\alpha{:}\kappa.\nu)$, then also $(B'', \tau_1, \tau_2) \ll (B, \nu, \nu)$, since (B, ν, ν) is deeper than $(B', \forall\alpha{:}\kappa.\nu, \forall\alpha{:}\kappa.\nu)$

- Case $B = B'[\exists\alpha{:}\kappa.\_][\exists\alpha{:}\kappa.\_]$:

  (a) By induction: $LB'[\exists\alpha{:}\kappa.\nu][\exists\alpha{:}\kappa.\nu] \longrightarrow e'$ or there exists a decomposition $(B'', \tau_1, \tau_2) \ll (B', \exists\alpha{:}\kappa.\nu, \exists\alpha{:}\kappa.\nu)$

    – If $LB'[\exists\alpha{:}\kappa.\nu][\exists\alpha{:}\kappa.\nu] \longrightarrow e'$, then: $LB[\nu][\nu] \longrightarrow e'$

  - If $(B'', \tau_1, \tau_2) \ll (B', \exists\alpha{:}\kappa.\nu, \exists\alpha{:}\kappa.\nu)$, then also $(B'', \tau_1, \tau_2) \ll (B, \nu, \nu)$, since (B, ν, ν) is deeper than $(B', \exists\alpha{:}\kappa.\nu, \exists\alpha{:}\kappa.\nu)$

- Case $B = B'[\lambda\alpha{:}\kappa.\_][\lambda\alpha{:}\kappa.\_]$:

  (a) By induction: $LB'[\lambda\alpha{:}\kappa.\nu][\lambda\alpha{:}\kappa.\nu] \longrightarrow e'$ or there exists a decomposition $(B'', \tau_1, \tau_2) \ll (B', \lambda\alpha{:}\kappa.\nu, \lambda\alpha{:}\kappa.\nu)$
  - If $LB'[\lambda\alpha{:}\kappa.\nu][\lambda\alpha{:}\kappa.\nu] \longrightarrow e'$, then: $LB[\nu][\nu] \longrightarrow e'$
  - If $(B'', \tau_1, \tau_2) \ll (B', \lambda\alpha{:}\kappa.\nu, \lambda\alpha{:}\kappa.\nu)$, then also $(B'', \tau_1, \tau_2) \ll (B, \nu, \nu)$, since (B, ν, ν) is deeper than $(B', \lambda\alpha{:}\kappa.\nu, \lambda\alpha{:}\kappa.\nu)$

- Case $B = P[p\ \_][p\ \_]$:

  (a) By (2): there exists a decomposition $(B', \tau_1, \tau_2) \ll (P, p\ \nu, p\ \nu)$ or $(P, p\ \nu, p\ \nu)$ is equal to or deeper than some decomposition (B', p', p')
  - If there exists a decomposition $(B', \tau_1, \tau_2) \ll (P, p\ \nu, p\ \nu)$, then $(B', \tau_1, \tau_2) \ll (B, \nu, \nu)$, since (B, ν, ν) is a deeper decomposition than $(P, p\ \nu, p\ \nu)$
  - If $(P, p\ \nu, p\ \nu)$ is equal to or deeper than some decomposition (B', p', p'), then by induction: $LB'[p'][p'] \longrightarrow e'$ or there exists a decomposition $(B'', \tau_1', \tau_2') \ll (B', p', p')$
    * If $LB'[p'][p'] \longrightarrow e'$, then: $LB[\nu][\nu] \longrightarrow e'$
    * If there exists a decomposition $(B'', \tau_1', \tau_2') \ll (B', p', p')$, then also $(B'', \tau_1', \tau_2') \ll (B, \nu, \nu)$, since (B, ν, ν) is deeper than (B', p', p')

- Case $B = B'[\langle \_, \tau_1\rangle][\langle \_, \tau_2\rangle]$:

  (a) Let $B'' = B'[\langle \nu, \_\rangle][\langle \nu, \_\rangle]$. Then: $LB[\nu][\nu] = LB''[\tau_1][\tau_2]$

  (b) By (a) and Weight Property: $(B'', \tau_1, \tau_2) \ll (B, \nu, \nu)$

- Case $B = B'[\langle \nu', \_\rangle][\langle \nu', \_\rangle]$:

  (a) By induction: $LB'[\langle \nu', \nu\rangle][\langle \nu', \nu\rangle] \longrightarrow e'$ or there exists a decomposition $(B'', \tau_1, \tau_2) \ll (B', \langle \nu', \nu\rangle, \langle \nu', \nu\rangle)$
  - If $LB'[\langle \nu', \nu\rangle][\langle \nu', \nu\rangle] \longrightarrow e'$, then: $LB[\nu][\nu] \longrightarrow e'$
  - If $(B'', \tau_1, \tau_2) \ll (B', \langle \nu', \nu\rangle, \langle \nu', \nu\rangle)$, then also $(B'', \tau_1, \tau_2) \ll (B, \nu, \nu)$, since (B, ν, ν) is deeper than $(B', \langle \nu', \nu\rangle, \langle \nu', \nu\rangle)$

4. Let $e = LP[q_1][q_2]$.

- Case $q_1 \not\sim q_2$:

  (a) By R-Case2: $e \longrightarrow e'$

- Case $q_1 \sim q_2$:
  - Subcase $q_1 = \alpha = q_2$:

    (a) By (2): there exists a decomposition $(B, \tau_1, \tau_2) \ll (P, \alpha, \alpha)$ or (P, α, α) is equal to or deeper than some decomposition (B', p', p')
    * If $(B, \tau_1, \tau_2) \ll (P, \alpha, \alpha)$, then $\textit{weight}(B, \tau_1, \tau_2) < \textit{weight}(P, \alpha, \alpha)$
    * If (P, α, α) is equal to or deeper than some decomposition (B', p', p'), then by (3): $e \longrightarrow e'$ or there exists a decomposition $(B'', \tau_1', \tau_2') \ll (B', p', p')$
      · If $(B'', \tau_1', \tau_2') \ll (B', p', p')$, then $\textit{weight}(B', \tau_1', \tau_2') < \textit{weight}(P, \alpha, \alpha)$, since (P, α, α) is deeper than (B', p', p').
  - Subcase $q_1 = q_1'.1$ and $q_2 = q_2'.1$:

    (a) Let $P' = P[\_.1][\_.1]$.

    (b) By (a) and induction: $e \longrightarrow e'$ or there exists a decomposition (B, τ_1, τ_2) of $P'[q_1'][q_2']$ with $\textit{weight}(B, \tau_1, \tau_2) < \textit{weight}(P', q_1', q_2')$.

∗ If there exists a decomposition (B, τ_1, τ_2) of $P'[q_1'][q_2']$ with $\mathit{weight}(B, \tau_1, \tau_2) < \mathit{weight}(P', q_1', q_2')$, then $\mathit{weight}(B, \tau_1, \tau_2) < \mathit{weight}(P, q_1, q_2)$, since $\mathit{weight}(P', q_1', q_2') < \mathit{weight}(P, q_1, q_2)$.

5. Let $\cdot \vdash LB[\tau_1][\tau_2] : \tau$.

- Case τ_1 not WHN:
  (a) Let $C = B[\_][\tau_2]$ and $T = \_$. Then: $LB[\tau_1][\tau_2] = LCT[\tau_1]$
  (b) By (a) and (1): $LB[\tau_1][\tau_2] \longrightarrow e$
- Case $\tau_1 = \omega_1$ and τ_2 not WHN:
  (a) Let $C = B[\omega_1][\_]$ and $T = \_$. Then: $LB[\tau_1][\tau_2] = LCT[\tau_2]$
  (b) By (a) and (1): $LB[\tau_1][\tau_2] \longrightarrow e$
- Case $\tau_1 = \omega_1$ and $\tau_2 = \omega_2$ where $\omega_1 \not\sim \omega_2$:
  (a) By R-Case2: $LB[\tau_1][\tau_2] \longrightarrow e$
- Case $\tau_1 = \omega_1$ and $\tau_2 = \omega_2$ where $\omega_1 \sim \omega_2$:
  - Subcase $\omega_1 = q_1$ and $\omega_2 = q_2$:
    (a) Let $P = B$. By (4): $LB[\tau_1][\tau_2] = LP[q_1][q_2] \longrightarrow e$ or $\mathit{weight}(B', \tau_1', \tau_2') < \mathit{weight}(P, q_1, q_2) = \mathit{weight}(B, q_1, q_2)$
      ∗ If $\mathit{weight}(B', \tau_1', \tau_2') < \mathit{weight}(B, q_1, q_2)$, then by induction: $LB[\tau_1][\tau_2] = LB'[\tau_1'][\tau_2'] \longrightarrow e$
  - Subcase $\omega_1 = \tau_{11} \to \tau_{12}$ and $\omega_2 = \tau_{21} \to \tau_{22}$:
    (a) Let $B' = B[\_ \to \tau_{12}][\_ \to \tau_{22}]$. Then: $\mathit{weight}(B', \tau_{11}, \tau_{21}) < \mathit{weight}(B, \tau_1, \tau_2)$
    (b) By (a) and induction: $LB[\tau_1][\tau_2] = LB'[\tau_{11}][\tau_{21}] \longrightarrow e$
  - Subcase $\omega_1 = \tau_{11} \times \tau_{12}$ and $\omega_2 = \tau_{21} \times \tau_{22}$:
    (a) Let $B' = B[\_ \times \tau_{12}][\_ \times \tau_{22}]$. Then: $\mathit{weight}(B', \tau_{11}, \tau_{21}) < \mathit{weight}(B, \tau_1, \tau_2)$
    (b) By (a) and induction: $LB[\tau_1][\tau_2] = LB'[\tau_{11}][\tau_{21}] \longrightarrow e$
  - Subcase $\omega_1 = \forall\alpha{:}\kappa.\tau_1'$ and $\omega_2 = \forall\alpha{:}\kappa.\tau_2'$:
    (a) Let $B' = B[\forall\alpha{:}\kappa.\_][\forall\alpha{:}\kappa.\_]$. Then: $\mathit{weight}(B', \tau_1', \tau_2') < \mathit{weight}(B, \tau_1, \tau_2)$
    (b) By (a) and induction: $LB[\tau_1][\tau_2] = LB'[\tau_1'][\tau_2'] \longrightarrow e$
  - Subcase $\omega_1 = \exists\alpha{:}\kappa.\tau_1'$ and $\omega_2 = \exists\alpha{:}\kappa.\tau_2'$:
    (a) Let $B' = B[\exists\alpha{:}\kappa.\_][\exists\alpha{:}\kappa.\_]$. Then: $\mathit{weight}(B', \tau_1', \tau_2') < \mathit{weight}(B, \tau_1, \tau_2)$
    (b) By (a) and induction: $LB[\tau_1][\tau_2] = LB'[\tau_1'][\tau_2'] \longrightarrow e$
  - Subcase $\omega_1 = \lambda\alpha{:}\kappa.\tau_1'$ and $\omega_2 = \lambda\alpha{:}\kappa.\tau_2'$:
    (a) Let $B' = B[\lambda\alpha{:}\kappa.\_][\lambda\alpha{:}\kappa.\_]$. Then: $\mathit{weight}(B', \tau_1', \tau_2') < \mathit{weight}(B, \tau_1, \tau_2)$
    (b) By (a) and induction: $LB[\tau_1][\tau_2] = LB'[\tau_1'][\tau_2'] \longrightarrow e$
  - Subcase $\omega_1 = \langle\tau_{11}, \tau_{12}\rangle$ and $\omega_2 = \langle\tau_{21}, \tau_{22}\rangle$:
    (a) Let $B' = B[\langle\_, \tau_{12}\rangle][\langle\_, \tau_{22}\rangle]$. Then: $\mathit{weight}(B', \tau_{11}, \tau_{21}) < \mathit{weight}(B, \tau_1, \tau_2)$
    (b) By (a) and induction: $LB[\tau_1][\tau_2] = LB'[\tau_{11}][\tau_{21}] \longrightarrow e$

□

Lemma 19 (Context Extension)**.** *If $L[e] \longrightarrow e'$ and e is not a* lazy *expression, then for all E exists an e'' such that $LE[e] \longrightarrow e''$.*

Proof. By case analysis on the applied reduction rule. Whe show only the cases that differ from the proof for the applicative order reduction strategy.

- Case R-Case2: $e = E'[\mathsf{tcase}\ v{:}\tau_1\ \mathsf{of}\ x{:}\tau_2\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]$ and $e' = LE'[e_2]$ where $\mathsf{tcase}\ v{:}\tau_1\ \mathsf{of}\ x{:}\tau_2\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 = B[\omega][\omega']$ with $\omega \not\sim \omega'$ or $\mathsf{tcase}\ v{:}\tau_1\ \mathsf{of}\ x{:}\tau_2\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 = P[q][q']$ with $q \not\sim q'$
    1. Let $E'' = E[E']$. Then: $LE[e] = LE''[\mathsf{tcase}\ v{:}\tau_1\ \mathsf{of}\ x{:}\tau_2\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]$
    2. By (1) and R-Case2: $LE[e] \longrightarrow LE''[e_1[x := v]]$
- Case RT-App: $e = E'CT[(\lambda\alpha{:}\kappa.\tau_1)\ \tau_2]$ and $e' = E'CT[\tau_1[\alpha := \tau_2]]$
    1. Let $E'' = E[E']$. Then: $LE[e] = LE''CT[(\lambda\alpha{:}\kappa.\tau_1)\ \tau_2]$
    2. By (1) and RT-App: $LE[e] \longrightarrow LE''CT[\tau_1[\alpha := \tau_2]]$
- Case RT-Proj1: $e = E'CT[\langle\tau_1, \tau_2\rangle.1]$ and $e' = E'CT[\tau_1]$
    1. Let $E'' = E[E']$. Then: $LE[e] = LE''CT[\langle\tau_1, \tau_2\rangle.1]$
    2. By (1) and RT-Proj1: $LE[e] \longrightarrow LE''CT[\tau_1]$
- Case RT-Proj2: $e = E'CT[\langle\tau_1, \tau_2\rangle.2]$ and $e' = E'CT[\tau_2]$
    1. Let $E'' = E[E']$. Then: $LE[e] = LE''CT[\langle\tau_1, \tau_2\rangle.2]$
    2. By (1) and RT-Proj2: $LE[e] \longrightarrow LE''CT[\tau_2]$

$\square$

Theorem 4 (Progress). *If $\cdot \vdash L[e] : \tau$ where e is neither a value nor a* lazy *expression, then $L[e] \longrightarrow e'$.*

Proof. By structural induction on e. We show only the cases that differ from the proof for the applicative order reduction strategy.

- Case $e \neq E[\mathsf{lazy}\ \ldots] \wedge e = \mathsf{tcase}\ e_0{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$:
    - Subcase e_0 is not a value:
        1. Let $E_0 = \mathsf{tcase}\ \_{:}\tau_0\ \mathsf{of}\ x{:}\tau_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$.
        2. By (1) and assumption: $\cdot \vdash LE_0[e_0] : \tau$
        3. By (2) and Context Elimination: $\cdot \vdash L[e_0] : \tau_0''$
        4. By assumption: e_0 is not a lazy expression
        5. By (3), (4), and induction: $L[e_0] \longrightarrow e_0'$
        6. By (4), (5), and Context Extension: $L[e] = LE_0[e_0] \longrightarrow e'$
    - Subcase $e_0 = v$:
        1. Let $B = \mathsf{tcase}\ v{:}\_\ \mathsf{of}\ x{:}\_\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$.
        2. By (1) and assumption: $\cdot \vdash LB[\tau_0][\tau_0'] : \tau$
        3. By (2) and Type Progress: $L[e] = LB[\tau_0][\tau_0'] \longrightarrow e'$

$\square$