Type-based Exception Analysis for Non-strict Higher-order Functional Languages with Imprecise Exception Semantics

Ruud Koot* and Jurriaan Hage

Department of Computing and Information Sciences
Utrecht University
{r.koot,j.hage}@uu.nl

Abstract. Most statically typed functional programming languages allow programmers to write partial functions: functions that are not defined on all the elements of their domain as specified by their type. Applying a partial function to a value on which it is not defined will raise a run-time exception, thus in practise well-typed programs can and *do* go wrong. Contemporary compilers for functional languages employ a local and purely syntactic analysis to detect incomplete **case**-expressions—those that do not cover all possible patterns of constructors. As programs often maintain invariants on their data—restricting the potential values of the scrutinee to a subtype of its given or inferred type—many of these incomplete **case**-expressions are harmless. Such an analysis will thus report many false positives, overwhelming the programmer. We develop and prove the correctness of a constraint-based type system that detects harmful sources of partiality by accurately tracing the flow of both exceptions—the manifestation of partiality gone wrong—and ordinary data through the program, as well as the dependencies between them. The latter is crucial for usable precision, but has been omitted from previously published exception analyses.

1 Introduction

Many modern programming languages come equipped with a type system. Type systems attempt to prevent bad behaviour at run-time by rejecting programs that may cause such behaviour at compile-time. As predicting the dynamic behaviour of an arbitrary program is undecidable, type systems—or at least those in languages relying on type inference—will always have to either reject some programs that can never cause bad behaviour, or accept some that do.

A major source of trouble are *partial functions*—functions that are not defined on all the elements of their domain, usually because they cannot be given any sensible definition on some of those elements. Canonical examples include the division operator, which is undefined when its right-hand argument is zero,

* This material is based upon work supported by the Netherlands Organisation for Scientific Research (NWO).

and the *head* function, extracting the first element from a list, which is undefined on the empty list. Outright rejecting such functions does not seem like a reasonable course of action, so we are only left with the possibility of accepting them as well-typed at compile-time and having them raise an *exception* at run-time, when invoked on an element in their domain on which they were left undefined.

Still, we would like to warn the programmer when we accept a correctly typed program that may potentially crash due to an exception at run-time. Most compilers for functional languages will already emit a warning when a function is defined by a non-exhaustive pattern-match, listing the missing patterns. These warnings are generated by a very local and syntactic analysis, however, generating many false positives, distracting the programmer. For example, they will complain the *head* function is missing a clause for the empty list, even if *head* is only ever invoked on a syntactically non-empty list, or not at all. Worse still are spurious warnings for local let-bound functions, for which one can no longer argue the warning is useful, as the function cannot be called from a different context at a future time.

Unlike other exception analyses that have appeared in the literature—which primarily attempt to track uncaught user-thrown or environment-induced exceptions, such as those that could be encountered when reading invalid data from disk—we are first and foremost concerned with accurately tracking the exceptions raised by failed pattern-matches. Therefore, our analysis might equally well be called a *pattern-match analysis*, although it is certainly not strictly limited to one as such. To get results of usable accuracy—eliminating as many false positives as possible—requires carefully keeping track of the data-flow through the program. This additionally improves the accuracy of reporting potential exceptions not related to pattern matching and opens the way for employing the analysis to perform static contract checking.

A prototype implementation and (pen-and-paper, as well as mechanized) proofs of the theorems in this paper are available from `http://www.staff.science.uu.nl/~0422819/tbea/`.

2 Motivation

Many algorithms maintain invariants on the data structures they use, that cannot easily be encoded into their types. These invariants often ensure that certain incomplete **case**-expressions are guaranteed not to cause a pattern-match failure.

2.1 *risers*

An example of such an algorithm is the *risers* function from [1], computing monotonically increasing subsegments of a list:

$$
\begin{array}{l}
risers\ :\ Ord\ \alpha \Rightarrow [\alpha] \to [[\alpha]] \\
risers\ []\ \ \ \ \ \ \ \ \ \ \ \ = [] \\
risers\ [x]\ \ \ \ \ \ \ \ \ \ \ = [[x]] \\
risers\ (x_1 :: x_2 :: xs) = \\
\quad \textbf{if}\ x_1 \leqslant x_2\ \textbf{then}\ (x_1 :: y) :: ys\ \textbf{else}\ [x_1] :: (y :: ys) \\
\qquad \textbf{where}\ (y :: ys) = risers\ (x_2 :: xs)
\end{array}
$$

For example:

$$risers\ [1,3,5,1,2] \longrightarrow^* [[1,3,5],[1,2]].$$

The irrefutable pattern in the **where**-clause in the third alternative of *risers* expects the recursive call to return a non-empty list. A naive analysis might raise a warning here. If we think a bit longer, however, we see that we also pass the recursive call to *risers* a non-empty list. This means we will end up in either the second or third alternative inside the recursive call. Both the second alternative and both branches of the **if**-expression in the third alternative produce a non-empty list, satisfying the assumption we made earlier and allowing us to conclude that this function is total and will never raise a pattern-match failure exception. (Raising or propagating an exception because we pattern-match on exceptional values present in the input is still possible, though.)

2.2 *bitstring*

Another example, from [2], comprises a collection of mathematical operations working on bitstrings: integers encoded as lists of the binary digits 0 and 1, with the least significant bit first. We model bitstrings as the type

$$\textbf{type}\ Bitstring = [\mathbb{Z}]$$

This type is too lenient in that it does not restrict the elements of the lists to the digits 0 and 1. We can however maintain this property as an implicit invariant.

If we now define an addition operation on bitstrings:

$$
\begin{array}{l}
add : Bitstring \to Bitstring \to Bitstring \\
add\ []\ \ \ \ y\ \ \ \ = y \\
add\ x\ \ \ \ \ []\ \ \ \ = x \\
add\ (0 :: x)\ (0 :: y) = 0 :: add\ x\ y \\
add\ (0 :: x)\ (1 :: y) = 1 :: add\ x\ y \\
add\ (1 :: x)\ (0 :: y) = 1 :: add\ x\ y \\
add\ (1 :: x)\ (1 :: y) = 0 :: add\ (add\ [1]\ x)\ y
\end{array}
$$

we see that the patterns in *add* are far from complete. However, if only passed arguments that satisfy the invariant it will neither crash due to a pattern-match failure, nor invalidate the invariant.

2.3 *desugar*

Compilers work with large and complex data types to represent the abstract syntax tree. These data structures must be able to represent all syntactic constructs the parser is able to recognize. This results in an abstract syntax tree that is unnecessarily complex, and too cumbersome for the later stages of the compiler—such as the optimizer—to work with. This problem is resolved by *desugaring* the original abstract syntax tree into a simpler—but semantically equivalent—abstract syntax tree that does not use all of the constructors available in the original abstract syntax tree.

The compiler writer now has a choice between two different options: either write a desugaring stage $desugar : ComplexAST \to SimpleAST$—duplicating most of the data type representing and functions operating on the abstract syntax tree—or take the easy route $desugar : AST \to AST$ and assume certain constructors will no longer be present in the abstract syntax tree at stages of the compiler executed after the desugaring step. The former has all the usual downsides of code duplication—such as having to manually keep multiple data types synchronized—while the latter forgoes many of the advantages of strong typing and type safety: if the compiler pipeline is restructured and one of the stages that was originally assumed to run only after the desugaring suddenly runs before that point the error might only be detected at run-time by a pattern-match failure. A pattern-match analysis should be able to detect such errors statically.

3 Informalities

3.1 Data flow

How would we capture the informal reasoning we used in Section 2.1 to convince ourselves that *risers* does not cause a pattern-match failure using a type system? A reasonable first approach would be to annotate all list types with the kind of list it can be: $\mathbf{N}$ if it must be an empty list, a list that necessarily has a nil-constructor at the head of its spine; $\mathbf{C}$ if it must be a non-empty list having a cons-constructor at its head; $\mathbf{N} \sqcup \mathbf{C}$ if it can be either. We can then assign to each of the three individual branches of *risers* the following types:

$$\begin{aligned} risers_1 &: \forall\alpha.\, Ord\ \alpha \Rightarrow [\alpha]^{\mathbf{N}} \to [[\alpha]^{\mathbf{N}\sqcup\mathbf{C}}]^{\mathbf{N}} \\ risers_2 &: \forall\alpha.\, Ord\ \alpha \Rightarrow [\alpha]^{\mathbf{C}} \to [[\alpha]^{\mathbf{N}\sqcup\mathbf{C}}]^{\mathbf{C}} \\ risers_3 &: \forall\alpha.\, Ord\ \alpha \Rightarrow [\alpha]^{\mathbf{C}} \to [[\alpha]^{\mathbf{N}\sqcup\mathbf{C}}]^{\mathbf{C}} \end{aligned}$$

From the three individual branches we may infer:

$$risers : \forall\alpha.\, Ord\ \alpha \Rightarrow [\alpha]^{\mathbf{N}\sqcup\mathbf{C}} \to [[\alpha]^{\mathbf{N}\sqcup\mathbf{C}}]^{\mathbf{N}\sqcup\mathbf{C}}$$

Assigning this type to *risers* will still let us believe that a pattern-match failure may occur in the irrefutable pattern in the **where**-clause, as this type tells us any invocation of *risers*—including the recursive call in the **where**-clause—may evaluate to an empty list. The problem is that $risers_1$—the branch that can never be reached from the call in the **where**-clause—is *poisoning* the overall result. *Polyvariance* (or *property polymorphism*) can rescue us from this precarious situation, however. We can instead assign to each of the branches, and thereby the overall result, the type:

$$risers : \forall\alpha\beta.Ord\ \alpha \Rightarrow [\alpha]^{\beta} \rightarrow [[\alpha]^{\mathbf{N}\sqcup\mathbf{C}}]^{\beta}$$

In the recursive call to *risers* we know the argument passed is a non-empty list, so we can instantiate β to $\mathbf{C}$, informing us that the result of the recursive call will be a non-empty list as well and guaranteeing that the irrefutable pattern-match will succeed.

3.2 Exception flow

The intention of our analysis it to track the exceptions that may be raised during the execution of a program. As with the data flow we express this set of exceptions as an annotation on the type of a program. For example, the program:

$$f\ x = x \div 0$$

should be given the exception type:

$$f : \forall\alpha.\mathbb{Z}^{\alpha} \xrightarrow{\emptyset} \mathbb{Z}^{\alpha\sqcup\mathbf{div\text{-}by\text{-}zero}}$$

This type explains that f is a function accepting an integer as its first and only parameter. As we are working in a call-by-name language, this integer might actually still be a thunk that raises an exception from the set α when evaluated. The program then divides this argument by zero, returning the result. While the result will be of type integer, this operation is almost guaranteed to raise a division-by-zero exception. It is *almost* guaranteed and not completely guaranteed to raise a division-by-zero exception, as the division operator is strict in both of its arguments and might thus force the left-hand side argument to be evaluated before raising the **div-by-zero** exception. This evaluation might then in turn cause an exception from the set α to be raised first. The complete result type is thus an integer with an exception annotation consisting of the union (or *join*) of the exception set α on the argument together with an additional exception **div-by-zero**. Finally, we note that there is an empty exception set annotating the function space constructor, indicating that no exceptions will be raised when evaluating f to closure.

While this approach seems promising at first, it is not immediately adequate for our purpose: detecting potential pattern-match failures that may occur at run time. Consider the following program:

$$head\ (x :: xs) = x$$

After an initial desugaring step, a compiler will translate this program into:

$$head\ xxs = \textbf{case}\ xxs\ \textbf{of}\ \{\,[] \mapsto \textbf{pattern-match-fail}; x :: xs \mapsto x\,\}$$

which can be assigned the exception type:

$$head : \forall \tau\alpha\beta.[\tau^{\alpha}]^{\beta} \xrightarrow{\emptyset} \tau^{\alpha \sqcup \beta \sqcup \textbf{pattern-match-fail}}$$

This type tells us that *head* might always raise a **pattern-match-fail** exception, irrespective of what argument is applied to. Clearly, we won't be able to outperform a simple syntactic analysis in this manner. What we need is a way to introduce a dependency of the exception flow on the data flow of the program, so we can express that *head* will only raise a **pattern-match-fail** if it possible for the argument passed to it to be an empty list. We can do so by introducing conditional constraints into our type system:

$$head : \forall \tau\alpha\beta\gamma.[\tau^{\alpha}]^{\beta} \xrightarrow{\emptyset} \tau^{\alpha \sqcup \beta \sqcup \gamma}\ \textbf{with}\ \{\mathbf{N} \sqsubseteq \beta \Rightarrow \textbf{pattern-match-fail} \sqsubseteq \gamma\}$$

This type explains that *head* will return an element of type τ that might—when inspected—raise any exception present in the elements of the list (α), the spine of the list (β) or from an additional set of exceptions (γ), with the constraint that if the list to which *head* is applied is empty, then this exception set contains the **pattern-match-fail** exception, otherwise it is taken to be empty. (We apologize for the slight abuse of notation—using the annotation β to hold both data and exception-flow information—this will be remedied in the formal type system.)

4 Formalities

4.1 Language

As our language of discourse we take a call-by-name λ-calculus with booleans, integers, pairs, lists, exceptional values, general recursion, let-bindings, pattern-matching, and a set of primitive operators:

$$\begin{aligned} v ::= {} & b \mid n \mid \lightning^{\ell} \mid [] \mid \textbf{close}\ e\ \textbf{in}\ \rho \\ \\ e ::= {} & x \mid v \mid \lambda x.e \mid \textbf{fix}\ f.\ e \mid e_1\ e_2 \\ & \mid \textbf{let}\ x = e_1\ \textbf{in}\ e_2 \mid \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \\ & \mid e_1 \oplus e_2 \mid (e_1, e_2) \mid \textbf{fst}\ e \mid \textbf{snd}\ e \\ & \mid e_1 :: e_2 \mid \textbf{case}\ e_1\ \textbf{of}\ \{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} \\ & \mid \textbf{bind}\ \rho\ \textbf{in}\ e \end{aligned}$$

$$b \in \mathbb{B} \quad n \in \mathbb{Z} \quad f, x \in \textbf{Var} \quad \ell \in \mathcal{P}\,(\textbf{Lbl})$$

Syntax The values v of the language include an exceptional value $\lightning^{\ell}$ where the annotation ℓ is a set of *exception labels*. We leave the exception labels uninterpreted, but an actual implementation will use them to distinguish between distinct exceptions (for example, division-by-zero or a pattern-match failure), as well as to store any additional information necessary to produce informative error messages, such as source locations.

We adopt the following syntactic convention: we denote non-exceptional values by v and possibly exceptional values by v^{ℓ}, where ℓ corresponds to the set of exception labels in case v^{ℓ} is an exceptional value and corresponding to the empty set otherwise.

The **case**-construct in the language is always assumed to be complete. As our primary goal is to detect pattern-match failures produced by incomplete **case**-expressions, we assume any incomplete **case**-expressions written by the programmer has first been appropriately desugared into an equivalent complete **case**-expression by filling out any missing arms in the incomplete **case**-expression with an exceptional value $\lightning^{\mathbf{pattern\text{-}match\text{-}fail}}$ before being passed to the analysis [3].

The **close** and **bind** constructs are only necessary to formalize the small-step operational semantics (Section 4.4) and are assumed to be absent from the program under analysis.

As call-by-name languages model exceptions as exceptional values, instead of exceptional control-flow, we do not need a **throw** or **raise**-construct. As most call-by-name languages omit a **catch**-construct, we shall do so as well.

Static semantics We assume that the program is well-typed according to the canonical monomorphic type system and the underlying type of each subexpression is available to the analysis. In Section 6 we discuss how the analysis can be extended to a language with a polymorphic underlying type system.

4.2 Types

The types $\tau \in \mathbf{Ty}$ of the type system are given by:

$$\tau ::= \alpha \quad | \quad \tau_1 \xrightarrow{\alpha} \tau_2 \quad | \quad \tau_1 \times^{\alpha} \tau_2 \quad | \quad [\tau]^{\alpha}$$

Types are *simply annotated types*, comprised of a single base type consisting of an *annotation variable* $\alpha \in \mathbf{AnnVar}$ and the compound types for functions, pairs and lists, each having its constructor annotated with an annotation variable. Simply annotated types are given meaning in combination with a mapping or substitution from its free annotation variables to a lattice Λ, forming Λ-annotated types.

The function $\lceil \cdot \rceil : \mathbf{Ty} \to \mathbf{AnnVar}$ extracts the outer-most annotation from a simply annotated type τ:

$$\lceil \alpha \rceil = \alpha \qquad \lceil \tau_1 \times^{\alpha} \tau_2 \rceil = \alpha$$
$$\left\lceil \tau_1 \xrightarrow{\alpha} \tau_2 \right\rceil = \alpha \qquad \lceil [\tau]^{\alpha} \rceil = \alpha$$

A type τ can be combined with a constraint set C (Section 4.5) and have some of its free annotation variables quantified over into a type scheme $\sigma \in \mathbf{TySch}$:

$$\sigma ::= \forall \overline{\alpha}.\ \tau \ \mathbf{with}\ C$$

4.3 Environments

An environment Γ binds variables to type schemes and an environment ρ binds variables to expressions:

$$\begin{aligned} \Gamma &::= \epsilon \quad | \quad \Gamma, x : \sigma \\ \rho &::= \epsilon \quad | \quad \rho, x : e \end{aligned}$$

As environments can be permuted, so long as shadowing of variables is not affected, we take the liberty of writing $\Gamma, x : \tau$ and $\rho, x : e$ to match the nearest (most recently bound) variable x in an environment.

4.4 Operational semantics

The operational semantics of the language is given in Figure 1 and models a call-by-name language with an *imprecise exception semantics* [4].

The small-step reduction relation $\rho \ \vdash \ e \longrightarrow \ e'$ has an explicit environment ρ, mapping variables to expressions (not necessarily values). Closures are represented by the operator **close** e **in** ρ, which closes the expression e in the environment ρ. The operator **bind** ρ **in** e binds the free variables in the expression e to expressions in the environment ρ. While similar there is one important distinction between the **close** and the **bind**-construct: a closure **close** e **in** ρ is a value, while a **bind**-expression can still be reduced further.

[E-Var] reduces a variable by looking up the expression bound to it in the environment. As we do not allow for recursive definitions without a mediating **fix**-construct, the variable x is removed from the scope by binding e to the environment ρ, having the (nearest) binding of x removed. [E-Abs] reduces a lambda-abstraction to a closure, closing over its free variables in the current scope. [E-App] first reduces e_1 until it has been evaluated to a function closure, after which [E-AppAbs] performs the application by binding x to e_2 in the scope of e_1. The scope of e_2 is ρ_1, the scope that was closed over, while e_1 gets bound by the outer scope ρ, as it is not necessarily a value and may thus still have free variables that need to remain bound in the outer scope. In case e_1 does not evaluate to a function closure, but to an exceptional value, [E-AppExn1] will first continue reducing the argument e_2 to a (possibly exceptional) value. Once this is done, [E-AppExn2] will reduce the whole application to a single exceptional value with a set of exceptional labels consisting of the union of both the exception labels associated with the applicee as well as the applicant. This behaviour is part of the imprecise exception semantics of our language and will be further motivated in the reduction rules for the **if-then-else** construct. [E-Let] binds x to e_1 in the scope of e_2. [E-Fix] performs a one-step unfolding,

$$\frac{}{\rho, x : e \;\vdash\; x \longrightarrow \textbf{bind}\ \rho\ \textbf{in}\ e} \qquad \frac{}{\rho \;\vdash\; \lambda x.e \longrightarrow \textbf{close}\ \lambda x.e\ \textbf{in}\ \rho}$$

$$\frac{\rho \;\vdash\; e_1 \longrightarrow e_1'}{\rho \;\vdash\; e_1\ e_2 \longrightarrow e_1'\ e_2} \qquad \frac{\rho \;\vdash\; e_2 \longrightarrow e_2'}{\rho \;\vdash\; \lightning^{\ell_1}\ e_2 \longrightarrow \lightning^{\ell_1}\ e_2'} \qquad \frac{}{\rho \;\vdash\; \lightning^{\ell_1}\ v_2^{\ell_2} \longrightarrow \lightning^{\ell_1 \sqcup \ell_2}}$$

$$\frac{}{\rho \;\vdash\; (\textbf{close}\ \lambda x.e_1\ \textbf{in}\ \rho_1)\ e_2 \longrightarrow \textbf{bind}\ (\rho_1, x : \textbf{bind}\ \rho\ \textbf{in}\ e_2)\ \textbf{in}\ e_1}$$

$$\frac{}{\rho \;\vdash\; \textbf{let}\ x = e_1\ \textbf{in}\ e_2 \longrightarrow \textbf{bind}\ (\rho, x : \textbf{bind}\ \rho\ \textbf{in}\ e_1)\ \textbf{in}\ e_2}$$

$$\frac{}{\rho \;\vdash\; \textbf{fix}\ f.\ e \longrightarrow \textbf{bind}\ (\rho, f : \textbf{bind}\ \rho\ \textbf{in fix}\ f.\ e)\ \textbf{in}\ e}$$

$$\frac{\rho \;\vdash\; e_1 \longrightarrow e_1'}{\rho \;\vdash\; e_1 \oplus e_2 \longrightarrow e_1' \oplus e_2} \qquad \frac{\rho \;\vdash\; e_2 \longrightarrow e_2'}{\rho \;\vdash\; e_1 \oplus e_2 \longrightarrow e_1 \oplus e_2'} \qquad \frac{}{\rho \;\vdash\; n_1 \oplus n_2 \longrightarrow [\![n_1 \oplus n_2]\!]}$$

$$\frac{}{\rho \;\vdash\; \lightning^{\ell_1} \oplus n_2 \longrightarrow \lightning^{\ell_1}} \qquad \frac{}{\rho \;\vdash\; n_1 \oplus \lightning^{\ell_2} \longrightarrow \lightning^{\ell_2}} \qquad \frac{}{\rho \;\vdash\; \lightning^{\ell_1} \oplus \lightning^{\ell_2} \longrightarrow \lightning^{\ell_1 \sqcup \ell_2}}$$

$$\frac{\rho \;\vdash\; e_1 \longrightarrow e_1'}{\rho \;\vdash\; \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \longrightarrow \textbf{if}\ e_1'\ \textbf{then}\ e_2\ \textbf{else}\ e_3}$$

$$\frac{}{\rho \;\vdash\; \textbf{if true then}\ e_2\ \textbf{else}\ e_3 \longrightarrow e_2} \qquad \frac{}{\rho \;\vdash\; \textbf{if false then}\ e_2\ \textbf{else}\ e_3 \longrightarrow e_3}$$

$$\frac{\rho \;\vdash\; e_2 \longrightarrow e_2'}{\rho \;\vdash\; \textbf{if}\ \lightning^{\ell_1}\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \longrightarrow \textbf{if}\ \lightning^{\ell_1}\ \textbf{then}\ e_2'\ \textbf{else}\ e_3} \qquad \frac{\rho \;\vdash\; e_3 \longrightarrow e_3'}{\rho \;\vdash\; \textbf{if}\ \lightning^{\ell_1}\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \longrightarrow \textbf{if}\ \lightning^{\ell_1}\ \textbf{then}\ e_2\ \textbf{else}\ e_3'}$$

$$\frac{}{\rho \;\vdash\; \textbf{if}\ \lightning^{\ell_1}\ \textbf{then}\ v_2^{\ell_2}\ \textbf{else}\ v_3^{\ell_3} \longrightarrow \lightning^{\ell_1 \sqcup \ell_2 \sqcup \ell_3}} \qquad \frac{\rho \;\vdash\; e \longrightarrow e'}{\rho \;\vdash\; \textbf{fst}\ e \longrightarrow \textbf{fst}\ e'}$$

$$\frac{}{\rho \;\vdash\; \textbf{fst}\ (\textbf{close}\ (e_1, e_2)\ \textbf{in}\ \rho_1) \longrightarrow \textbf{bind}\ \rho_1\ \textbf{in}\ e_1} \qquad \frac{}{\rho \;\vdash\; \textbf{fst}\ \lightning^{\ell} \longrightarrow \lightning^{\ell}}$$

$$\frac{}{\rho \;\vdash\; (e_1, e_2) \longrightarrow \textbf{close}\ (e_1, e_2)\ \textbf{in}\ \rho} \qquad \frac{}{\rho \;\vdash\; e_1 :: e_2 \longrightarrow \textbf{close}\ e_1 :: e_2\ \textbf{in}\ \rho}$$

$$\frac{\rho \;\vdash\; e_1 \longrightarrow e_1'}{\rho \;\vdash\; \textbf{case}\ e_1\ \textbf{of}\{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} \longrightarrow \textbf{case}\ e_1'\ \textbf{of}\{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\}}$$

$$\frac{}{\rho \;\vdash\; \textbf{case}\ []\ \textbf{of}\{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} \longrightarrow e_2}$$

$$\frac{}{\rho \;\vdash\; \textbf{case}\ \left(\textbf{close}\ e_1 :: e_1'\ \textbf{in}\ \rho_1\right)\ \textbf{of}\{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} \longrightarrow \textbf{bind}\ \left(\rho, x_1 : \textbf{bind}\ \rho_1\ \textbf{in}\ e_1, x_2 : \textbf{bind}\ \rho_1\ \textbf{in}\ e_1'\right)\ \textbf{in}\ e_3}$$

$$\frac{\rho \;\vdash\; e_2 \longrightarrow e_2'}{\rho \;\vdash\; \textbf{case}\ \lightning^{\ell_1}\ \textbf{of}\{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} \longrightarrow \textbf{case}\ \lightning^{\ell_1}\ \textbf{of}\{[] \mapsto e_2'; x_1 :: x_2 \mapsto e_3\}}$$

$$\frac{\rho, x_1 : \lightning^{\emptyset}, x_2 : \lightning^{\emptyset} \;\vdash\; e_3 \longrightarrow e_3'}{\rho \;\vdash\; \textbf{case}\ \lightning^{\ell_1}\ \textbf{of}\{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} \longrightarrow \textbf{case}\ \lightning^{\ell_1}\ \textbf{of}\{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3'\}}$$

$$\frac{}{\rho \;\vdash\; \textbf{case}\ \lightning^{\ell_1}\ \textbf{of}\left\{[] \mapsto v_2^{\ell_2}; x_1 :: x_2 \mapsto v_3^{\ell_3}\right\} \longrightarrow \lightning^{\ell_1 \sqcup \ell_2 \sqcup \ell_3}}$$

$$\frac{\rho_1 \;\vdash\; e_1 \longrightarrow e_1'}{\rho \;\vdash\; \textbf{bind}\ \rho_1\ \textbf{in}\ e_1 \longrightarrow \textbf{bind}\ \rho_1\ \textbf{in}\ e_1'} \qquad \frac{}{\rho \;\vdash\; \textbf{bind}\ \rho_1\ \textbf{in}\ v_1^{\ell_1} \longrightarrow v_1^{\ell_1}}$$

Fig. 1. Operational semantics ($\rho \;\vdash\; e_1 \longrightarrow e_2$)

binding any recursive occurrences of the binder to the original expression in its original scope. To reduce an **if-then-else** expression, [E-If] will start by evaluating the conditional e_1. If it evaluates to either the value **true** or the value **false**, [E-IfTrue] respectively [E-IfFalse], will reduce the whole expression to the appropriate branch e_2 or e_3. In case the conditional e_1 reduces to an exceptional value $\lightning^{\ell}$, the rules [If-Exn1] and [If-Exn2] will continue evaluating both branches of the **if-then-else** expression, as dictated by the imprecise exception semantics. Finally, once both arms have been fully evaluated to possibly exceptional values v^{ℓ_2} and v^{ℓ_3}, [If-Exn3] will join all the exception labels from the conditional and both arms together—remembering that we by convention associate an empty set of exception labels with non-exceptional values—in an exceptional value $\lightning^{\ell_1 \sqcup \ell_2 \sqcup \ell_3}$. This "imprecision" is necessary to validate program transformations, such as **case**-switching, in the presence of distinguishable exceptions:

$$
\begin{array}{l}
\forall e_i.\textbf{if } e_1 \textbf{ then} \\
\quad \textbf{if } e_2 \textbf{ then } e_3 \textbf{ else } e_4 \\
\textbf{else} \\
\quad \textbf{if } e_2 \textbf{ then } e_5 \textbf{ else } e_6 = \textbf{if } e_2 \textbf{ then} \\
\qquad\qquad\qquad\qquad\qquad\qquad \textbf{if } e_1 \textbf{ then } e_3 \textbf{ else } e_5 \\
\qquad\qquad\qquad\qquad\quad \textbf{else} \\
\qquad\qquad\qquad\qquad\qquad\qquad \textbf{if } e_1 \textbf{ then } e_4 \textbf{ else } e_6
\end{array}
$$

Note that [E-Op1] and [E-Op2], as well a several other reduction rules, make the reduction relation non-deterministic, instead of enforcing a left-to-right evaluation order for operators, by requiring its left-hand argument to already be fully evaluated. Not enforcing an evaluation order for operators will allow the compiler to apply optimizing transformations, such as making use of the associativity or commutativity of the operator. If both the left and right-hand side of the operator reduce to a numeric value, [E-OpNum] will reduce the expression to its interpretation $[\![n_1 \oplus n_2]\!]$. If either of, or both, the arguments reduce to an exceptional value, the rules [E-OpExn1], [E-OpExn2] and [E-OpExn3] will propagate the exception labels of all the exceptional values in the expression. The rules [E-Pair] and [E-Cons] wrap the syntactic pair and cons constructors in a closure to make them into values—in a call-by-name language constructors are only evaluated up to weak head normal form (*whnf*) and can still contain unevaluated subexpressions that need to have their free variables closed over in their original scope. [E-Fst] evaluates the argument passed to **fst** to a normal form. If it is an exceptional value, [E-FstExn] will propagate it; if it is a closed pair constructor, [E-FstPair] will project the first argument and bind its free variables in the environment it has been closed over. Accordingly for [E-Snd], [E-SndExn] and [E-SndPair]. [E-Case] will evaluate the scrutinee of a **case**-expression to a normal form. If it evaluates to a nil-constructor, [E-CaseNil] will select the first arm. If it evaluates to a closed cons-constructor, [E-CaseCons] will select the second arm, binding x_1 and x_2 to respectively the first and second component of

the constructor in the environment the constructor was closed over. In case the scrutinee evaluates to an exceptional value [E-CaseExn1], [E-CaseExn2] and [E-CaseExn3] will continue evaluating both arms and gather and propagate all exception labels encountered. In the reduction rule [E-CaseExn-3] we still need to bind x_1 and x_2 to some expression in ρ. As this expression we take $\lightning^{\emptyset}$ in both cases, as it is the least committing value in our system: it is associated with both an empty set of exceptional values, as well as with an empty set of non-exceptional values. [E-Bind1] and [E-Bind2] will continue evaluating any expression e in the given environment ρ_1 until it has been fully reduced to a value, which either contains no free variables, or has them explicitly closed over by a **close**-construct.

4.5 Constraints

A constraint c restricts the Λ-substitutions that may be applied to a simply annotated type to turn it into an Λ-annotated type. A constraint c is a conditional, consisting of a left-hand side g and a right-hand side r:

$$\begin{aligned} c &::= g \Rightarrow r \\ g &::= \Lambda_\iota \sqsubseteq_\iota \alpha \quad | \quad \exists_\iota \alpha \quad | \quad g_1 \vee g_2 \quad | \quad \mathbf{true} \\ r &::= \Lambda_\iota \sqsubseteq_\iota \alpha \quad | \quad \alpha_1 \sqsubseteq_\iota \alpha_2 \quad | \quad \tau_1 \leqslant_\iota \tau_2 \end{aligned}$$

The left-hand side g of a conditional constraint, its *guard*, consists of a disjunction of *atomic guards* $\Lambda_\iota \sqsubseteq_\iota \alpha$, relating an element of the lattice Λ_ι to an annotation variable α, and *non-emptiness guards* $\exists_\iota \alpha$, a predicate on the annotation variable α.

The right-hand side r of a conditional or unconditional constraint can either be an *atomic constraint* $\Lambda_\iota \sqsubseteq_\iota \alpha$, relating an element of the lattice Λ_ι to an annotation variable α, or an atomic constraint $\alpha_1 \sqsubseteq_\iota \alpha_2$, relating two annotation variables, or it can be a *structural constraint* $\tau_1 \leqslant_\iota \tau_2$, relating two simply annotated types.

The asymmetry between the allowed forms of the antecedent g and consequent r of the conditional constraints is intentional: they allow constraints to be formed that are expressive enough to build an accurate analysis, but limited enough so as to allow tractable constraint solving. Both allowing constraints of the form $g_1 \vee g_2$ on the right-hand side of a conditional, or allowing constraints of the form $\alpha_1 \sqsubseteq_\iota \alpha_2$ on the left-hand side of a conditional make constraint solving notoriously difficult: the former because it cannot be trivially decomposed into a set of simpler constraints and the latter because it does not behave monotonically under a fixed-point iteration.

The atomic and structural constraint relations are qualified by an index ι, which for the purpose of our analysis can be one of two constants:

$$\iota ::= \delta \quad | \quad \chi$$

Here δ is used to indicate data-flow, while χ indicates exception-flow.

To ease the syntactic burden we freely write constraint expressions indexed by the two constraint indices $\delta\chi$ simultaneously. Formally these should always be read as standing for two separate constraint expressions: one indexed by δ and the other indexed by χ. As none of the constraint expression in this paper contains more than one such paired index, no ambiguities should arise. Furthermore,

$$c^g \quad ::= \quad c \quad | \quad g$$

and constraints $\mathbf{true} \Rightarrow r$ shall be written simply as r.

Constraints are given meaning by the *constraint satisfaction* predicate $\theta \vDash c^g$, which relates a constraint c, g or r to the meaning of its free variables, which are in turn given by a pair of ground substitutions $\theta = \langle \theta_\delta, \theta_\chi \rangle$:

$$\frac{\theta_\iota \alpha_1 \sqsubseteq_\iota \theta_\iota \alpha_2}{\theta \vDash \alpha_1 \sqsubseteq_\iota \alpha_2}\ \text{CM-Var} \quad \frac{\Lambda_\iota \sqsubseteq_\iota \theta_\iota \alpha}{\theta \vDash \Lambda_\iota \sqsubseteq_\iota \alpha}\ \text{CM-Con} \quad \frac{}{\theta \vDash \mathbf{true}}\ \text{CM-True}$$

$$\frac{\theta \vDash g \Rightarrow \theta \vDash r}{\theta \vDash g \Rightarrow r}\ \text{CM-Impl} \quad \frac{\exists \ell \in \Lambda_\iota : \ell \sqsubseteq_\iota \theta_\iota \alpha}{\theta \vDash \exists_\iota \alpha}\ \text{CM-Exists}$$

$$\frac{\theta \vDash g_1}{\theta \vDash g_1 \vee g_2}\ \text{CM-Left} \quad \frac{\theta \vDash g_2}{\theta \vDash g_1 \vee g_2}\ \text{CM-Right} \quad \frac{\theta_\iota \tau_1 \leqslant_\iota \theta_\iota \tau_2}{\theta \vDash \tau_1 \leqslant_\iota \tau_2}\ \text{CM-Sub}$$

Working with constraints in terms of the constraint satisfaction predicate is rather tedious, so we prefer to work with a *constraint entailment* relation $C \Vdash c^g$ and an associated constraint logic:

$$\frac{C, g \Vdash r}{C \Vdash g \Rightarrow r}\ \text{CL-}\Rightarrow\text{I} \quad \frac{C \Vdash g \Rightarrow r}{C, g \Vdash r}\ \text{CL-}\Rightarrow\text{E} \quad \frac{C \Vdash \Lambda_\iota \sqsubseteq_\iota \alpha}{C \Vdash \exists_\iota \alpha}\ \text{CL-}\exists\text{I}$$

$$\frac{C \Vdash g_1}{C \Vdash g_1 \vee g_2}\ \text{CL-}\vee\text{I} \quad \frac{C \Vdash g_1 \vee g_2}{C \Vdash g_2 \vee g_1}\ \text{CL-}\vee\text{C} \quad \frac{C_1 \Vdash c^g}{C_1, C_2 \Vdash c^g}\ \text{CL-Weak}$$

$$\frac{}{C \Vdash \bot_\iota \sqsubseteq_\iota \alpha}\ \text{CL-}\bot \quad \frac{}{C \Vdash \alpha \sqsubseteq_\iota \top_\iota}\ \text{CL-}\top \quad \frac{C \Vdash c_1^g \quad C, c_1^g \Vdash c_2^g}{C \Vdash c_2^g}\ \text{CL-MP}$$

We lift the constraint entailment relation to work on constraint sets $C_1 \Vdash C_2$ in the obvious way.

Theorem 1 (Soundness of constraint logic). *If $\theta \vDash C$ and $C \Vdash D$ then $\theta \vDash D$.*

The subtyping relation is as usual for a type and effect system, thus note the subeffecting of the annotations:

$$\frac{\alpha_1 \sqsubseteq_\iota \alpha_2}{\alpha_1 \leqslant_\iota \alpha_2}\ \text{SA-Base} \quad \frac{\tau_3 \leqslant_\iota \tau_1 \quad \tau_2 \leqslant_\iota \tau_4 \quad \alpha_1 \sqsubseteq_\iota \alpha_2}{\tau_1 \xrightarrow{\alpha_1} \tau_2 \leqslant_\iota \tau_3 \xrightarrow{\alpha_2} \tau_4}\ \text{SA-Fun}$$

$$\frac{\tau_1 \leqslant_\iota \tau_3 \quad \tau_2 \leqslant_\iota \tau_4 \quad \alpha_1 \sqsubseteq_\iota \alpha_2}{\tau_1 \times^{\alpha_1} \tau_2 \leqslant_\iota \tau_3 \times^{\alpha_2} \tau_4}\ \text{SA-Pair} \quad \frac{\tau_1 \leqslant_\iota \tau_2 \quad \alpha_1 \sqsubseteq_\iota \alpha_2}{[\tau_1]^{\alpha_1} \leqslant_\iota [\tau_2]^{\alpha_2}}\ \text{SA-List}$$

$$\frac{}{\tau \leqslant_\iota \tau}\ \text{S-Refl} \quad \frac{\tau_1 \leqslant_\iota \tau_2 \quad \tau_2 \leqslant_\iota \tau_3}{\tau_1 \leqslant_\iota \tau_3}\ \text{S-Trans}$$

4.6 Type system

A normalized type system for exception analysis is given in Figure 2.

[T-Var] combines a lookup in the type environment with instantiation of variables quantified over in the type scheme. [T-Con] and [T-Exn] make sure that any constants and exception literals flow into the top-level annotations of their type. Constants will have to be abstracted into an element of the lattice Λ_δ, using the auxiliary function i, first.

[T-Abs] is standard, with only an additional annotation present on the function-space constructor. [T-App] incorporates a subtyping check between the formal and the actual parameter and flows all exceptions than can be caused by evaluating the function abstraction—as represented by the annotation on the function-space constructor—into the top-level annotation of the resulting type. The final premise flows any exceptions that can be raised by evaluating the argument to weak head normal form to the top-level annotation on the result type if it is possible for the function that the argument is applied to, to evaluate to an exceptional value. This is necessary to soundly model the imprecise exception semantics. [T-Fix] and [T-Let] are the conventional rules for recursion and polymorphic let-bindings with constrained types.

[T-If] uses subtyping to ensure that the exceptional and non-exceptional values of both branches, as well as any exceptions that can occur while evaluating the conditional are propagated to the resulting type. Additionally, conditional constraints are used to ensure that only reachable branches will contribute to the resulting type. A branch is considered reachable if either the conditional can evaluate to **true** respectively **false**, or because evaluating the conditional can cause an exception and we have to consider both branches as being reachable to validate the **case**-switching transformation.

The typing rule [T-Op] for primitive operators is discussed in Section 4.7.

[T-Pair] is the standard type rule for pairs, except that we add an unconstrained annotation to the pair constructor. [T-Fst] and [T-Snd] are standard type rules for projections from a pair. As they implicitly perform a pattern-match, they have to propagate any exceptions that can occur when evaluating the pair-constructor to the resulting type.

[T-Nil] gives the nil-constructor the type list of τ, with τ unconstrained, and an annotation α indicating the head of the spine of the list can at least contain a nil-constructor. [T-Cons] merges data-flow and exception-flow of the head and elements in the tail of the list, annotates the resulting type with an α indicating the head of the spine of the list can at least contain a cons-constructor, and propagates the exceptions that can occur in the tail of the spine of the list to the resulting type.

[T-Case] is similar to [T-If]. The additional complication lies in the fact that the pattern for a cons-constructor must also bring its two fields into scope and give them an appropriate type. Note how in [T-Cons] and [T-Case] exceptional and non-exceptional values are treated asymmetrically. In the rule [T-Cons] exceptional values in the spine of the tail of the list flow into the result, while we only remember **C** as the non-exceptional value than can occur at the head of the

$$\frac{C \Vdash D\left[\overline{\beta}/\overline{\alpha}\right]}{C; \Gamma, x : \forall \overline{\alpha}.\ \tau \textbf{ with } D \vdash x : \tau\left[\overline{\beta}/\overline{\alpha}\right]}\ \textsc{Var} \quad \frac{C \Vdash i(c) \sqsubseteq_\delta \alpha}{C; \Gamma \vdash c : \alpha}\ \textsc{Con} \quad \frac{C \Vdash \ell \sqsubseteq_\chi \lceil \tau \rceil}{C; \Gamma \vdash \lightning^\ell : \tau}\ \textsc{Exn}$$

$$\frac{C; \Gamma, x : \tau_1 \vdash e : \tau_2}{C; \Gamma \vdash \lambda x.e : \tau_1 \xrightarrow{\alpha} \tau_2}\ \textsc{Abs} \quad \frac{\begin{array}{c} C; \Gamma \vdash e_1 : \tau_1 \xrightarrow{\alpha} \tau_2 \quad C; \Gamma \vdash e_2 : \tau_3 \\ C \Vdash \tau_3 \leqslant_{\delta\chi} \tau_1 \quad C \Vdash \alpha \sqsubseteq_\chi \lceil \tau_2 \rceil \\ C \Vdash \exists_\chi \alpha \Rightarrow \lceil \tau_3 \rceil \sqsubseteq_\chi \lceil \tau_2 \rceil \end{array}}{C; \Gamma \vdash e_1\ e_2 : \tau_2}\ \textsc{App}$$

$$\frac{D; \Gamma \vdash e_1 : \tau_1 \quad C; \Gamma, x : \forall \overline{\alpha}.\ \tau_1 \textbf{ with } D \vdash e_2 : \tau_2 \quad \overline{\alpha} \cap fv(\Gamma) = \emptyset}{C; \Gamma \vdash \textbf{let } x = e_1 \textbf{ in } e_2 : \tau_2}\ \textsc{Let}$$

$$\frac{C \Vdash D\left[\overline{\beta}/\overline{\alpha}\right] \quad D; \Gamma, f : \forall \overline{\alpha}.\ \tau_1 \textbf{ with } D \vdash e : \tau_2 \quad D \Vdash \tau_2 \leqslant_{\delta\chi} \tau_1 \quad \overline{\alpha} \cap fv(\Gamma) = \emptyset}{C; \Gamma \vdash \textbf{fix } f.\ e : \tau_1\left[\overline{\beta}/\overline{\alpha}\right]}\ \textsc{Fix}$$

$$\frac{\begin{array}{c} C; \Gamma \vdash e_1 : \alpha_1 \quad C; \Gamma \vdash e_2 : \tau_2 \quad C; \Gamma \vdash e_3 : \tau_3 \quad C \Vdash \alpha_1 \sqsubseteq_\chi \lceil \tau \rceil \\ C \Vdash \mathbf{T} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_2 \leqslant_{\delta\chi} \tau \quad C \Vdash \mathbf{F} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_3 \leqslant_{\delta\chi} \tau \end{array}}{C; \Gamma \vdash \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 : \tau}\ \textsc{If}$$

$$\frac{C; \Gamma \vdash e_1 : \alpha_1 \quad C; \Gamma \vdash e_2 : \alpha_2 \quad C \Vdash \alpha_1 \sqsubseteq_\chi \alpha \quad C \Vdash \alpha_2 \sqsubseteq_\chi \alpha \quad C \Vdash \omega_\oplus(\alpha_1, \alpha_2, \alpha)}{C; \Gamma \vdash e_1 \oplus e_2 : \alpha}\ \textsc{Op}$$

$$\frac{C; \Gamma \vdash e_1 : \tau_1 \quad C; \Gamma \vdash e_2 : \tau_2}{C; \Gamma \vdash (e_1, e_2) : \tau_1 \times^\alpha \tau_2}\ \textsc{Pair}$$

$$\frac{\begin{array}{c} C; \Gamma \vdash e : \tau_1 \times^\alpha \tau_2 \\ C \Vdash \tau_1 \leqslant_{\delta\chi} \tau \quad C \Vdash \alpha \sqsubseteq_\chi \lceil \tau \rceil \end{array}}{C; \Gamma \vdash \textbf{fst } e : \tau}\ \textsc{Fst} \quad \frac{\begin{array}{c} C; \Gamma \vdash e : \tau_1 \times^\alpha \tau_2 \\ C \Vdash \tau_2 \leqslant_{\delta\chi} \tau \quad C \Vdash \alpha \sqsubseteq_\chi \lceil \tau \rceil \end{array}}{C; \Gamma \vdash \textbf{snd } e : \tau}\ \textsc{Snd}$$

$$\frac{C \Vdash \mathbf{N} \sqsubseteq_\delta \alpha}{C; \Gamma \vdash [] : [\tau]^\alpha}\ \textsc{Nil} \quad \frac{\begin{array}{c} C; \Gamma \vdash e_1 : \tau_1 \quad C; \Gamma \vdash e_2 : [\tau_2]^{\alpha_2} \quad C \Vdash \tau_1 \leqslant_{\delta\chi} \tau \\ C \Vdash \tau_2 \leqslant_{\delta\chi} \tau \quad C \Vdash \mathbf{C} \sqsubseteq_\delta \alpha \quad C \Vdash \alpha_2 \sqsubseteq_\chi \alpha \end{array}}{C; \Gamma \vdash e_1 :: e_2 : [\tau]^\alpha}\ \textsc{Cons}$$

$$\frac{\begin{array}{c} C; \Gamma \vdash e_1 : [\tau_1]^{\alpha_1} \quad C; \Gamma \vdash e_2 : \tau_2 \quad C; \Gamma, x_1 : \tau_1, x_2 : [\tau_1]^\beta \vdash e_3 : \tau_3 \\ C \Vdash \mathbf{N} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_2 \leqslant_{\delta\chi} \tau \quad C \Vdash \mathbf{C} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_3 \leqslant_{\delta\chi} \tau \\ C \Vdash \alpha_1 \sqsubseteq_\chi \lceil \tau \rceil \quad C \Vdash \mathbf{N} \sqcup \mathbf{C} \sqsubseteq_\delta \beta \quad C \Vdash \alpha_1 \sqsubseteq_\chi \beta \end{array}}{C; \Gamma \vdash \textbf{case } e_1 \textbf{ of } \{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} : \tau}\ \textsc{Case}$$

$$\frac{C; \Delta \vdash e : \sigma \quad C \vdash \Delta \bowtie \rho}{C; \Gamma \vdash \textbf{close } e \textbf{ in } \rho : \sigma}\ \textsc{Close} \quad \frac{C; \Delta \vdash e : \sigma \quad C \vdash \Delta \bowtie \rho}{C; \Gamma \vdash \textbf{bind } \rho \textbf{ in } e : \sigma}\ \textsc{Bind}$$

Fig. 2. Normalized type system ($C; \Gamma \vdash e : \sigma$)

spine of the resulting list. This choice means that in [T-Case], while we have more precise information about the head of the list, we have to be *pessimistic* about the shape of the list that gets bound to x_2. Conversely, for exceptional values we have less precise information about the head of the spine of the list. We can, however, be *optimistic* about the exceptional values occuring in the spine of the list that gets bound to x_2. Not being able to do so would be disasterous for the precision of the analysis. Any further pattern-matching on the spine of x_2, whether directly or through applying operations such as *map* or *reverse* to it, would cause the—likely spurious—exceptions to propagate.

Finally, [T-Close] and [T-Bind] type the expression e they close over or bind in an expression environment ρ in under a type environment Δ that types the expression environment ρ. These rules relate the type environments Γ with the expression environments ρ using an *environmental consistency* relation $C \vdash \Gamma \bowtie \rho$:

$$\frac{}{C \vdash \epsilon \bowtie \epsilon}\ \text{EC-Empty} \qquad \frac{C \vdash \Gamma \bowtie \rho \quad C; \Gamma \vdash e : \sigma}{C \vdash \Gamma, x : \sigma \bowtie \rho, x : e}\ \text{EC-Extend}$$

4.7 Primitive operators

The typing rule [T-Op] for primitive operators relies on an auxiliary function ω that assigns a constraint set for each operator, giving its abstract interpretation. For an addition operator + one can take the constraint set:

$$\omega_+(\alpha_1, \alpha_2, \alpha) \stackrel{\text{def}}{=} \{\top_{\mathbb{Z}} \sqsubseteq_\delta \alpha\}$$

or the more precise:

$$\omega_+(\alpha_1, \alpha_2, \alpha) \stackrel{\text{def}}{=} \left\{ \begin{array}{c} \mathbf{-} \sqsubseteq_\delta \alpha_1 \Rightarrow \mathbf{-} \sqsubseteq_\delta \alpha \\ \mathbf{-} \sqsubseteq_\delta \alpha_2 \Rightarrow \mathbf{-} \sqsubseteq_\delta \alpha \\ \mathbf{0} \sqsubseteq_\delta \alpha \\ \mathbf{+} \sqsubseteq_\delta \alpha_1 \Rightarrow \mathbf{+} \sqsubseteq_\delta \alpha \\ \mathbf{+} \sqsubseteq_\delta \alpha_2 \Rightarrow \mathbf{+} \sqsubseteq_\delta \alpha \end{array} \right\}$$

If we would extend our constraints to allow conjunctions on the left-hand side of conditionals we can even get rid of the spurious **0**'s:

$$\omega_+(\alpha_1, \alpha_2, \alpha) \stackrel{\text{def}}{=} \left\{ \begin{array}{c} \mathbf{-} \sqsubseteq_\delta \alpha_1 \Rightarrow \mathbf{-} \sqsubseteq_\delta \alpha \\ \mathbf{-} \sqsubseteq_\delta \alpha_2 \Rightarrow \mathbf{-} \sqsubseteq_\delta \alpha \\ \mathbf{-} \sqsubseteq_\delta \alpha_1 \wedge \mathbf{+} \sqsubseteq_\delta \alpha_2 \Rightarrow \mathbf{0} \sqsubseteq_\delta \alpha \\ \mathbf{0} \sqsubseteq_\delta \alpha_1 \wedge \mathbf{0} \sqsubseteq_\delta \alpha_2 \Rightarrow \mathbf{0} \sqsubseteq_\delta \alpha \\ \mathbf{+} \sqsubseteq_\delta \alpha_1 \wedge \mathbf{-} \sqsubseteq_\delta \alpha_2 \Rightarrow \mathbf{0} \sqsubseteq_\delta \alpha \\ \mathbf{+} \sqsubseteq_\delta \alpha_1 \Rightarrow \mathbf{+} \sqsubseteq_\delta \alpha \\ \mathbf{+} \sqsubseteq_\delta \alpha_2 \Rightarrow \mathbf{+} \sqsubseteq_\delta \alpha \end{array} \right\}$$

Similarly, we are able to detect division-by-zero exceptions caused by an integer division operator $\div$:

$$\omega_{\div}(\alpha_1, \alpha_2, \alpha) \stackrel{\text{def}}{=} \left\{ \begin{array}{c} \mathbf{0} \sqsubseteq_\delta \alpha_2 \Rightarrow \{\textbf{div-by-0}\} \sqsubseteq_\chi \alpha \\ \text{-} \sqsubseteq_\delta \alpha_1 \wedge \text{-} \sqsubseteq_\delta \alpha_2 \Rightarrow + \sqsubseteq_\delta \alpha \\ \text{-} \sqsubseteq_\delta \alpha_1 \wedge + \sqsubseteq_\delta \alpha_2 \Rightarrow \text{-} \sqsubseteq_\delta \alpha \\ + \sqsubseteq_\delta \alpha_1 \wedge \text{-} \sqsubseteq_\delta \alpha_2 \Rightarrow \text{-} \sqsubseteq_\delta \alpha \\ + \sqsubseteq_\delta \alpha_1 \wedge + \sqsubseteq_\delta \alpha_2 \Rightarrow + \sqsubseteq_\delta \alpha \\ \mathbf{0} \sqsubseteq_\delta \alpha \end{array} \right\}$$

We do impose two restrictions on the operator constraint sets; they need to be *consistent* and *monotonic*:

Definition 1. *An operator constraint set $\omega_\oplus$ is said to be* consistent *with respect to an operator interpretation $[\![\cdot \oplus \cdot]\!]$ if, whenever $C; \Gamma \vdash n_1 : \alpha_1$, $C; \Gamma \vdash n_2 : \alpha_2$, and $C \Vdash \omega_\oplus(\alpha_1, \alpha_2, \alpha)$ then $C; \Gamma \vdash [\![n_1 \oplus n_2]\!] : \alpha'$ with $C \Vdash \alpha' \leqslant_{\delta\chi} \alpha$ for some α'.*

This restriction states that the interpretation of an operator and its abstract interpretation by the operator constraint set should coincide. We need one slightly more technical restriction to be able to prove our system sound:

Definition 2. *An operator constraint set $\omega_\oplus$ is said to be* monotonic *if, whenever $C \Vdash \omega_\oplus(\alpha_1, \alpha_2, \alpha)$ and $C \Vdash \alpha_1' \sqsubseteq_{\delta\chi} \alpha_1$, $C \Vdash \alpha_2' \sqsubseteq_{\delta\chi} \alpha_2$, $C \Vdash \alpha \sqsubseteq_{\delta\chi} \alpha'$ then $C \Vdash \omega_\oplus(\alpha_1', \alpha_2', \alpha')$.*

Informally this means that operator constraint sets can only let the result of evaluating an operator and its operands depend on the values of those operands and not the other way around: the operator constraint sets must respect the fact that we are defining a *forwards* analysis.

4.8 Metatheory

Three theorems imply the correctness of the analysis:

Theorem 2 (Conservative extension). *If e is well-typed in the underlying type system, then it can be given a type in the annotated type system.*

Theorem 3 (Progress). *If $C; \Gamma \vdash e : \sigma$ then either e is a value or there exist an e', such that for any ρ with $C \vdash \Gamma \bowtie \rho$ we have $\rho \vdash e \longrightarrow e'$.*

Theorem 4 (Preservation). *If $C; \Gamma \vdash e : \sigma_1$, $\rho \vdash e \longrightarrow e'$ and $C \vdash \Gamma \bowtie \rho$ then $C; \Gamma \vdash e' : \sigma_2$ with $C \Vdash \sigma_2 \leqslant_{\delta\chi} \sigma_1$.*

5 Constraint solving

The analysis can be implemented as a three-stage type inference process. In the first stage we invoke a standard Hindley–Milner type inference algorithm to make sure the input program is well-typed and to give the second stage access to the underlying types of all subexpressions. The second stage generates a set of constraints and the third stage solves those constraints.

The constraint solver $\mathcal{S}$ takes a constraint set C and produces a substitution θ that solves it. The solver assumes that all subtyping constaints ($\leqslant_\iota$) have been decomposed into atomic constrains ($\sqsubseteq_\iota$) using the syntax-directed part of the subtyping relation (SA) as a decomposition algorithm.

$$
\begin{array}{l}
\mathcal{S} \quad : \mathcal{P}\ \mathbf{Constr} \to \mathbf{Subst} \\
\mathcal{S}\ C = \mathbf{do\ for\ each}\ \alpha \in \mathit{fv}\ C\ \mathbf{do} \\
\qquad\qquad \theta_\delta[\alpha] \leftarrow \emptyset \\
\qquad\qquad \theta_\chi[\alpha] \leftarrow \emptyset \\
\qquad\qquad D[\alpha] \leftarrow \emptyset \\
\qquad\quad \mathbf{for\ each}\ c \in C\ \mathbf{do} \\
\qquad\qquad D \leftarrow \mathcal{D}\ D\ c \\
\qquad\quad W \leftarrow C \\
\qquad\quad \mathbf{while}\ W \not\equiv \emptyset\ \mathbf{do} \\
\qquad\qquad \{g \Rightarrow r\} \cup W \leftarrow W \\
\qquad\qquad \mathbf{if}\ \langle \theta_\delta, \theta_\chi \rangle \vDash g\ \mathbf{then} \\
\qquad\qquad\quad \mathbf{case}\ r\ \mathbf{of} \\
\qquad\qquad\qquad \Lambda_\iota \sqsubseteq_\iota \alpha \ \mapsto \mathbf{if}\ \theta_\iota[\alpha] \not\equiv \theta_\iota[\alpha] \sqcup \Lambda_\iota\ \mathbf{then} \\
\qquad\qquad\qquad\qquad\qquad \theta_\iota[\alpha] \leftarrow \theta_\iota[\alpha] \sqcup \Lambda_\iota \\
\qquad\qquad\qquad\qquad\qquad W \leftarrow W \cup D[\alpha] \\
\qquad\qquad\qquad \alpha_1 \sqsubseteq_\iota \alpha_2 \mapsto \mathbf{if}\ \theta_\iota[\alpha] \not\equiv \theta_\iota[\alpha_1] \sqcup \theta_\iota[\alpha_2]\ \mathbf{then} \\
\qquad\qquad\qquad\qquad\qquad \theta_\iota[\alpha_1] \leftarrow \theta_\iota[\alpha_1] \cup \theta_\iota[\alpha_2] \\
\qquad\qquad\qquad\qquad\qquad W \leftarrow W \cup D[\alpha_1] \\
\qquad\quad \mathbf{return}\ \langle \theta_\delta, \theta_\chi \rangle
\end{array}
$$

The solver $\mathcal{S}$ relies on a recursive subroutine $\mathcal{D}$ that performs a dependency analysis on the constraints, expressed as a dependency map $D : \mathbf{Var} \hookrightarrow \mathbf{Constr}$. There is an entry in the dependency map for each variable for which we are solving, associating it with a set of constraints that may need to be reconsidered by the constraint solver whenever the variable's value is updated.

$$
\begin{array}{l}
\mathcal{D} \quad : (\mathbf{Var} \hookrightarrow \mathbf{Constr}) \to \mathbf{Constr} \to (\mathbf{Var} \hookrightarrow \mathbf{Constr}) \\
\mathcal{D}\ D\ c = \mathbf{do\ case}\ c\ \mathbf{of} \\
\qquad\qquad g \Rightarrow \alpha_1 \sqsubseteq_\iota \alpha_2 \mapsto D[\alpha_1] \leftarrow D[\alpha_1] \cup c \\
\qquad\quad \mathbf{case}\ c\ \mathbf{of} \\
\qquad\qquad \Lambda_\iota \sqsubseteq_\iota \alpha \Rightarrow r \ \mapsto D[\alpha] \leftarrow D[\alpha] \cup c \\
\qquad\qquad \exists_\iota \alpha \Rightarrow r \quad\ \mapsto D[\alpha] \leftarrow D[\alpha] \cup c \\
\qquad\qquad g_1 \vee g_2 \Rightarrow r \ \mapsto \mathbf{do}\ D \leftarrow \mathcal{D}\ D\ (g_1 \Rightarrow r) \\
\qquad\qquad\qquad\qquad\qquad\quad D \leftarrow \mathcal{D}\ D\ (g_2 \Rightarrow r) \\
\qquad\quad \mathbf{return}\ D
\end{array}
$$

6 Polymorphism

Consider the polymorphic function *apply*:

$$\begin{array}{l} \mathit{apply} : \forall\alpha\beta.(\alpha \to \beta) \to \alpha \to \beta \\ \mathit{apply}\ f\ x = f\ x \end{array}$$

As we cannot inspect an expression with a polymorphic type other than by the exceptions that it may raise when forced to weak head normal form, it is sufficient to treat them as any other base type.

Thus, the function *apply* will be given the following type by our analysis:

$$\forall\alpha\beta\gamma\delta\epsilon\zeta.\ \left(\alpha \xrightarrow{\delta} \beta\right) \xrightarrow{\epsilon} \gamma \xrightarrow{\zeta} \beta \ \textbf{with}\ \{\gamma \leqslant_{\delta\chi} \alpha, \delta \sqsubseteq_{\chi} \beta, \exists_{\chi}\delta \Rightarrow \gamma \sqsubseteq_{\chi} \beta\}$$

Care needs to be taken when we instantiate the polymorphic variables of a polymorphic type in the underlying type system. When doing so we must also simultaneously update the corresponding polyvariant type used by the analysis.

For example, instantiating the polymorphic underlying type of *apply* with $[\alpha \mapsto \mathbb{Z} \times \mathbb{Z}, \beta \mapsto \mathbb{B} \times \mathbb{B}]$ will give rise to the instantiation

$$\left[\alpha \mapsto \eta \times^{\alpha} \theta, \beta \mapsto \iota \times^{\beta} \kappa, \gamma \mapsto \lambda \times^{\gamma} \mu\right]$$

of the polyvariant type used in the analysis:

$$\begin{array}{l} \forall\alpha\beta\gamma\delta\epsilon\zeta\eta\theta\iota\kappa\lambda\mu.\ \left(\eta \times^{\alpha} \theta \xrightarrow{\delta} \iota \xrightarrow{\beta} \kappa\right) \xrightarrow{\epsilon} \lambda \times^{\gamma} \mu \xrightarrow{\zeta} \iota \xrightarrow{\beta} \kappa \\ \qquad\qquad \textbf{with}\ \{\lambda \times^{\gamma} \mu \leqslant_{\delta\chi} \eta \times^{\alpha} \theta, \delta \sqsubseteq_{\chi} \beta, \exists_{\chi}\delta \Rightarrow \gamma \sqsubseteq_{\chi} \beta\} \end{array}$$

In the general case we need to:

1. Generate an *almost* fresh linear type for each of the polyvariant variables positionally corresponding to a polymorphic variable that has been instantiated in the underlying type. This type is almost, but not entirely, fresh as we do need to preserve the original variable as the top-level annotation on the new fresh type. Thus, for a type substitution $\alpha \mapsto \tau$, we need $\lceil\tau\rceil = \alpha$. Note that multiple polyvariant type variables in the type inferred by the analysis might correspond to a single polymorphic type variable that is being instantiated in the underlying type. In the example given above both the polyvariant variables α and γ correspond positionally to the polymorphic type variable α.
2. We need to quantify over all the fresh variables introduced.
3. We need to apply the type substitution to the constraint set, but only to the subtype constraints ($\leqslant_{\iota}$) and not the atomic constraints ($\sqsubseteq_{\iota}$), as the latter were generated solely to relate top-level annotations to each other.

Due to subtype constraints being treated differently from atomic constraints, the analysis now must also be careful not to decompose the subtype constraints into atomic constraints too early. While an implementation will already want to postpone this until right before sending all the gathered constraints to the constraint solver for performance reasons, this now also becomes important for correctness.

7 Related Work

Catch and Dialyzer Mitchell's case totality checker Catch [1] employs a first-order backwards analysis, inferring preconditions on functions under which it is guaranteed that no exceptions will be raised or pattern-match failures will occur. To analyze Haskell programs, a specially crafted and incomplete defunctionalization step is required. In contrast, our forwards analysis is type-driven and will naturally work on higher-order programs. Catch assumes functions are strict in all their arguments, while our analysis tries to model the call-by-name semantics more accurately. The Dialyzer discrepancy analyzer for Erlang [5] has a dual notion of soundness: Dialyzer will warn about function applications that are guaranteed to generate an exception.

Exception analyses Several exception analyses have been described in the literature, primarily targeting the detection of uncaught exceptions in ML. The exception analysis in [6] is based on abstract interpretation. [7] and [8] describe type-based exception analyses, neither are very precise. The row-based type system for exception analysis described in [9] does contain a data-flow analysis component, although one that is specialized towards tracking value-carrying exceptions instead of value-dependent exceptions and thus employs less precise unification methods instead of subtyping. [10] developed the first exception analysis for non-strict languages. It is a type-based analysis using Boolean constraints and, although it does not take data flow into account, has a similar flavour to our system.

Conditional constraints In [11] conditional constraints are used to model branches in **case**-expressions for a dead-code analysis. Constraint-based k-CFA analyses [12], although traditionally not formulated as a type system, can use conditional constraints to let the control flow depend on the data flow. [13] uses conditional constraints to formulate a soft-typing system for dynamic languages. [14] developed an expressive constraint-based type system incorporating subtyping, conditional constraints and rows and applied it to several inference problems, including accurate pattern-matchings.

Refinement types The refinement type system in [2] attempts to assign more accurate, refined types to already well-typed ML programs using *union* and *intersection types* with the detection of potential pattern-match failures and reduction of warnings about incomplete patterns as one of its goals. There has been a long line of work exploring various approaches of *refinement types* in the sense of using *dependent types* to specify contracts. Dependent ML [15] purposely limits the expressiveness of contracts, so contract checking—although not inference—remains decidable. [16] uses symbolic evaluation to check contracts on Haskell programs. [17] developed a framework for *hybrid type checking*, where the checking of contracts that could not be proven to either always hold or be violated at compile-time are deferred until run-time. The work on *liquid types* [18] attempts to infer refinements using predicate abstraction. MoCHi [19] employs higher-order model checking. The static contract checker HALO [20] translates a Haskell program and its contracts into first-order predicates, which can be proven by SMT solver.

Acknowledgements

The authors would like to thank Stefan Holdermans for his contributions in the earlier stages of this research, Neil Mitchell for answering several questions related to Catch, and the anonymous reviewers for helpful comments on an earlier version of this paper.

References

1. Mitchell, N., Runciman, C.: Not all patterns, but enough: an automatic verifier for partial but sufficient pattern matching. Haskell '08, ACM (2008) 49–60
2. Freeman, T., Pfenning, F.: Refinement types for ML. PLDI '91, ACM (1991) 268–277
3. Augustsson, L.: Compiling pattern matching. In: Functional Programming Languages and Computer Architecture, Springer-Verlag (1985) 368–381
4. Peyton Jones, S., Reid, A., Henderson, F., Hoare, T., Marlow, S.: A semantics for imprecise exceptions. PLDI '99, ACM (1999) 25–36
5. Lindahl, T., Sagonas, K.: Practical type inference based on success typings. PPDP '06, ACM (2006) 167–178
6. Yi, K.: Compile-time detection of uncaught exceptions in Standard ML programs. Volume 864 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1994) 238–254
7. Guzmán, J.C., Suárez, A.: An extended type system for exceptions. In: Proceedings of the ACM SIGPLAN Workshop on ML and its Applications. (1994) 127–135
8. Fahndrich, M., Foster, J., Cu, J., Aiken, A.: Tracking down exceptions in Standard ML programs. Technical report (1998)
9. Leroy, X., Pessaux, F.: Type-based analysis of uncaught exceptions. ACM Trans. Program. Lang. Syst. **22**(2) (March 2000) 340–377
10. Glynn, K., Stuckey, P.J., Sulzmann, M., Søndergaard, H.: Exception analysis for non-strict languages. ICFP '02, ACM (2002) 98–109
11. Heintze, N.: Set-based analysis of ML programs. LFP '94, ACM (1994) 306–317
12. Shivers, O.: Control flow analysis in Scheme. PLDI '88, ACM (1988) 164–174
13. Aiken, A., Wimmers, E.L., Lakshman, T.K.: Soft typing with conditional types. POPL '94, ACM (1994) 163–173
14. Pottier, F.: A versatile constraint-based type inference system. Nordic J. of Computing **7**(4) (December 2000) 312–347
15. Xi, H.: Dependent ML An approach to practical programming with dependent types. J. Funct. Program. **17**(2) (March 2007) 215–286
16. Xu, D.N., Peyton Jones, S., Claessen, K.: Static contract checking for Haskell. POPL '09, ACM (2009) 41–52
17. Knowles, K., Flanagan, C.: Hybrid type checking. ACM Trans. Program. Lang. Syst. **32**(2) (February 2010) 6:1–6:34
18. Rondon, P.M., Kawaguci, M., Jhala, R.: Liquid types. PLDI '08, ACM (2008) 159–169
19. Kobayashi, N., Sato, R., Unno, H.: Predicate abstraction and CEGAR for higher-order model checking. PLDI '11, ACM (2011) 222–233
20. Vytiniotis, D., Peyton Jones, S., Claessen, K., Rosén, D.: HALO: Haskell to logic through denotational semantics. POPL '13, ACM (2013) 431–442

A Appendix

A.1 Subtyping

Lemma 1. *If* $C \Vdash \tau_1 \leqslant_\iota \tau_2$ *then* $C \Vdash \lceil \tau_1 \rceil \sqsubseteq_\iota \lceil \tau_2 \rceil$.

Proof. By inversion of the subtype relation; the result is immediate in all cases.

A.2 Constraints

Theorem 5 (Soundness of constraint logic). *If* $\theta \vDash C$ *and* $C \Vdash D$ *then* $\theta \vDash D$.

Proof. By induction on the derivation of $C \Vdash D$.

Case [CL-⇒I]: We need to show that if $\theta \vDash C$ and $C \Vdash g \Rightarrow r$ then $\theta \vDash g \Rightarrow r$. The induction hypothesis states that if $\theta \vDash C, g$ and $C, g \Vdash r$ then $\theta \vDash r$. Our assumptions are:

$$\theta \vDash C \tag{1}$$
$$C \Vdash g \Rightarrow r \tag{2}$$
$$C, g \Vdash r \tag{3}$$

We perform a case-split on $\theta \vDash g$.

Subcase "$\theta \vDash g$": We can now apply the induction hypothesis and find that

$$\theta \vDash r \tag{4}$$

We apply [CL-⇒I] using (4) and the assumption introduced in the subcase to obtain

$$\theta \vDash g \Rightarrow r \tag{5}$$

Subcase "$\theta \nvDash g$": We can immediately apply [CL-⇒I] using the assumption introduced in the subcase to obtain the desired result.

Case [CL-⇒E]: We need to show that if $\theta \vDash C, g$ and $C, g \Vdash r$ then $\theta \vDash r$. The induction hypothesis states that if $\theta \vDash C$ and $C \Vdash g \Rightarrow r$ then $\theta \vDash g \to r$. Our assumptions are:

$$\theta \vDash C, g \tag{6}$$
$$C, g \Vdash r \tag{7}$$
$$C \Vdash g \Rightarrow r \tag{8}$$

From (6) it follows that

$$\theta \vDash C \tag{9}$$
$$\theta \vDash g \tag{10}$$

We can now apply the induction hypothesis to find

$$\theta \vDash g \Rightarrow r \tag{11}$$

Applying the inversion lemma to (11) reveals that the last inference rule used in its derivation can only have been [CM-Impl], so we find that

$$\theta \vDash g \Rightarrow \theta \vDash r \tag{12}$$

From (10) and (12) we obtain

$$\theta \vDash r \tag{13}$$

concluding the proof.

Case [CL-$\vee$I]: We need to show that if $\theta \vDash C$ and $C \Vdash g_1 \vee g_2$ then $\theta \vDash g_1 \vee g_2$. The induction hypothesis states that if $\theta \vDash C$ and $C \Vdash g$ then $\theta \vDash g_1$. Our assumptions are:

$$\theta \vDash C \tag{14}$$

$$C \Vdash g_1 \vee g_2 \tag{15}$$

$$C \Vdash g_1 \tag{16}$$

We can immediately apply the induction hypothesis and get

$$\theta \vDash g_1 \tag{17}$$

Applying [CM-Left] to (17) gives us

$$\theta \vDash g_1 \vee g_2 \tag{18}$$

concluding the proof.

Case [CL-$\vee$C]: We need to show that if $\theta \vDash C$ and $C \Vdash g_2 \vee g_1$ then $\theta \vDash g_2 \vee g_1$. The induction hypothesis states that if $\theta \vDash C$ and $C \Vdash g_1 \vee g_2$ and $\theta \vDash g_1 \vee g_2$. Our assumptions are:

$$\theta \vDash C \tag{19}$$

$$C \Vdash g_2 \vee g_1 \tag{20}$$

$$C \Vdash g_1 \vee g_2 \tag{21}$$

We can immediately apply the induction hypothesis, giving us:

$$\theta \vDash g_1 \vee g_2 \tag{22}$$

Applying the inversion lemma to (22) reveals it can either have been constructed by [CM-Left] or otherwise by [CM-Right].

Subcase [CM-Left]: We obtain the additional assumption $\theta \vDash g_1$ and can apply [CM-Right] to get $\theta \vDash g_2 \vee g_1$.

Subcase [CM-Right]: We obtain the additional assumption $\theta \vDash g_2$ and can apply [CM-Left] to get $\theta \vDash g_2 \vee g_1$.

Case [CL-$\exists$I]: We need to show that if $\theta \vDash C$ and $C \Vdash \exists_\iota \alpha$ then $\theta \vDash \exists_\iota \alpha$.

The induction hypothesis states that if $\theta \vDash C$ and $C \Vdash \Lambda_\iota \sqsubseteq_\iota \alpha$ then $\theta \vDash \Lambda_\iota \sqsubseteq_\iota \alpha$.

We can immediately apply the induction hypothesis. Inversion on its result reveals that it can only have been constructed by [CM-Con], giving us $\Lambda_\iota \sqsubseteq_\iota \theta_\iota \alpha$. We can use this to apply [CM-Exists], giving us the desired result.

Case [CL-$\emptyset$]: We need to show that if $\theta \vDash C$ and $C \Vdash \emptyset \sqsubseteq_\iota \alpha$ then $\theta \vDash \emptyset \sqsubseteq_\iota \alpha$.

From the order theoretic axioms it follows that

$$\emptyset \sqsubseteq_\iota \theta_\iota \alpha \tag{23}$$

Applying [CM-Con] to (23) gives us

$$\theta \vDash \emptyset \sqsubseteq_\iota \alpha. \tag{24}$$

Case [CL-Lub]: We need to show that if $\theta \vDash C$ and $C \Vdash \alpha_1 \sqcup \alpha_2$ then $\theta \vDash \alpha_1 \sqcup \alpha_2 \sqsubseteq_\iota \beta$. The induction hypotheses state that:

1. If $\theta \vDash C$ and $C \Vdash \alpha_1 \sqsubseteq_\iota \beta$ then $\theta \vDash \alpha_1 \sqsubseteq_\iota \beta$.
2. If $\theta \vDash C$ and $C \Vdash \alpha_2 \sqsubseteq_\iota \beta$ then $\theta \vDash \alpha_2 \sqsubseteq_\iota \beta$.

Our assumptions are:

$$\theta \vDash C \tag{25}$$
$$C \Vdash \alpha_1 \sqcup \alpha_2 \sqsubseteq -_\iota \beta \tag{26}$$
$$C \Vdash \alpha_1 \sqsubseteq_\iota \beta \tag{27}$$
$$C \Vdash \alpha_2 \sqsubseteq_\iota \beta \tag{28}$$

We can immediately apply the induction hypotheses, giving us:

$$\theta \vDash \alpha_1 \sqsubseteq_\iota \beta \tag{29}$$
$$\theta \vDash \alpha_2 \sqsubseteq_\iota \beta \tag{30}$$

Applying the inversion lemma to both (29) and (30) reveals that they can only have been constructed using [CM-Var], giving us:

$$\theta_\iota \alpha_1 \sqsubseteq_\iota \theta_\iota \beta \tag{31}$$
$$\theta_\iota \alpha_2 \sqsubseteq_\iota \theta_\iota \beta \tag{32}$$

By order theoretic reasoning we find that

$$\theta_\iota \alpha_1 \sqcup \theta_\iota \alpha_2 \sqsubseteq_\iota \theta_\iota \beta \tag{33}$$

As substitutions distribute over lattice operations, we also have:

$$\theta_\iota \left(\alpha_1 \sqcup \alpha_2\right) \sqsubseteq_\iota \theta_\iota \beta \tag{34}$$

Finally, we can apply [CM-Var] to (34) to obtain

$$\theta \vDash \alpha_1 \sqcup \alpha_2 \sqsubseteq_\iota \beta \tag{35}$$

Case [CL-Weak]: We need to show that if $\theta \vDash C_1, C_2$ and $C_1, C_2 \Vdash c$ then $\theta \vDash c$. The induction hypothesis states that if $\theta \vDash c$ and $C_1 \Vdash c$ then $\theta \vDash c$. Our assumptions are:

$$\theta \vDash C_1, C_2 \tag{36}$$
$$C_1, C_2 \Vdash c \tag{37}$$
$$C_1 \Vdash c \tag{38}$$

From the definition of (36) it follows that

$$\theta \vDash C_1 \tag{39}$$

We can now apply the induction hypothesis and obtain

$$\theta \vDash c \tag{40}$$

Case [CL-MP]: We need to show that if $\theta \vDash C$ and $C \Vdash c_2$ then $\theta \vDash c_2$. The induction hypotheses state that:

1. If $\theta \vDash C$ and $C \Vdash c_1$ then $\theta \vDash c_1$.
2. If $\theta \vDash C, c_1$ and $C, c_1 \Vdash c_2$ then $\theta \vDash c_2$.

Our assumptions are:

$$\theta \vDash C \tag{41}$$
$$C \Vdash c_1 \tag{42}$$
$$C, c_1 \vDash c_2 \tag{43}$$

Applying the first induction hypothesis using (41) and (42) gives us:

$$\theta \vDash c_1 \tag{44}$$

We can write (41) and (44) as

$$\theta \vDash C, c_1 \tag{45}$$

Applying the second induction hypothesis using (45) and (43) gives us:

$$\theta \vDash c_2 \tag{46}$$

which is what was asked for.

A.3 Progress

Lemma 2 (Canonical forms). *If v is a value with underlying type $\mathbb{B}$ then $v = \mathbf{true}$, $v = \mathbf{false}$ or $v = \lightning^{\ell}$ for some ℓ. If v is a value with underlying type $\mathbb{Z}$ then $v = n$ or $v = \lightning^{\ell}$. If v is a value of type $\tau_1 \xrightarrow{\alpha} \tau_2$ then $v = \mathbf{close}\ \lambda x.e\ \mathbf{in}\ \rho$ or $v = \lightning^{\ell}$. If v is a value of type $\tau_1 \times^{\alpha} \tau_2$ then $v = \mathbf{close}\ (e_1, e_2)\ \mathbf{in}\ \rho$ or $v = \lightning^{\ell}$. If v is a value of type $[\tau]^{\alpha}$ then $v = []$, $v = \mathbf{close}\ e_1 :: e_2\ \mathbf{in}\ \rho$ or $v = \lightning^{\ell}$.*

Proof. Recall that values can only be of the forms **true**, **false**, n, $\lightning^{\ell}$, [] or **close** e **in** ρ. Except for values of the last form all cases follow either immediately or can be discarded by inversion of the typing relation. We thus proceed to check values of the last form.

We assumed no subexpressions of the form **close** e **in** ρ were present in the original program, so they can only have been introduced by evaluation using rules [E-Abs], [E-Pair] or [E-Cons]. [T-Close] tells us that for **close** e **in** ρ to have underlying type $\mathbb{B}$ or $\mathbb{Z}$, e must be of underlying type $\mathbb{B}$ or $\mathbb{Z}$, respectively. None of the three aforementioned evaluation rules can produce an expression of this type. If v is of either type $\tau_1 \xrightarrow{\alpha} \tau_2$, $\tau_1 \times^{\alpha} \tau_2$ or $[\tau]^{\alpha}$ then an expression of the form **close** e **in** ρ must have been produced by [E-Abs], [E-Pair] or [E-Cons], respectively. In each case e is then of the required form.

Theorem 6 (Progress). *If $C; \Gamma \vdash e : \sigma$ then either e is value or there exist an e', such that for any ρ with $C \vdash \Gamma \bowtie \rho$ we have $\rho \vdash e \longrightarrow e'$.*

Proof. By induction on the typing derivation.

Case [T-Var]: We can make progress by applying [E-Var].

Case [T-Con]: c is a value.

Case [T-Exn]: $\lightning^{\ell}$ is a value.

Case [T-Abs]: We can make progress by applying [E-Abs].

Case [T-App]: Given $C; \Gamma \vdash e_1\ e_2 : \tau_1$, the last of the typing derivation must have been [T-App] and by inversion on the typing relation we get $C; \Gamma \vdash e_1 : \tau_2 \xrightarrow{\alpha} \tau_1$ and $C; \Gamma \vdash e_2 : \tau_2$. From the induction hypothesis it follows that either e_1 is a value or e_1 can make progress. In the latter case we can make progress by applying [E-App]. If e_1 is a value then it follows from the canonical forms lemma that either $e_1 = \textbf{close}\ \lambda x.e\ \textbf{in}\ \rho$ or $e_1 = \lightning^{\ell}$. In the first case we can make progress by applying [E-AppAbs]. In the second case it follows from the induction hypothesis that either e_2 can make progress or is a value. In these cases we can make progress by applying [E-AppExn1] or [E-AppExn2], respectively.

Case [T-Fix]: We can make progress by applying [E-Fix].

Case [T-Let]: We can make progress by applying [E-Let].

Case [T-If]: Given $C; \Gamma \vdash \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 : \tau$ then it follows from the induction hypothesis that either e_1 is a value or $\rho \vdash e_1 \longrightarrow e_1'$. In the latter case we can make progress by applying [E-If]. In the former case we continue by case-inspection. We assumed that e is well-typed in the underlying type system, thus from the canonical forms lemma if follows that either $e_1 = \textbf{true}$, $e_1 = \textbf{false}$ or $e_1 = \lightning^{\ell}$. In the first two cases we can make progress by applying [E-IfTrue] or [E-IfFalse], respectively. In the third case it follows from the induction hypothesis that either $\rho \vdash e_2 \longrightarrow e_2'$ or $\rho \vdash e_3 \longrightarrow e_3'$, or both e_2 and e_3 are values. In those cases [E-IfExn1], [E-IfExn2] or [E-IfExn3] apply, respectively.

Case [T-Op]: Given $C; \Gamma \vdash e_1 \oplus e_2 : \alpha$, it follows from the induction hypothesis that either at least one of e_1 and e_2 can make progress or both are values. In the first case either [E-Op1] or [E-Op2] applies. We assumed the expression was well-typed in the underlying type system, thus in the second case if follows

from the canonical forms lemma that either $e_i = n_i$ or $e_i = \lightning^{\ell_i}$ for $i \in \{1, 2\}$. In these cases either [E-OpNum], [E-OpExn1], [E-OpExn2] or [E-OpExn3] applies.

Case [T-Pair]: We can make progress by applying [E-Pair].

Case [T-Fst]: Given $C; \Gamma \vdash \mathbf{fst}\ e : \tau_1$, it follows from the induction hypothesis that either e can make progress or it is a value. In the first case the whole expression can make progress by applying [E-Fst]. In the second case it follows from the canonical forms lemma that either $e = \mathbf{close}\ (e_1, e_2)\ \mathbf{in}\ \rho$ or $e = \lightning^{\ell}$. In both cases we can make progress by applying [E-FstPair] or [E-FstExn], respectively.

Case [T-Snd]: As [T-Fst].

Case [T-Nil]: [] is a value.

Case [T-Cons]: We can make progress by applying [E-Cons].

Case [T-Case]: Given

$$C; \Gamma \vdash \mathbf{case}\ e_1\ \mathbf{of}\ \{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} : \tau,$$

it follows from the induction hypothesis that either e_1 is a value or it can make progress. In the latter case we can make progress by applying [E-Case]. In the former case we continue by case inspection on the values of e_1. We assumed that e is well-typed in the underlying type system, thus from the canonical forms lemma it follows that either $e_1 = []$, $e_1 = \mathbf{close}\ e_1 :: e_2\ \mathbf{in}\ \rho$ or $e_1 = \lightning^{\ell}$. In the first and second case we can make progress by applying [E-CaseNil] or [E-CaseCons], respectively. In case e_1 is an exceptional value, the induction hypothesis tells us that either at least one of e_2 and e_3 can make progress, or both of them are values. In these case we can apply [E-CaseExn1] or [E-CaseExn2], or [E-CaseExn3], respectively.

Case [T-Close]: $\mathbf{close}\ e\ \mathbf{in}\ \rho$ is a value.

Case [T-Bind]: Given $C; \Gamma \vdash \mathbf{bind}\ \rho\ \mathbf{in}\ e : \tau$ it follows from the induction hypothesis that either e can make progress or it is a value. In these cases we can apply [E-Bind1] or [E-Bind2], respectively.

A.4 Preservation

Theorem 7 (Preservation). *If* $C; \Gamma \vdash e : \sigma_1$, $\rho \vdash e \longrightarrow e'$ *and* $C \vdash \Gamma \bowtie \rho$ *then* $C; \Gamma \vdash e' : \sigma_2$ *with* $C \Vdash \sigma_2 \leqslant_{\delta\chi} \sigma_1$.

Proof. By induction on the reduction relation.

Case [E-Var]: We need to show that if

$$D; \Delta \vdash x : \tau$$
$$\rho, x : e \vdash x \longrightarrow \mathbf{bind}\ \rho\ \mathbf{in}\ e$$

and

$$C \vdash \Delta \bowtie \rho, x : e$$

then

$$C; \Delta \vdash \textbf{bind } \rho \textbf{ in } e : \tau'$$

with

$$C \Vdash \tau' \leqslant_{\delta\chi} \tau.$$

Applying the inversion lemma to $D; \Delta \vdash x : \tau$ reveals it can only have been constructed using [T-Var], giving us:

$$C; \Gamma, x : \forall \overline{\alpha}.\ \tau \textbf{ with } D \vdash x : \tau\left[\overline{\beta}/\overline{\alpha}\right] \tag{47}$$

$$C \Vdash D\left[\overline{\beta}/\overline{\alpha}\right] \tag{48}$$

Thus we can take $\Delta \mapsto \Gamma, x : \forall \overline{\alpha}.\ \tau \textbf{ with } D$ and $\tau \mapsto \tau\left[\overline{\beta}/\overline{\alpha}\right]$ and show that if

$$C; \Gamma, x : \forall \overline{\alpha}.\ \tau \textbf{ with } D \vdash x : \tau\left[\overline{\beta}/\overline{\alpha}\right]$$
$$\rho, x : e \ \vdash \ x \longrightarrow \textbf{ bind } \rho \textbf{ in } e$$

and

$$C \vdash \Gamma, x : \forall \overline{\alpha}.\ \tau \textbf{ with } D \bowtie \rho, x : e$$

then

$$C; \Gamma, x : \forall \overline{\alpha}.\ \tau \textbf{ with } D \vdash \textbf{bind } \rho \textbf{ in } e : \tau\left[\overline{\beta}/\overline{\alpha}\right]$$

with

$$C \Vdash \tau\left[\overline{\beta}/\overline{\alpha}\right] \leqslant_{\delta\chi} \tau\left[\overline{\beta}/\overline{\alpha}\right].$$

Applying the inversion lemma to

$$C \vdash \Gamma, x : \forall \overline{\alpha}.\ \tau \textbf{ with } D \bowtie \rho, x : e$$

reveals it can only have been constructed by [EC-Extend], giving us:

$$C \vdash \Gamma \bowtie \rho \tag{49}$$

$$C; \Gamma \vdash e : \forall \overline{\alpha}.\ \tau \textbf{ with } D \tag{50}$$

Applying [T-Bind] to (49) and (50) gives us:

$$C; \Gamma \vdash \textbf{bind } \rho \textbf{ in } e : \forall \overline{\alpha}.\ \tau \textbf{ with } D \tag{51}$$

Applying [T-Inst] to (51) and (48) gives us:

$$C; \Gamma \vdash \textbf{bind } \rho \textbf{ in } e : \tau\left[\overline{\beta}/\overline{\alpha}\right] \tag{52}$$

We conclude the proof by noting that $C \Vdash \tau\left[\overline{\beta}/\overline{\alpha}\right] \leqslant_{\delta\chi} \tau\left[\overline{\beta}/\overline{\alpha}\right]$ follows from the reflexivity of the subtype relation.

Case [E-Abs]: Apply [T-Close] (and note that the subtype relation is reflexive.)

Case [E-App]: We need to show that if $C; \Gamma \vdash e_1\ e_2 : \tau_{12}$ and $\rho \vdash e_1\ e_2 \longrightarrow e_1'\ e_2$ then $C; \Gamma \vdash e_1'\ e_2 : \tau_2$ with $C \Vdash \tau_2 \leqslant_{\delta\chi} \tau_{12}$. The induction hypothesis states that if

$$C; \Gamma \vdash e_1 : \tau_{11} \xrightarrow{\alpha_1} \tau_{12}$$

and

$$\rho \vdash e_1 \longrightarrow e_1'$$

then

$$C; \Gamma \vdash e_1' : \tau_{21} \xrightarrow{\alpha_2} \tau_{22} \tag{53}$$

with $C \Vdash \tau_{21} \xrightarrow{\alpha_2} \tau_{22} \leqslant_{\delta\chi} \tau_{11} \xrightarrow{\alpha_1} \tau_{12}$, which—by inversion of the subtyping relation—implies

$$C \Vdash \tau_{11} \leqslant_{\delta\chi} \tau_{21} \tag{54}$$
$$C \Vdash \tau_{22} \leqslant_{\delta\chi} \tau_{12} \tag{55}$$
$$C \Vdash \alpha_2 \sqsubseteq_{\delta\chi} \alpha_1. \tag{56}$$

Applying the inversion lemma to $C; \Gamma \vdash e_1\ e_2 : \tau_{12}$ reveals it can only have been constructed by [T-App], thus we have

$$C; \Gamma \vdash e_1 : \tau_{11} \xrightarrow{\alpha_1} \tau_{12} \tag{57}$$
$$C; \Gamma \vdash e_2 : \tau_{13} \tag{58}$$
$$C \Vdash \tau_{13} \leqslant_{\delta\chi} \tau_{11} \tag{59}$$
$$C \Vdash \alpha_1 \sqsubseteq_{\chi} \lceil \tau_{12} \rceil \tag{60}$$
$$C \Vdash \exists_\chi \alpha_1 \Rightarrow \lceil \tau_{13} \rceil \sqsubseteq_{\chi} \lceil \tau_{12} \rceil. \tag{61}$$

We can now apply the induction hypothesis. By applying [T-App] with $e_1 \mapsto e_1'$, $e_2 \mapsto e_2$, $\tau_1 \mapsto \tau_{21}$, $\tau_2 \mapsto \tau_{12}$, $\tau_3 \mapsto \tau_{13}$ and $\alpha \mapsto \alpha_1$ we find that

$$C; \Gamma \vdash e_1'\ e_2 : \tau_{12}.$$

We still need to check that the preconditions of [T-App] hold:

1. $C; \Gamma \vdash e_1' : \tau_{21} \xrightarrow{\alpha_1} \tau_{12}$ follows from applying [T-Sub] to (53) using (55) and (56).
2. $C \Vdash \tau_{13} \leqslant_{\delta\chi} \tau_{21}$ follows from the transitivity of the subtyping relation applied to (54) and (59).

The other three preconditions follow directly from (58), (60), and (61) respectively.

To conclude the proof, we need to show that $C \Vdash \tau_{12} \leqslant_{\delta\chi} \tau_{12}$, which holds by reflexivity of the subtyping relation.

Case [E-AppAbs]: We need to show that if

$$C; \Gamma \vdash (\textbf{close}\ \lambda x.e_1\ \textbf{in}\ \rho_1)\ \ e_2 : \tau_2, \tag{62}$$
$$C \vdash \Gamma \bowtie \rho \tag{63}$$

and

$$\rho\ \vdash\ \ (\textbf{close}\ \lambda x.e_1\ \textbf{in}\ \rho_1)\ \ e_2 \longrightarrow\ \textbf{bind}\ \ (\rho_1, x : \textbf{bind}\ \rho\ \textbf{in}\ e_2)\ \ \textbf{in}\ e_1 \tag{64}$$

then

$$C; \Gamma \vdash \textbf{bind}\ \ (\rho_1, x : \textbf{bind}\ \rho\ \textbf{in}\ e_2)\ \ \textbf{in}\ e_1 : \tau_2' \tag{65}$$

with

$$C \Vdash \tau_2' \leqslant_{\delta\chi} \tau_2. \tag{66}$$

The inversion lemma reveals that the last inference rule used to construct (62) can only have been [T-App], giving us:

$$C; \Gamma \vdash \textbf{close}\ \lambda x.e_1\ \textbf{in}\ \rho_1 : \tau_1 \xrightarrow{\alpha} \tau_2 \tag{67}$$
$$C; \Gamma \vdash e_2 : \tau_3 \tag{68}$$
$$C \Vdash \tau_3 \leqslant_{\delta\chi} \tau_1 \tag{69}$$
$$C \Vdash \alpha \sqsubseteq_\chi \lceil \tau_2 \rceil \tag{70}$$
$$C \Vdash \exists_\chi \alpha \Rightarrow \lceil \tau_3 \rceil \sqsubseteq_\chi \lceil \tau_2 \rceil \tag{71}$$

Applying the inversion lemma to (67) reveals that the last inference rule used in the derivation can only have been [T-Close], giving us:

$$C; \Delta \vdash \lambda x.e_1 : \tau_1 \xrightarrow{\alpha} \tau_2 \tag{72}$$
$$C; \Gamma \vdash \Delta : \rho_1 \tag{73}$$

Applying the inversion lemma to (72) reveals that the last inference rule used in the derivation can only have been [T-Abs], giving us:

$$C; \Delta, x : \tau \vdash e_1 : \tau_2 \tag{74}$$

We construct the desired result using [T-Bind] with $e \mapsto e_1$, $\tau \mapsto \tau_2$, $\rho \mapsto (\rho_1, x : \textbf{bind}\ \rho\ \textbf{in}\ e_2)$ and $\Delta \mapsto \Delta, x : \tau_1$. We still need to check the preconditions on [T-Bind] hold:

1. The premise $C; \Delta, x : \tau \vdash e_1 : \tau_2$ is satisfied by (74).
2. The premise $C \vdash \Delta, x : \tau_1 \bowtie \rho_1, x : \textbf{bind}\ \rho\ \textbf{in}\ e_2$ can be constructed using [EC-Extend], checking its premises in turn:
   (a) $C \vdash \Delta \bowtie \rho_1$ is satisfied by (73).

(b) $C; \Delta \vdash \mathbf{bind}\ \rho\ \mathbf{in}\ e_2 : \tau_1$ can be constructed by [T-Bind] with $\Gamma \mapsto \Delta$, $\Delta \mapsto \Gamma$, $\rho \mapsto \rho$, $e \mapsto e_2$ and $\tau \mapsto \tau_1$; checking its premises in turn:
   i. $C \vdash \Gamma \bowtie \rho$ is satisfied by (63).
   ii. $C; \Gamma \vdash e_2 : \tau_1$ holds by using [T-Sub] on (68) and (69).

We conclude the proof by noting that $C \Vdash \tau_2 \leqslant_{\delta\chi} \tau_2$ follows from the reflexivity of the subtype relation.

Case [E-AppExn1]: We need to show that if $C; \Gamma \vdash \lightning^{\ell_1} : \tau_1$ and $\rho \vdash \lightning^{\ell_1}\ e_2 \longrightarrow \lightning^{\ell_1}\ e_2'$ then $C; \Gamma \vdash e_2' : \tau_2$ with $C \Vdash \tau_2 \leqslant_{\delta\chi} \tau_1$. The induction hypothesis states that if

$$C; \Gamma \vdash e_2 : \tau_3$$

and

$$\rho \vdash e_2 \longrightarrow e_2'$$

then

$$C; \Gamma \vdash e_2' : \tau_4 \tag{75}$$
$$C \Vdash \tau_4 \leqslant_{\delta\chi} \tau_3. \tag{76}$$

Applying the inversion lemma to $C; \Gamma \vdash \lightning^{\ell_1}\ e_2 : \tau_1$ reveals it can only have been constructed by [T-App], thus we have:

$$C; \Gamma \vdash \lightning^{\ell_1} : \tau_1 \xrightarrow{\alpha} \tau_2 \tag{77}$$
$$C; \Gamma \vdash e_2 : \tau_3 \tag{78}$$
$$C \Vdash \tau_3 \leqslant_{\delta\chi} \tau_1 \tag{79}$$
$$C \Vdash \alpha \sqsubseteq_{\chi} \lceil \tau_2 \rceil \tag{80}$$
$$C \Vdash \exists_{\chi}\alpha \Rightarrow \lceil \tau_3 \rceil \sqsubseteq_{\chi} \lceil \tau_2 \rceil \tag{81}$$

We can now apply the induction hypothesis. By applying [T-App] with $e_1 \mapsto \lightning^{\ell_1}$, $e_2 \mapsto e_2'$, $\tau_1 \mapsto \tau_1$, $\tau_2 \mapsto \tau_2$, $\tau_3 \mapsto \tau_4$ and $\alpha \mapsto \alpha$ we find that

$$C; \Gamma \vdash \lightning^{\ell_1}\ e_2' : \tau_2.$$

We still need to check that the preconditions on [T-App] hold:

1. $C \Vdash \tau_4 \leqslant_{\delta\chi} \tau_1$ follows from (76), (79) and the transitivity of the subtype relation.
2. Using implication elimination on (81) gives us:

   $$C, \exists_{\chi}\alpha \Vdash \lceil \tau_3 \rceil \sqsubseteq_{\chi} \lceil \tau_2 \rceil \tag{82}$$

   Using Lemma 1 on (76) gives us:

   $$C \Vdash \lceil \tau_4 \rceil \sqsubseteq_{\delta\chi} \lceil \tau_3 \rceil \tag{83}$$

   Weakening (83) gives us:

   $$C, \exists_{\chi}\alpha \Vdash \lceil \tau_4 \rceil \sqsubseteq_{\delta\chi} \lceil \tau_3 \rceil \tag{84}$$

From (84), (82) and the transitivity of the subtype relation we get:

$$C, \exists_\chi \alpha \Vdash \lceil \tau_4 \rceil \sqsubseteq_\chi \lceil \tau_2 \rceil \tag{85}$$

Using implication introduction on (85) gives us the desired precondition:

$$C \Vdash \exists_\chi \alpha \Rightarrow \lceil \tau_4 \rceil \sqsubseteq_\chi \lceil \tau_2 \rceil \tag{86}$$

The other three preconditions are fulfilled by assumptions (77), (75) and (80). respectively.

To conclude the proof, we need to show that $C \Vdash \tau_2 \leqslant_{\delta\chi} \tau_2$, which holds by the reflexivity of the subtype relation.

Case [E-AppExn2]: We need to show that if $C; \Gamma \vdash \lightning^{\ell_1}\ v_2^{\ell_2} : \tau_{12}$ and $\rho \vdash \lightning^{\ell_1}\ v_2^{\ell_2} \longrightarrow \lightning^{\ell_1 \sqcup \ell_2}$ then $C; \Gamma \vdash \lightning^{\ell_1 \sqcup \ell_2} : \tau'_{12}$ with $C \Vdash \tau'_{12} \leqslant_{\delta\chi} \tau_{12}$.

Applying the inversion lemma to $C; \Gamma \vdash \lightning^{\ell_1}\ v_2^{\ell_2} : \tau_{12}$ reveals it can only have been constructed by [T-App], thus we have:

$$C; \Gamma \vdash \lightning^{\ell_1} : \tau_{11} \xrightarrow{\alpha_1} \tau_{12} \tag{87}$$

$$C; \Gamma \vdash v_2^{\ell_2} : \tau_{13} \tag{88}$$

$$C \Vdash \tau_{13} \leqslant_{\delta\chi} \tau_{11} \tag{89}$$

$$C \Vdash \alpha_1 \sqsubseteq_\chi \lceil \tau_{12} \rceil \tag{90}$$

$$C \Vdash \exists_\chi \alpha_1 \Rightarrow \lceil \tau_{13} \rceil \leqslant_\chi \lceil \tau_{12} \rceil \tag{91}$$

Applying the inversion lemma to (87) reveals it can only have been constructed by [T-Exn], giving us:

$$C \Vdash \ell_1 \sqsubseteq_\chi \lceil \tau_{11} \xrightarrow{\alpha_1} \tau_{12} \rceil \tag{92}$$

or, by simplification of $\lceil \cdot \rceil$:

$$C \Vdash \ell_1 \sqsubseteq_\chi \alpha_1 \tag{93}$$

When inverting (88) we have to distinguish between two cases: either $v_2^{\ell_2}$ is an exceptional value and it can—by the inversion lemma—only have been been constructed by [T-Exn], or $v_2^{\ell_2}$ is a non-exceptional value and we took $\ell_2 = \emptyset$.

Subcase "$v_2^{\ell_2}$ is an exceptional value": Applying the inversion lemma to (88) reveals it can only have been constructed by [T-Exn], giving us:

$$C \Vdash \ell_2 \sqsubseteq_\chi \lceil \tau_{13} \rceil \tag{94}$$

We use [T-Exn] to construct an exceptional value

$$C; \Gamma \vdash \lightning^{\ell_1 \sqcup \ell_2} : \tau_{12}.$$

The precondition $C \Vdash \ell_1 \sqcup \ell_2 \sqsubseteq_\chi \lceil \tau_{12} \rceil$ can be decomposed into:

$$C \Vdash \ell_1 \sqsubseteq_\chi \lceil \tau_{12} \rceil \tag{95}$$

$$C \Vdash \ell_2 \sqsubseteq_\chi \lceil \tau_{12} \rceil \tag{96}$$

Precondition (95) is satisfied by transitively combining (93) with (90). Using implication elimination on (91) using (93), we find:

$$C \Vdash \lceil \tau_{13} \rceil \sqsubseteq_\chi \lceil \tau_{12} \rceil \tag{97}$$

Precondition (96) is satisfied by transitively combining (94) with (97). For this subcase we can conclude the whole proof by noting that $C \Vdash \tau_{12} \leqslant_{\delta\chi} \tau_{12}$ follows from the reflexivity of the subtype relation.

Subcase "$v_2^{\ell_2}$ is a non-exceptional value": If $v_2^{\ell_2}$ is a non-exceptional value then we took $\ell_2 = \emptyset$. We use [T-Exn] to construct an exceptional value $C; \Gamma \vdash \lightning^{\ell_1 \sqcup \ell_2} : \tau_{12}$. The precondition $C \Vdash \ell_1 \sqcup \emptyset \sqsubseteq_\chi \lceil \tau_{12} \rceil$ can be simplified to:

$$C \Vdash \ell_1 \sqsubseteq_\chi \lceil \tau_{12} \rceil \tag{98}$$

Precondition (98) follows from transitively combining (93) with (90). For this subcase we can conclude the whole proof by noting that $C \Vdash \tau_{12} \leqslant_{\delta\chi} \tau_{12}$ follows from the reflexivity of the subtype relation.

Case [E-Fix]: We need to show that if

$$C; \Gamma \vdash \mathbf{fix}\ f.\ e : \tau_1 \left[\overline{\beta}/\overline{\alpha}\right]$$
$$\rho \ \vdash\ \mathbf{fix}\ f.\ e \longrightarrow\ \mathbf{bind}\ (\rho, f : \mathbf{bind}\ \rho\ \mathbf{in}\ \mathbf{fix}\ f.\ e)\ \mathbf{in}\ e$$

and

$$C \vdash \rho \bowtie \Gamma$$

then

$$C; \Gamma \vdash \mathbf{bind}\ (\rho, f : \mathbf{bind}\ \rho\ \mathbf{in}\ \mathbf{fix}\ f.\ e)\ \mathbf{in}\ e : \tau_1 \left[\overline{\beta}/\overline{\alpha}\right]$$

with

$$C \Vdash \tau_1 \left[\overline{\beta}/\overline{\alpha}\right] \leqslant_{\delta\chi} \tau_1 \left[\overline{\beta}/\overline{\alpha}\right].$$

Applying the inversion lemma to $C; \Gamma \vdash \mathbf{fix}\ f.\ e : \tau_1 \left[\overline{\beta}/\overline{\alpha}\right]$ reveals it can only have been constructed using [T-Fix], giving us:

$$C \Vdash D \left[\overline{\beta}/\overline{\alpha}\right] \tag{99}$$
$$D; \Gamma, f : \forall \overline{\alpha}.\ \tau_1\ \mathbf{with}\ D \vdash e : \tau_2 \tag{100}$$
$$D \Vdash \tau_2 \leqslant_{\delta\chi} \tau_1 \tag{101}$$
$$\overline{\alpha} \cap fv(\Gamma; C) = \emptyset \tag{102}$$

We can now construct the desired result using [T-Bind] with $\rho \mapsto (\rho, f : \mathbf{bind}\ \rho\ \mathbf{in}\ \mathbf{fix}\ f.\ e)$, $\sigma \mapsto \tau_1 \left[\overline{\beta}/\overline{\alpha}\right]$, $e \mapsto e$ and $\Delta \mapsto \Gamma, f : \forall \overline{\alpha}.\ \tau_1\ \mathbf{with}\ D$.

We still need to check the preconditions on [T-Bind] hold:

1. Applying the substitution $\left[\overline{\beta}/\overline{\alpha}\right]$ to (100) gives us

$$D \left[\overline{\beta}/\overline{\alpha}\right]; \Gamma, f : \forall \overline{\alpha}.\ \tau_1\ \mathbf{with}\ D \left[\overline{\beta}/\overline{\alpha}\right] \vdash e : \tau_2 \left[\overline{\beta}/\overline{\alpha}\right] \tag{103}$$

However, as $fv(\Gamma)$ and the domain of the substitution are disjoint and f quantifies over the entire domain of the substitution, this is equivalent to

$$D\left[\overline{\beta}/\overline{\alpha}\right];\Gamma, f : \forall\overline{\alpha}.\ \tau_1\ \textbf{with}\ D \vdash e : \tau_2\left[\overline{\beta}/\overline{\alpha}\right] \tag{104}$$

From (101) it follows that

$$D\left[\overline{\beta}/\overline{\alpha}\right] \Vdash \tau_2\left[\overline{\beta}/\overline{\alpha}\right] \leqslant_{\delta\chi} \tau_1\left[\overline{\beta}/\overline{\alpha}\right] \tag{105}$$

Using [T-Sub] on (104) with (105) gives us

$$D\left[\overline{\beta}/\overline{\alpha}\right];\Gamma, f : \forall\overline{\alpha}.\ \tau_1\ \textbf{with}\ D \vdash e : \tau_2\left[\overline{\beta}/\overline{\alpha}\right] \tag{106}$$

From (107) and (99) the desired result follows:

$$C;\Gamma, f : \forall\overline{\alpha}.\ \tau_1\ \textbf{with}\ D \vdash e : \tau_2\left[\overline{\beta}/\overline{\alpha}\right] \tag{107}$$

2. We can construct

$$C \vdash \Gamma, f : \forall\overline{\alpha}.\ \tau_1\ \textbf{with}\ D \bowtie \rho, f : \textbf{bind}\ \rho\ \textbf{in}\ \textbf{fix}\ f.\ e$$

using [EC-Extend]. We check its preconditions:

(a) The precondition $C \vdash \Gamma \bowtie \rho$ is an assumption made by the theorem.

(b) The precondition $C;\Gamma \vdash \textbf{bind}\ \rho\ \textbf{in}\ \textbf{fix}\ f.\ e : \forall\overline{\alpha}.\ \tau_1\ \textbf{with}\ D$ can be constructed using [T-Bind] with $\Delta \mapsto \Gamma$. We check its preconditions:

   i. The precondition $C \vdash \Gamma \bowtie \rho$ is an assumption made by the theorem.

   ii. The precondition $C;\Gamma \vdash \textbf{fix}\ f.\ e : \forall\overline{\alpha}.\ \tau_1\ \textbf{with}\ D$ can be constructed by [T-Gen]. We check its preconditions:

      A. The precondition $\overline{\alpha} \cap fv(\Gamma; C)$ is satisfied by (102).

      B. The precondition $C, D;\Gamma \vdash \textbf{fix}\ f.\ e : \tau_1$ can be constructed using [T-Fix] with $C \mapsto C, D$ and $\overline{\beta} \mapsto \overline{\alpha}$.

Case [E-Let]: We need to show that if

$$C;\Gamma \vdash \textbf{let}\ x = e_1\ \textbf{in}\ e_2 : \tau_2 \tag{108}$$

$$C \vdash \Gamma \bowtie \rho \tag{109}$$

and

$$\rho\ \vdash\ \textbf{let}\ x = e_1\ \textbf{in}\ e_2 \longrightarrow\ \textbf{bind}\ \ (\rho, x : \textbf{bind}\ \rho\ \textbf{in}\ e_1)\ \ \textbf{in}\ e_2 \tag{110}$$

then

$$C;\Gamma \vdash \textbf{bind}\ \ (\rho, x : \textbf{bind}\ \rho\ \textbf{in}\ e_1)\ \ \textbf{in}\ e_2 : \tau_2' \tag{111}$$

with

$$C \Vdash \tau_2' \leqslant_{\delta\chi} \tau. \tag{112}$$

Applying the inversion lemma to (108) reveals that the last inference rule used in its derivation can only have been [T-Let], giving us:

$$C, D; \Gamma \vdash e_1 : \tau_1 \tag{113}$$

$$C; \Gamma, x : \forall \overline{\alpha}.\ \tau_1 \textbf{ with } D \vdash e_2 : \tau_2 \tag{114}$$

$$\overline{\alpha} \cap fv(\Gamma; C) = \emptyset \tag{115}$$

We construct the desired result using [T-Bind] with $\tau \mapsto \tau_2$; $e \mapsto e_2$; $\rho \mapsto (\rho, x : \textbf{bind } \rho \textbf{ in } e_1)$ and $\Delta \mapsto \Gamma, x : \forall \overline{\alpha}.\ \tau_1 \textbf{ with } D$. We still need to check the preconditions on [T-Bind]:

1. The premise $C; \Gamma, \forall \overline{\alpha}.\ \tau_1 \textbf{ with } D \vdash e_2 : \tau_2$ is satisfied by (114).
2. The premise $C \vdash \Gamma, x : \forall \overline{\alpha}.\ \tau_1 \textbf{ with } D \bowtie \rho, x : \textbf{bind } \rho \textbf{ in } e_1$ can be constructed by [EC-Extend]. We check its preconditions:
   (a) The premise $C \vdash \Gamma \bowtie \rho$ follows from the assumption made by the theorem.
   (b) The premise $C; \Gamma \vdash \textbf{bind } \rho \textbf{ in } e_1 : \forall \overline{\alpha}.\ \tau_1 \textbf{ with } D$ can be constructed by [T-Gen]. We check its preconditions:
      i. The premise $\overline{\alpha} \cap fv(\Gamma; C)$ is satisfied by (115).
      ii. The premise $C, D; \Gamma \vdash \textbf{bind } \rho \textbf{ in } e_1 : \tau_1$ can be constructed by [T-Bind] with $\rho \mapsto \rho$, $e \mapsto e_1$, $\Delta \mapsto \Gamma$, $\tau \mapsto \tau_1$ and $C \mapsto C, D$. We check its preconditions:
         A. The premise $C, D; \Gamma \vdash e_1 : \tau_1$ is satisfied by (113).
         B. The premise $C, D \vdash \Gamma \bowtie \rho$ is satisfied by weakening the assumption made by the theorem

We conclude the proof by noting that $C \Vdash \tau_2 \leqslant_{\delta\chi} \tau_2$ follows from the reflexivity of the subtype relation.

Case [E-If]: We need to show that if

$$C; \Gamma \vdash \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 : \tau \tag{116}$$

and

$$\rho \ \vdash\ \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 \longrightarrow\ \textbf{if } e_1' \textbf{ then } e_2 \textbf{ else } e_3 \tag{117}$$

then $C; \Gamma \vdash \textbf{if } e_1' \textbf{ then } e_2 \textbf{ else } e_3 : \tau'$ with $C \Vdash \tau' \leqslant_{\delta\chi} \tau$.

The induction hypothesis states that if $C; \Gamma \vdash e_1 : \alpha_1$ and $\rho \ \vdash\ e_1 \longrightarrow\ e_1'$ then $C; \Gamma \vdash e_1' : \alpha_1'$ with $C \Vdash \alpha_1' \leqslant_{\delta\chi} \alpha_1$.

Applying the inversion lemma to (116) reveals it can only have been constructed by [T-If], giving us:

$$C; \Gamma \vdash e_1 : \alpha_1 \tag{118}$$

$$C; \Gamma \vdash e_2 : \tau_2 \tag{119}$$

$$C; \Gamma \vdash e_3 : \tau_3 \tag{120}$$

$$C \Vdash \mathbf{T} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_2 \leqslant_{\delta\chi} \tau \tag{121}$$

$$C \Vdash \mathbf{F} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_3 \leqslant_{\delta\chi} \tau \tag{122}$$

$$C \Vdash \alpha_1 \sqsubseteq_\chi \lceil \tau \rceil \tag{123}$$

We can now apply the induction hypothesis, giving us:

$$C; \Gamma \vdash e_1' : \alpha_1' \tag{124}$$
$$C \Vdash \alpha_1' \leqslant_{\delta\chi} \alpha_1 \tag{125}$$

Using the inversion lemma on (125) reveals it can only have been constructed by

$$C \Vdash \alpha_1' \sqsubseteq_{\delta\chi} \alpha_1 \tag{126}$$

We can construct the desired result using [T-If] with $e_1 \mapsto e_1$. We still need to check that the preconditions on [T-If] hold:

1. Applying [T-Sub] with (124) and (126) gives us $C; \Gamma \vdash e_1' : \alpha_1$.

The other five preconditions follow immediately from (119)–(123).

Case [E-IfTrue]: We need to show that if

$$C; \Gamma \vdash \textbf{if true then } e_2 \textbf{ else } e_3 : \tau$$

and

$$\rho \ \vdash \ \textbf{if true then } e_2 \textbf{ else } e_3 \longrightarrow \ e_2$$

then

$$C; \Gamma \vdash e_2 : \tau'$$

with

$$C \Vdash \tau' \leqslant_{\delta\chi} \tau.$$

Applying the inversion lemma to $C; \Gamma \vdash \textbf{if true then } e_2 \textbf{ else } e_3 : \tau$ reveals it can only have been constructed by [T-If], thus we have:

$$C; \Gamma \vdash \textbf{true} : \alpha_1 \tag{127}$$
$$C; \Gamma \vdash e_2 : \tau_2 \tag{128}$$
$$C; \Gamma \vdash e_3 : \tau_3 \tag{129}$$
$$C \Vdash \mathbf{T} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_2 \leqslant_{\delta\chi} \tau \tag{130}$$
$$C \Vdash \mathbf{F} \sqsubseteq_\chi \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_3 \leqslant_{\delta\chi} \tau \tag{131}$$
$$C \Vdash \alpha_1 \sqsubseteq_\chi \lceil \tau \rceil \tag{132}$$

Applying the inversion lemma to (127) reveals it can only have been constructed using [T-Con], giving us:

$$C \Vdash \iota(\textbf{true}) \sqsubseteq_\delta \alpha_1, \tag{133}$$

which is equivalent to

$$C \Vdash \mathbf{T} \sqsubseteq_\delta \alpha_1. \tag{134}$$

Using $\vee$-introduction on (134) and implication elimination on (130) we find

$$C \Vdash \mathbf{T} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \tag{135}$$
$$C, \mathbf{T} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Vdash \tau_2 \leqslant_{\delta\chi} \tau \tag{136}$$

Applying the *modus ponens* rules on (135) and (136) gives us

$$C \Vdash \tau_2 \leqslant_{\delta\chi} \tau \tag{137}$$

This concludes the proof, as (128) and (137) are what was asked for.

Case [E-IfFalse]: As [E-IfTrue].

Case [E-IfExn1]: We need to show that if

$$\Gamma; \Gamma \vdash \textbf{if } \lightning^{\ell_1} \textbf{ then } e_2 \textbf{ else } e_3 : \tau$$

and

$$\rho \ \vdash \ \textbf{if } \lightning^{\ell_1} \textbf{ then } e_2 \textbf{ else } e_3 \longrightarrow \ \textbf{if } \lightning^{\ell_1} \textbf{ then } e_2' \textbf{ else } e_3$$

then

$$\Gamma; \Gamma \vdash \textbf{if } \lightning^{\ell_1} \textbf{ then } e_2' \textbf{ else } e_3 : \tau'$$

with

$$C \Vdash \tau' \leqslant_{\delta\chi} \tau.$$

The induction hypothesis states that if $\Gamma; \Gamma \vdash e_2 : \tau_2$ and $\rho \ \vdash \ e_2 \longrightarrow \ e_2'$ then $\Gamma; \Gamma \vdash e_2' : \tau_2'$ with $C \Vdash \tau_2' \leqslant_{\delta\chi} \tau_2$.

Applying the inversion lemma to $\Gamma; \Gamma \vdash \textbf{if } \lightning^{\ell_1} \textbf{ then } e_2 \textbf{ else } e_3 : \tau$ reveals it can only have been constructed by [T-If], giving us:

$$\Gamma; \Gamma \vdash \lightning^{\ell_1} : \alpha_1 \tag{138}$$
$$\Gamma; \Gamma \vdash e_2 : \tau_2 \tag{139}$$
$$\Gamma; \Gamma \vdash e_3 : \tau_3 \tag{140}$$
$$C \Vdash \mathbf{T} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_2 \leqslant_{\delta\chi} \tau \tag{141}$$
$$C \Vdash \mathbf{F} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_3 \leqslant_{\delta\chi} \tau \tag{142}$$
$$C \Vdash \alpha_1 \sqsubseteq_\chi \lceil \tau \rceil \tag{143}$$

We can now apply the induction hypothesis, giving us:

$$\Gamma; \Gamma \vdash e_2' : \tau_2' \tag{144}$$
$$C \Vdash \tau_2' \leqslant_{\delta\chi} \tau_2 \tag{145}$$

By applying [T-If] with $\alpha_1 \mapsto \alpha_1$, $\tau_2 \mapsto \tau_2'$, $\tau_3 \mapsto \tau_3$, $e_1 \mapsto \lightning^{\ell_1}$, $e_2 \mapsto e_2'$ and $e_3 \mapsto e_3$ we find that

$$\Gamma; \Gamma \vdash \textbf{if } \lightning^{\ell_1} \textbf{ then } e_2' \textbf{ else } e_3 : \tau \tag{146}$$

We still need to check that the preconditions on [T-If] hold:

1. Using weakening on (145) we get

$$C, \mathbf{T} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Vdash \tau_2' \leqslant_{\delta\chi} \tau_2 \tag{147}$$

Using implication elimination on (141) we get

$$C, \mathbf{T} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Vdash \tau_2 \leqslant_{\delta\chi} \tau \tag{148}$$

Using the transitivity of the subtype relation on (147) and (148) we get

$$C, \mathbf{T} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Vdash \tau_2' \leqslant_{\delta\chi} \tau \tag{149}$$

Using implication introduction on (149) get

$$C \Vdash \mathbf{T} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_2' \leqslant_{\delta\chi} \tau \tag{150}$$

The other five preconditions follow directly from (138), (140), (142), (143) and (144). This—together with the reflexivity of the subtype relation giving us $C \Vdash \tau \leqslant_{\delta\chi} \tau$—concludes the proof.

Case [E-IfExn2]: As [E-IfExn1].

Case [E-IfExn3]: We need to show that if

$$C; \Gamma \vdash \mathbf{if}\ \lightning^{\ell_1}\ \mathbf{then}\ v_2^{\ell_2}\ \mathbf{else}\ v_3^{\ell_3} : \tau$$

and

$$\rho\ \vdash\ \mathbf{if}\ \lightning^{\ell_1}\ \mathbf{then}\ v_2^{\ell_2}\ \mathbf{else}\ v_3^{\ell_3} \longrightarrow\ \lightning^{\ell_1 \sqcup \ell_2 \sqcup \ell_3}$$

then

$$C; \Gamma \vdash \lightning^{\ell_1 \sqcup \ell_2 \sqcup \ell_3} : \tau$$

with

$$C \Vdash \tau \leqslant_{\delta\chi} \tau.$$

Applying the inversion lemma to $C; \Gamma \vdash \mathbf{if}\ \lightning^{\ell_1}\ \mathbf{then}\ v_2^{\ell_2}\ \mathbf{else}\ v_3^{\ell_3} : \tau$ reveals it can only have been constructed by [T-If], thus we have:

$$C; \Gamma \vdash \lightning^{\ell_1} : \alpha_1 \tag{151}$$

$$C; \Gamma \vdash v_2^{\ell_2} : \tau_2 \tag{152}$$

$$C; \Gamma \vdash v_3^{\ell_3} : \tau_3 \tag{153}$$

$$C \Vdash \mathbf{T} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_2 \leqslant_{\delta\chi} \tau \tag{154}$$

$$C \Vdash \mathbf{F} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_3 \leqslant_{\delta\chi} \tau \tag{155}$$

$$C \Vdash \alpha_1 \sqsubseteq_\chi \lceil \tau \rceil \tag{156}$$

We construct a new exceptional value $C; \Gamma \vdash \lightning^{\ell_1 \sqcup \ell_2 \sqcup \ell_3} : \tau$ using [T-Exn]. This gives us the precondition

$$\ell_1 \sqcup \ell_2 \sqcup \ell_3 \sqsubseteq_\chi \alpha_1 \tag{157}$$

that we need to check it satisfied. It can be decomposed into

$$C \Vdash \ell_1 \sqsubseteq_\chi \alpha_1 \tag{158}$$

$$C \Vdash \ell_2 \sqsubseteq_\chi \alpha_1 \tag{159}$$

$$C \Vdash \ell_3 \sqsubseteq_\chi \alpha_1 \tag{160}$$

1. (158) follows from applying the inversion lemma to (151).
2. (159) follows from case inspection on $v_2^{\ell_2}$: either $v_2^{\ell_2}$ is an exceptional value and must thus have been constructed by [T-Exn], from which the requested constraint follows immediately by inversion of (152); or $v_2^{\ell_2}$ is a non-exceptional value and we took $\ell_2 = \emptyset$, which simplifies (159) to the trivially satisfied $C \Vdash \emptyset \sqsubseteq_\chi \alpha_1$.
3. The reasoning for (160) if analogous to (159).

We conclude the proof by noting that $C \Vdash \tau \leqslant_{\delta\chi} \tau$ follows from the reflexivity of the subtype relation.

Case [E-Op1]: We need to show that if $C; \Gamma \vdash e_1 \oplus e_2 : \alpha$ and $\rho \vdash e_1 \oplus e_2 \longrightarrow e_1' \oplus e_2$ then $C; \Gamma \vdash e_1' \oplus e_2 : \alpha'$ with $C \Vdash \alpha' \sqsubseteq_{\delta\chi} \alpha$. The induction hypothesis states that if $C; \Gamma \vdash e_1 : \alpha_1$ and $\rho \vdash e_1 \longrightarrow e_1'$ then

$$C; \Gamma \vdash e_1' : \alpha_1' \tag{161}$$
$$C \Vdash \alpha_1' \sqsubseteq_{\delta\chi} \alpha_1. \tag{162}$$

Applying the inversion lemma to $C; \Gamma \vdash e_1 \oplus e_2 : \alpha_1$ can only have been constructed by [T-Op], thus we have

$$C; \Gamma \vdash e_1 : \alpha_1 \tag{163}$$
$$C; \Gamma \vdash e_2 : \alpha_2 \tag{164}$$
$$C \Vdash \alpha_1 \sqsubseteq_\chi \alpha \tag{165}$$
$$C \Vdash \alpha_2 \sqsubseteq_\chi \alpha \tag{166}$$
$$C \Vdash \omega_\oplus(\alpha_1, \alpha_2, \alpha) \tag{167}$$

Applying [T-Op] with $e_1 \mapsto e_1'$, $e_2 \mapsto e_2$, $\alpha_1 \mapsto \alpha_1'$, $\alpha_2 \mapsto \alpha_2$ and $\alpha \mapsto \alpha$ we find that

$$C; \Gamma \vdash e_1' \oplus e_2 : \alpha.$$

We still need to check that the preconditions of [T-Op] hold:

1. $C \Vdash \alpha_1' \sqsubseteq_\chi \alpha$ follows from (165), (162) and the transitivity of the subtype relation.
2. $C \Vdash \omega_\oplus(\alpha_1', \alpha_2, \alpha)$ follows from (162) and the monotonicity of the operator constraint set $\omega_\oplus$ (Definition 2).

The other three preconditions follow directly from assumptions (161), (164) and (166), respectively.

To conclude the proof, we need to show that $C \Vdash \alpha \sqsubseteq_{\delta\chi} \alpha$, which holds by the the reflexivity of the subtype relation.

Case [E-Op2]: As [E-Op1].

Case [E-OpNum]: We need to show that if $C; \Gamma \vdash n_1 \oplus n_2 : \alpha$ and $\rho \vdash n_1 \oplus n_2 \longrightarrow [\![n_1 \oplus n_2]\!]$ then $C; \Gamma \vdash [\![n_1 \oplus n_2]\!] : \alpha'$ with $C \Vdash \alpha' \leqslant_{\delta\chi} \alpha$. Applying the inversion lemma to $C; \Gamma \vdash n_1 \oplus n_2 : \alpha$ can only have been constructed by [T-Op],

thus we have

$$C; \Gamma \vdash n_1 : \alpha_1 \tag{168}$$
$$C; \Gamma \vdash n_2 : \alpha_2 \tag{169}$$
$$C \Vdash \alpha_1 \sqsubseteq_\chi \alpha \tag{170}$$
$$C \Vdash \alpha_2 \sqsubseteq_\chi \alpha \tag{171}$$
$$C \Vdash \omega_\oplus(\alpha_1, \alpha_2, \alpha) \tag{172}$$

From the consistency (Definition 1) of the operator constraint set $\omega_\oplus$ and assumption (172) we conclude that

$$C; \Gamma \vdash [\![n_1 \oplus n_2]\!] : \alpha' \tag{173}$$
$$C \Vdash \alpha' \leqslant_{\delta\chi} \alpha \tag{174}$$

This finishes the proof.

Case [E-OpExn1]: We need to show that if $C; \Gamma \vdash \lightning^{\ell_1} \oplus n_2 : \alpha$ and $\rho \vdash \lightning^{\ell_1} \oplus n_2 \longrightarrow \lightning^{\ell_1}$ then $C; \Gamma \vdash \lightning^{\ell_1} : \alpha'$ with $C \Vdash \alpha' \leqslant_{\delta\chi} \alpha$.

Applying the inversion lemma to $C; \Gamma \vdash \lightning^{\ell_1} \oplus n_2 : \alpha$ reveals it can only have been constructed by [T-Op], thus we have

$$C; \Gamma \vdash \lightning^{\ell_1} : \alpha_1 \tag{175}$$
$$C; \Gamma \vdash n_2 : \alpha_2 \tag{176}$$
$$C \Vdash \alpha_1 \sqsubseteq_\chi \alpha \tag{177}$$
$$C \Vdash \alpha_2 \sqsubseteq_\chi \alpha \tag{178}$$
$$C \Vdash \omega_\oplus(\alpha_1, \alpha_2, \alpha) \tag{179}$$

Applying the inversion lemma to (175) reveals it can only have been constructed by [T-Exn], giving us

$$C \Vdash \ell_1 \sqsubseteq_\chi \alpha_1 \tag{180}$$

Applying [T-Exn] with $\ell \mapsto \ell_1$ and $\tau \mapsto \alpha$, noting that its precondition is fulfilled by transitively combining (180) with (177), we find that

$$C; \Gamma \vdash \lightning^{\ell_1} : \alpha \tag{181}$$

To conclude the proof we note that $C \Vdash \alpha \leqslant_{\delta\chi} \alpha$ follows from the transitivity of the subtype relation.

Case [E-OpExn2]: As [E-OpExn1].

Case [E-OpExn3]: We need to show that if $C; \Gamma \vdash \lightning^{\ell_1} \oplus \lightning^{\ell_2} : \alpha$ and $\rho \vdash \lightning^{\ell_1} \oplus \lightning^{\ell_2} \longrightarrow \lightning^{\ell_1 \sqcup \ell_2}$ then $C; \Gamma \vdash \lightning^{\ell_1 \sqcup \ell_2} : \alpha'$ with $C \Vdash \alpha' \sqsubseteq_{\delta\chi} \alpha$.

Applying the inversion lemma to $C; \Gamma \vdash \lightning^{\ell_1} \oplus \lightning^{\ell_2} : \alpha$ reveals it can only have been constructed by [T-Op], thus we have

$$C; \Gamma \vdash \lightning^{\ell_1} : \alpha_1 \tag{182}$$
$$C; \Gamma \vdash \lightning^{\ell_2} : \alpha_2 \tag{183}$$
$$C \Vdash \alpha_1 \sqsubseteq_\chi \alpha \tag{184}$$
$$C \Vdash \alpha_2 \sqsubseteq_\chi \alpha \tag{185}$$
$$C \Vdash \omega_\oplus(\alpha_1, \alpha_2, \alpha) \tag{186}$$

The inversion lemma shows that the last derivation rule used to construct both (182) and (183) can only have been [T-Exn], giving us

$$C \Vdash \ell_1 \sqsubseteq_\chi \lceil \alpha_1 \rceil \tag{187}$$
$$C \Vdash \ell_2 \sqsubseteq_\chi \lceil \alpha_2 \rceil \tag{188}$$

Transitively combining (184) with (187) and (185) with (188), we find

$$C \Vdash \ell_1 \sqcup \ell_2 \sqsubseteq_\chi \alpha \tag{189}$$

Finally, we construct $C; \Gamma \vdash \lightning^{\ell_1 \sqcup \ell_2} : \alpha$ by [T-Exn] using (189) to fulfil its precondition and conclude by noting that $C \Vdash \alpha \sqsubseteq_{\delta\chi} \alpha$ follows from the reflexivity of the subtype relation.

Case [E-Pair]: Apply [T-Close] (and note that the subtype relation is reflexive.)

Case [E-Cons]: Apply [T-Close] (and note that the subtype relation is reflexive.)

Case [E-Fst]: We need to show that if

$$C; \Gamma \vdash \mathbf{fst}\ e : \tau$$
$$C \vdash \Gamma \bowtie \rho$$

and

$$\rho \vdash \mathbf{fst}\ e \longrightarrow \mathbf{fst}\ e'$$

then

$$C; \Gamma \vdash \mathbf{fst}\ e' : \tau'$$

with

$$C \Vdash \tau' \leqslant_{\delta\chi} \tau.$$

The induction hypothesis states that if

$$C; \Gamma \vdash e : \tau_1 \times^\alpha \tau_2$$
$$C \vdash \Gamma \bowtie \rho$$

and

$$\rho \vdash e \longrightarrow e'$$

then

$$C; \Gamma \vdash e' : \tau_1' \times^{\alpha'} \tau_2' \tag{190}$$

with $C \Vdash \tau_1' \times^{\alpha'} \tau_2' \leqslant_{\delta\chi} \tau_1 \times^{\alpha} \tau_2$, which—by inversion of the subtyping relation—entails:

$$C \Vdash \tau_1' \leqslant_{\delta\chi} \tau_1 \tag{191}$$
$$C \Vdash \tau_2' \leqslant_{\delta\chi} \tau_2 \tag{192}$$
$$C \Vdash \alpha' \sqsubseteq_{\delta\chi} \alpha \tag{193}$$

Applying the inversion lemma to $C; \Gamma \vdash \mathbf{fst}\ e : \tau_1$ reveals it can only have been constructed by [T-Fst], thus we have

$$C; \Gamma \vdash e : \tau_1 \times^{\alpha} \tau_2 \tag{194}$$
$$C \Vdash \tau_1 \leqslant_{\delta\chi} \tau \tag{195}$$
$$C \Vdash \alpha \sqsubseteq_{\chi} \lceil \tau \rceil \tag{196}$$

We can now apply the induction hypothesis. By applying [T-Fst] with $e \mapsto e'$, $\tau \mapsto \tau$, $\tau_1 \mapsto \tau_1'$, $\tau_2 \mapsto \tau_2'$ and $\alpha \mapsto \alpha'$ we find that

$$C; \Gamma \vdash \mathbf{fst}\ e' : \tau.$$

We still need to check that the preconditions on [T-Fst] hold:

1. $C; \Gamma \vdash e' : \tau_1' \times^{\alpha'} \tau_2'$ follows from (121).
2. $C \Vdash \tau_1' \leqslant_{\delta\chi} \tau$ follows from (191), (195) and the transitivity of the subtype relation.
3. $C \Vdash \alpha' \sqsubseteq_{\chi} \lceil \tau \rceil$ follows from (193), (196) and the transitivity of the subeffect relation.

Case [E-FstPair]: The last three inference rules used in the derivation of $C; \Gamma \vdash \mathbf{fst}\ (\mathbf{close}\ (e_1, e_2)\ \mathbf{in}\ \rho_1) : \tau_1$ must have been [T-Fst], [T-Close] and [T-Pair]. By inversion we find that:

$$C; \Delta \vdash e_1 : \tau_1 \tag{197}$$
$$C; \Delta \vdash e_2 : \tau_2 \tag{198}$$
$$C \Vdash \tau_1 \leqslant_{\delta\chi} \tau \tag{199}$$
$$C \Vdash \alpha \sqsubseteq_{\chi} \lceil \tau \rceil \tag{200}$$
$$C \vdash \Delta \bowtie \rho_1 \tag{201}$$

Applying [T-Bind] with $e \mapsto e_1$, $\sigma \mapsto \tau_1$ and $\rho \mapsto \rho_1$ gives us $C; \Gamma \vdash \mathbf{bind}\ \rho_1\ \mathbf{in}\ e_1 : \tau_1$, with its preconditions following from (201) and (197) $C \Vdash \tau_1 \leqslant_{\delta\chi} \tau$ follows from (199).

Case [E-FstExn]: The last two inference rules used in the derivation of $C; \Gamma \vdash \mathbf{fst}\ \lightning^{\ell} : \tau_1$ must have been [T-Fst] and [T-Exn]. By inversion be find

that:

$$C; \Gamma \vdash \lightning^{\ell} : \tau_1 \times^{\alpha} \tau_2 \tag{202}$$
$$C \Vdash \tau_1 \leqslant_{\delta\chi} \tau \tag{203}$$
$$C \Vdash \alpha \sqsubseteq_{\chi} \lceil \tau \rceil \tag{204}$$
$$C \Vdash \ell \sqsubseteq_{\chi} \lceil \tau_1 \times^{\alpha} \tau_2 \rceil \tag{205}$$

We can simplify (205) to $C \Vdash \ell \sqsubseteq_{\chi} \alpha$ and transitively combine it with (204) to obtain:

$$C \Vdash \ell \sqsubseteq_{\chi} \lceil \tau \rceil \tag{206}$$

We finish the proof by applying [T-Exn] with $\tau \mapsto \tau$, noting that its precondition is satisfied by (206) and $C \Vdash \tau \leqslant_{\delta\chi} \tau$ follows from the reflexivity of the subtype relation.

Case [E-Snd]: As [E-Fst].

Case [E-SndPair]: As [E-FstPair].

Case [E-SndExn]: As [E-FstExn].

Case [E-Case]: We need to show that if

$$C; \Gamma \vdash \mathbf{case}\ e_1\ \mathbf{of}\ \{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} : \tau \tag{207}$$
$$C \vdash \Gamma \bowtie \rho \tag{208}$$

and

$$\rho \vdash \mathbf{case}\ e_1\ \mathbf{of}\ \{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} \longrightarrow \mathbf{case}\ e_1'\ \mathbf{of}\ \{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} \tag{209}$$

then

$$C; \Gamma \vdash \mathbf{case}\ e_1'\ \mathbf{of}\ \{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} : \tau' \tag{210}$$

with

$$C \Vdash \tau' \leqslant_{\delta\chi} \tau. \tag{211}$$

The induction hypothesis states that if $C; \Gamma \vdash e_1 : [\tau_1]^{\alpha_1}$, $C \vdash \Gamma \bowtie \rho$ and $\rho \vdash e_1 \longrightarrow e_1'$ then $C; \Gamma \vdash e_1' : [\tau_1']^{\alpha_1'}$ with $C \Vdash [\tau_1']^{\alpha_1'} \leqslant_{\delta\chi} [\tau_1]^{\alpha_1}$.

Applying the inversion lemma to (207) reveals that it can only have been constructed by [T-Case], thus we have:

$$C; \Gamma \vdash e_1 : [\tau_1]^{\alpha_1} \tag{212}$$
$$C; \Gamma \vdash e_2 : \tau_2 \tag{213}$$
$$C; \Gamma, x_1 : \tau_1, x_2 : [\tau_1]^{\beta} \vdash e_3 : \tau_3 \tag{214}$$
$$C \Vdash \mathbf{N} \sqsubseteq_{\delta} \alpha_1 \vee \exists_{\chi} \alpha_1 \Rightarrow \tau_2 \leqslant_{\delta\chi} \tau \tag{215}$$
$$C \Vdash \mathbf{C} \sqsubseteq_{\delta} \alpha_1 \vee \exists_{\chi} \alpha_1 \Rightarrow \tau_3 \leqslant_{\delta\chi} \tau \tag{216}$$
$$C \Vdash \alpha_1 \sqsubseteq_{\chi} \lceil \tau \rceil \tag{217}$$
$$C \Vdash \mathbf{N} \sqcup \mathbf{C} \sqsubseteq_{\delta} \beta \tag{218}$$
$$C \Vdash \alpha_1 \sqsubseteq_{\chi} \beta \tag{219}$$

We can now apply the induction hypothesis, giving us

$$C; \Gamma \vdash e_1' : [\tau_1']^{\alpha_1'} \tag{220}$$

$$C \Vdash [\tau_1']^{\alpha_1'} \leqslant_{\delta\chi} [\tau_1]^{\alpha_1} \tag{221}$$

We construct the desired result using [T-Case] with $e_1 \mapsto e_1'$ and verify that all the premises hold:

1. Invoking [T-Sub] using (220) and (221) gives us

$$C; \Gamma \vdash e_1' : [\tau_1]^{\alpha_1}.$$

All other cases follow immediately from (213)–(219).

Case [E-CaseNil]: We need to show that if

$$C; \Gamma \vdash \mathbf{case}\ []\ \mathbf{of}\{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} : \tau \tag{222}$$

and $\rho \vdash \mathbf{case}\ []\ \mathbf{of}\{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} \longrightarrow e_2$ then $C; \Gamma \vdash e_2 : \tau'$ with $C \Vdash \tau' \leqslant_{\delta\chi} \tau$.

Applying the inversion lemma to (222) reveals it can only have been constructed by [T-Case], thus we have

$$C; \Gamma \vdash [] : [\tau_1]^{\alpha_1} \tag{223}$$

$$C; \Gamma \vdash e_2 : \tau_2 \tag{224}$$

$$C; \Gamma, x_1 : \tau_1, x_2 : [\tau_1]^{\beta} \vdash e_3 : \tau_3 \tag{225}$$

$$C \Vdash \mathbf{N} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \Rightarrow \alpha_1\tau_2 \leqslant_{\delta\chi} \tau \tag{226}$$

$$C \Vdash \mathbf{C} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \Rightarrow \alpha_1\tau_3 \leqslant_{\delta\chi} \tau \tag{227}$$

$$C \Vdash \alpha_1 \sqsubseteq_\chi \lceil \tau \rceil \tag{228}$$

$$C \Vdash \mathbf{N} \sqcup \mathbf{C} \sqsubseteq_\delta \beta \tag{229}$$

$$C \Vdash \alpha_1 \sqsubseteq_\chi \beta \tag{230}$$

Applying the inversion lemma to (223) reveals it can only have been constructed by [T-Nil], giving us:

$$C \Vdash \mathbf{N} \sqsubseteq_\delta \alpha_1 \tag{231}$$

Using $\vee$-introduction on (231) gives us

$$C \Vdash \mathbf{N} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \tag{232}$$

Using implication elimination on (226) gives us

$$C, \mathbf{N} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Vdash \tau_2 \leqslant_{\delta\chi} \tau \tag{233}$$

Using *modus ponens* on (232) and (233) gives us

$$C \Vdash \tau_2 \leqslant_{\delta\chi} \tau \tag{234}$$

which—together with (225)—concludes the proof.

Case [E-CaseCons]: We need to show that if

$$C; \Gamma \vdash \mathbf{case}\ (\mathbf{close}\ e_1 :: e_1'\ \mathbf{in}\ \rho_1)\ \mathbf{of}\ \{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} : \tau \tag{235}$$

$$C \vdash \Gamma \bowtie \rho \tag{236}$$

and

$$\rho \vdash \mathbf{case}\ (\mathbf{close}\ e_1 :: e_1'\ \mathbf{in}\ \rho_1)\ \mathbf{of}\ \{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} \longrightarrow \mathbf{bind}\ (\rho, x_1 : \mathbf{bind}\ \rho_1\ \mathbf{in}\ e_1, x_2 : \mathbf{bind}\ \rho_1\ \mathbf{in}\ e_1')\ \mathbf{in}\ e_3$$

then

$$C; \Gamma \vdash \mathbf{bind}\ (\rho, x_1 : \mathbf{bind}\ \rho_1\ \mathbf{in}\ e_1, x_2 : \mathbf{bind}\ \rho_1\ \mathbf{in}\ e_1')\ \mathbf{in}\ e_3 : \tau'$$

with

$$C \Vdash \tau' \leqslant_{\delta\chi} \tau.$$

Applying the inversion lemma to (235) reveals that the last inference rule used in its derivation can only have been [T-Case], giving us:

$$C; \Gamma \vdash \mathbf{close}\ e_1 :: e_1'\ \mathbf{in}\ \rho_1 : [\tau_1]^{\beta} \tag{237}$$

$$C; \Gamma \vdash e_2 : \tau_2 \tag{238}$$

$$C; \Gamma, x_1 : \tau_1, x_2 : [\tau_1]^{\beta} \vdash e_3 : \tau_3 \tag{239}$$

$$C \Vdash \mathbf{N} \sqsubseteq_{\delta} \alpha_1 \vee \exists_{\chi}\alpha_1 \Rightarrow \tau_2 \leqslant_{\delta\chi} \tau \tag{240}$$

$$C \Vdash \mathbf{C} \sqsubseteq_{\delta} \alpha_1 \vee \exists_{\chi}\alpha_1 \Rightarrow \tau_3 \leqslant_{\delta\chi} \tau \tag{241}$$

$$C \Vdash \alpha_1 \sqsubseteq_{\chi} \lceil \tau \rceil \tag{242}$$

$$C \Vdash \mathbf{N} \sqcup \mathbf{C} \sqsubseteq_{\delta} \beta \tag{243}$$

$$C \Vdash \alpha_1 \sqsubseteq_{\chi} \beta \tag{244}$$

Applying the inversion lemma to (237) reveals that the last inference rule used in its derivation can only have been [T-Close], giving us:

$$C; \Delta \vdash e_1 :: e_1' : [\tau_1]^{\alpha_1} \tag{245}$$

$$C \vdash \Delta \bowtie \rho_1 \tag{246}$$

Applying the inversion lemma to (245) reveals that the last inference rules used in its derivation can only have been [T-Cons], giving us:

$$C; \Delta \vdash e_1 : \tau_{11} \tag{247}$$

$$C; \Delta \vdash e_1' : [\tau_{12}]^{\alpha_{12}} \tag{248}$$

$$C \Vdash \tau_{11} \leqslant_{\delta\chi} \tau_1 \tag{249}$$

$$C \Vdash \tau_{12} \leqslant_{\delta\chi} \tau_1 \tag{250}$$

$$C \Vdash \mathbf{C} \sqsubseteq_{\delta} \alpha_1 \tag{251}$$

$$C \Vdash \alpha_{12} \sqsubseteq_{\chi} \alpha_1 \tag{252}$$

We construct the desired result using [T-Bind] with $e \mapsto e_3$; $\tau \mapsto \tau$; $\rho \mapsto (\rho, x_1 : \textbf{bind } \rho_1 \textbf{ in } e_1, x_2 : \textbf{bind } \rho_1 \textbf{ in } e_1')$ and $\Delta \mapsto \Gamma, x_1 : \tau_1, x_2 : [\tau_1]^\beta$. We still need to verify all the preconditions on [T-Bind] are satisfied:

1. Using $\vee$-introduction on (251) gives us:

$$C \Vdash \mathbf{C} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \tag{253}$$

Using implication elimination on (241) gives us:

$$C, \mathbf{C} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Vdash \tau_3 \leqslant_{\delta\chi} \tau \tag{254}$$

Using *modus ponens* on (253) and (254) gives us:

$$C \Vdash \tau_3 \leqslant_{\delta\chi} \tau \tag{255}$$

Applying [T-Sub] to (239) and (257) gives us the first premise:

$$C; \Gamma, x_1 : \tau_1, x_2 : [\tau_1]^\beta \vdash e_3 : \tau.$$

2. The premise

$$C \vdash \Gamma, x_1 : \tau_1, x_2 : [\tau_1]^\beta \bowtie \rho, x_1 : \textbf{bind } \rho_1 \textbf{ in } e_1, x_2 : \textbf{bind } \rho_1 \textbf{ in } e_1' \tag{256}$$

can be constructed by applying [EC-Extend] twice. We verify that their preconditions hold:

(a) The premise $C \vdash \Gamma \bowtie \rho$ is an assumption given by the theorem.

(b) The premise $C; \Gamma \vdash \textbf{bind } \rho_1 \textbf{ in } e_1 : \tau_1$ can be constructed by [T-Bind] with $e \mapsto e_1$, $\rho \mapsto \rho_1$, $\tau \mapsto \tau_1$ and $\Delta \mapsto \Delta$. We need to check its preconditions:

i. The premise $C \vdash \Delta \bowtie \rho_1$ is satisfied by (246).

ii. The premise $C; \Delta \vdash e_1 : \tau_1$ can be constructed by applying [T-Sub] to (247) and (249).

(c) The premise $C; \Gamma \vdash \textbf{bind } \rho_1 \textbf{ in } e_1' : \tau_1$ can be constructed by [T-Bind] with $e \mapsto e_1'$, $\rho \mapsto \rho_1$, $\tau \mapsto \tau_1$ and $\Delta \mapsto \Delta$. We need to check its preconditions:

i. The premise $C \vdash \Delta \bowtie \rho_1$ is satisfied by (246).

ii. From [CL-$\top$] we get

$$\alpha_{12} \sqsubseteq_\delta \mathbf{N} \sqcup \mathbf{C} \tag{257}$$

From (257), (243) and the reflexivity of the subeffecting relation it follows that

$$\alpha_{12} \sqsubseteq_\delta \beta \tag{258}$$

From (252), (244) and the transitivity of the subeffecting relation it follows that

$$\alpha_{12} \sqsubseteq_\chi \beta \tag{259}$$

We construct the desired premise $\Delta \vdash e_1' : [\tau_1]^\beta$ by applying [T-Sub] to (248) using (250), (258) and (259).

We conclude the proof by noting that $C \Vdash \tau \leqslant_{\delta\chi} \tau$ follows from the reflexivity of the subtype relation.

Case [E-CaseExn1]: We need to show that if

$$C; \Gamma \vdash \mathbf{case}\ \lightning^{\ell_1}\ \mathbf{of}\{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} : \tau \tag{260}$$

and

$$\rho \ \vdash\ \mathbf{case}\ \lightning^{\ell_1}\ \mathbf{of}\{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} \longrightarrow\ \mathbf{case}\ \lightning^{\ell_1}\ \mathbf{of}\{[] \mapsto e_2'; x_1 :: x_2 \mapsto e_3\} \tag{261}$$

then

$$C; \Gamma \vdash \mathbf{case}\ \lightning^{\ell_1}\ \mathbf{of}\{[] \mapsto e_2'; x_1 :: x_2 \mapsto e_3\} : \tau' \tag{262}$$

with

$$C \Vdash \tau' \leqslant_{\delta\chi} \tau. \tag{263}$$

The induction hypothesis states that if $C; \Gamma \vdash e_2 : \tau_2$ and $\rho \ \vdash\ e_2 \longrightarrow\ e_2'$ then $C; \Gamma \vdash e_2' : \tau_2'$ with $C \Vdash \tau_2' \leqslant_{\delta\chi} \tau_2$.

Applying the inversion lemma to (260) reveals it can only have been constructed by [T-Case], thus we have

$$C; \Gamma \vdash \lightning^{\ell_1} : [\tau_1]^{\alpha_1} \tag{264}$$
$$C; \Gamma \vdash e_2 : \tau_2 \tag{265}$$
$$C; \Gamma, x_1 : \tau_1, x_2 : [\tau_1]^{\beta} \vdash e_3 : \tau_3 \tag{266}$$
$$C \Vdash \mathbf{N} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_2 \leqslant_{\delta\chi} \tau \tag{267}$$
$$C \Vdash \mathbf{C} \sqsubseteq_\chi \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_3 \leqslant_{\delta\chi} \tau \tag{268}$$
$$C \Vdash \alpha_1 \sqsubseteq_\chi \lceil \tau \rceil \tag{269}$$
$$C \Vdash \mathbf{N} \sqcup \mathbf{C} \sqsubseteq_\delta \beta \tag{270}$$
$$C \Vdash \alpha_1 \sqsubseteq_\chi \beta \tag{271}$$

We can now apply the induction hypothesis, giving us:

$$C; \Gamma \vdash e_2' : \tau_2' \tag{272}$$
$$C \Vdash \tau_2' \leqslant_{\delta\chi} \tau_2 \tag{273}$$

Applying the inversion lemma to (264) reveals it can only have been constructed by [T-Exn], giving us:

$$C \Vdash \ell_1 \sqsubseteq_\chi \lceil [\tau_1]^{\alpha_1} \rceil \tag{274}$$

which, by simplification of $\lceil \cdot \rceil$, equals

$$C \Vdash \ell_1 \sqsubseteq_\chi \alpha_1 \tag{275}$$

Using $\exists$-introduction on (275) gives us

$$C \Vdash \exists_\chi \alpha_1 \tag{276}$$

Using $\vee$-introduction on (276) gives us

$$C \Vdash \mathbf{N} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \tag{277}$$

Applying [T-Case] with $e_1 \mapsto \lightning^{\ell_1}$, $e_2 \mapsto e_2'$, $e_3 \mapsto e_3$, $x_1 \mapsto x_1$, $x_2 \mapsto x_2$, $\tau \mapsto \tau$, $\tau_1 \mapsto \tau_1$, $\tau_2 \mapsto \tau_2'$, $\tau_3 \mapsto \tau_3$, $\alpha_1 \mapsto \alpha_1$ and $\beta \mapsto \beta$ we find:

$$C; \Gamma \vdash \mathbf{case}\ \lightning^{\ell_1}\ \mathbf{of}\ \{[] \mapsto e_2'; x_1 :: x_2 \mapsto e_3\} : \tau \tag{278}$$

We still need to check that the preconditions on [T-Case] hold:

1. Using implication elimination on (267) gives us:

$$C, \mathbf{N} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Vdash \tau_2 \leqslant_{\delta\chi} \tau \tag{279}$$

Using *modus ponens* on (277) and (279) gives us:

$$C \Vdash \tau_2 \leqslant_{\delta\chi} \tau \tag{280}$$

From (273), (280) and the transitivity of the subtype relation it follows that:

$$C \Vdash \tau_2' \leqslant_{\delta\chi} \tau \tag{281}$$

Using weakening and implication introduction on (281) gives us:

$$C \Vdash \mathbf{N} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_2' \leqslant_{\delta\chi} \tau \tag{282}$$

which is what was asked for.

The other seven preconditions follow immediately from (264), (272), (266), (268), (269), (270) and (271).

We conclude the proof using (278) and noting that $C \Vdash \tau \leqslant_{\delta\chi} \tau$ follows from the reflexivity of the subtype relation.

Case [E-CaseExn2]: We need to show that if

$$C; \Gamma \vdash \mathbf{case}\ \lightning^{\ell_1}\ \mathbf{of}\ \{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} : \tau, \tag{283}$$

$$C \vdash \Gamma \bowtie \rho \tag{284}$$

and

$$\rho \vdash \mathbf{case}\ \lightning^{\ell_1}\ \mathbf{of}\ \{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} \longrightarrow \mathbf{case}\ \lightning^{\ell_1}\ \mathbf{of}\ \{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3'\} \tag{285}$$

then

$$C; \Gamma \vdash \mathbf{case}\ \lightning^{\ell_1}\ \mathbf{of}\ \{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} : \tau \tag{286}$$

with

$$C \Vdash \tau \leqslant_{\delta\chi} \tau. \tag{287}$$

The induction hypothesis states that if

$$C; \Gamma, x_1 : \tau_1, x_2 : [\tau_1]^\beta \vdash e_3 : \tau_3, \tag{288}$$

$$C \vdash \Gamma, x_1 : \tau_1, x_2 : \tau_2 \bowtie \rho, x_1 : \lightning^\emptyset, x_2 : \lightning^\emptyset \tag{289}$$

and

$$\rho, x_1 : \lightning^\emptyset, x_3 : \lightning^\emptyset \ \vdash \ e_3 \longrightarrow \ e_3' \tag{290}$$

then

$$C; \Gamma, x_1 : \tau_1, x_2 : [\tau_1]^\beta \vdash e_3 : \tau_3' \tag{291}$$

with

$$C \Vdash \tau_3' \leqslant_{\delta\chi} \tau_3. \tag{292}$$

Applying the inversion lemma to (283) reveals that the last rule used to construct it can only have been [T-Case], giving us:

$$C; \Gamma \vdash \lightning^{\ell_1} : [\tau_1]^{\alpha_1} \tag{293}$$

$$C; \Gamma \vdash e_2 : \tau_2 \tag{294}$$

$$C; \Gamma, x_1 : \tau_1, x_2 : [\tau_1]^\beta \vdash e_3 : \tau_3 \tag{295}$$

$$C \Vdash \mathbf{N} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_2 \leqslant_{\delta\chi} \tau \tag{296}$$

$$C \Vdash \mathbf{C} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_3 \leqslant_{\delta\chi} \tau \tag{297}$$

$$C \Vdash \alpha_1 \sqsubseteq_\chi \lceil \tau \rceil \tag{298}$$

$$C \Vdash \mathbf{N} \sqcup \mathbf{C} \sqsubseteq_\delta \beta \tag{299}$$

$$C \Vdash \alpha_1 \sqsubseteq_\chi \beta \tag{300}$$

If we can show that

$$C; \Gamma \vdash \lightning^\emptyset : \tau_1 \tag{301}$$

$$C; \Gamma, x_1 : \tau_1 \vdash \lightning^\emptyset : [\tau_1]^\beta \tag{302}$$

then we can apply [EC-Extend] twice to (284) in order to get

$$C \vdash \Gamma, x_1 : \tau_1, x_2 : [\tau_1]^\beta \bowtie \rho, x_1 : \lightning^\emptyset, x_2 : \lightning^\emptyset. \tag{303}$$

Both (301) and (302) can be constructed using [T-Exn] as their premises $C \Vdash \emptyset \sqsubseteq_\chi \lceil \tau_1 \rceil$ and $C \Vdash \emptyset \sqsubseteq_\chi \beta$ follow immediately from [CL-$\emptyset$].

We can now apply the induction hypothesis using (295), (303) and the premise on [E-CaseExn2]. We can construct (286) using [T-Case] with $e_1 \mapsto \lightning^{\ell_1}$, $e_3 \mapsto e_3'$ and $\tau_3 \mapsto \tau_3'$. We still need to show that its premises are satisfied, though:

1. Using weakening on (292), we get

$$C, \mathbf{C} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Vdash \tau_3' \leqslant_{\delta\chi} \tau_3. \tag{304}$$

Using implication eliminiation on (297), we get

$$C, \mathbf{C} \sqsubseteq_\delta \alpha_1 \vee \exists_\delta \alpha_1 \Vdash \tau_3 \leqslant_{\delta\chi} \tau. \tag{305}$$

From (304), (305) and the transitivity of the subtype relation it follows that

$$C, \mathbf{C} \sqsubseteq_\delta \alpha_1 \vee \exists_\delta \alpha_1 \Vdash \tau_3 \leqslant_{\delta\chi} \tau. \tag{306}$$

Using implication introduction on (306), we obtain the premise

$$C \Vdash \mathbf{C} \sqsubseteq_\delta \alpha_1 \vee \exists_\delta \alpha_1 \Rightarrow \tau_3 \leqslant_{\delta\chi} \tau.$$

The other seven premises follow directly from (291), (293), (294), (296), (298), (299) and (300).

Finally, we observe that (287) follows from the reflexivity of the subtype relation, concluding the proof.

Case [E-CaseExn3]: We need to show that if

$$C; \Gamma \vdash \mathbf{case}\ \lightning^{\ell_1}\ \mathbf{of} \left\{ [] \mapsto v_2^{\ell_2}; x_1 :: x_2 \mapsto v_3^{\ell_3} \right\} : \tau \tag{307}$$

and $\rho \vdash \mathbf{case}\ \lightning^{\ell_1}\ \mathbf{of} \left\{ [] \mapsto v_2^{\ell_2}; x_1 :: x_2 \mapsto v_3^{\ell_3} \right\} \longrightarrow \lightning^{\ell_1 \sqcup \ell_2 \sqcup \ell_3}$ then $C; \Gamma \vdash \lightning^{\ell_1 \sqcup \ell_2 \sqcup \ell_3} : \tau'$ with $C \Vdash \tau' \leqslant_{\delta\chi} \tau$.

Applying the inversion lemma to (307) reveals that it can only have been constructed by [T-Case], thus we have:

$$C; \Gamma \vdash \lightning^{\ell_1} : [\tau_1]^{\alpha_1} \tag{308}$$
$$C; \Gamma \vdash v_2^{\ell_2} : \tau_2 \tag{309}$$
$$C; \Gamma, x_1 : \tau_1, x_2 : [\tau_1]^{\beta} \vdash v_3^{\ell_3} : \tau_3 \tag{310}$$
$$C \Vdash \mathbf{N} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_2 \leqslant_{\delta\chi} \tau \tag{311}$$
$$C \Vdash \mathbf{C} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \Rightarrow \tau_3 \leqslant_{\delta\chi} \tau \tag{312}$$
$$C \Vdash \alpha_1 \sqsubseteq_\chi \lceil \tau \rceil \tag{313}$$
$$C \Vdash \mathbf{N} \sqcup \mathbf{C} \sqsubseteq_\delta \beta \tag{314}$$
$$C \Vdash \alpha_1 \sqsubseteq_\chi \beta \tag{315}$$

Applying the inversion lemma to (308) reveals it can only have been constructed by [T-Exn], giving us:

$$C \Vdash \ell_1 \sqsubseteq_\chi \lceil [\tau_1]^{\alpha_1} \rceil \tag{316}$$

which, after simplification of $\lceil \cdot \rceil$, is equivalent to

$$C \Vdash \ell_1 \sqsubseteq_\chi \alpha_1. \tag{317}$$

Using $\vee$-introduction on (317) gives us:

$$C \Vdash \mathbf{N} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \tag{318}$$
$$C \Vdash \mathbf{C} \sqsubseteq_\delta \alpha_1 \vee \exists_\chi \alpha_1 \tag{319}$$

Using implication elimination and *modus ponens* on both (311) and (318), and, (312) and (319) gives us:

$$C \Vdash \tau_2 \leqslant_{\delta\chi} \tau \tag{320}$$
$$C \Vdash \tau_3 \leqslant_{\delta\chi} \tau \tag{321}$$

We construct—using [T-Exn]—an exceptional value

$$C; \Gamma \vdash \lightning^{\ell_1 \sqcup \ell_2 \sqcup \ell_3} : \tau \tag{322}$$

which needs as a precondition that

$$C \Vdash \ell_1 \sqcup \ell_2 \sqcup \ell_3 \sqsubseteq_\chi \lceil \tau \rceil \tag{323}$$

This precondition can be decomposed into

$$C \Vdash \ell_1 \sqsubseteq_\chi \lceil \tau \rceil \tag{324}$$
$$C \Vdash \ell_2 \sqsubseteq_\chi \lceil \tau \rceil \tag{325}$$
$$C \Vdash \ell_3 \sqsubseteq_\chi \lceil \tau \rceil \tag{326}$$

1. (324) follows (316), (313) and the transitivity of the subtype relation.
2. (325) follows by case analysis on $v_2^{\ell_2}$: if $v_2^{\ell_2}$ is an exceptional value then the inversion lemma reveals (309) must have been constructed by [T-Exn] and we have

   $$C \Vdash \ell_2 \sqsubseteq_\chi \lceil \tau_2 \rceil \tag{327}$$

   Applying Lemma 1 to (320) we get:

   $$C \Vdash \lceil \tau_2 \rceil \sqsubseteq_{\delta\chi} \lceil \tau \rceil \tag{328}$$

   Using the transitivity of the subtype relation on (327) and (328) gives us the result we were asked for.
3. The reasoning for (326) is analogous to (325).

Case [E-Bind1]: We need to show that if

$$C; \Gamma \vdash \mathbf{bind}\ \rho_1\ \mathbf{in}\ e_1 : \sigma_1$$

and

$$\rho \ \vdash\ \mathbf{bind}\ \rho_1\ \mathbf{in}\ e_1 \longrightarrow\ \mathbf{bind}\ \rho_1\ \mathbf{in}\ e_1'$$

then $C; \Gamma \vdash \mathbf{bind}\ \rho_1\ \mathbf{in}\ e_1' : \sigma_2$ with $C \Vdash \sigma_2 \leqslant_{\delta\chi} \sigma_1$. The induction hypothesis states that if $C; \Gamma \vdash e_1 : \sigma_1$ and $\rho \ \vdash\ e_1 \longrightarrow\ e_1'$ then $C; \Gamma \vdash e_1' : \sigma_2$ with

$C \Vdash \sigma_2 \leqslant_{\delta\chi} \sigma_1$. Applying the inversion lemma to $C; \Gamma \vdash \mathbf{bind}\ \rho_1\ \mathbf{in}\ e_1 : \sigma_1$ reveals it can only have been constructed by [T-Bind], thus we have

$$C; \Delta \vdash e_1 : \sigma_1 \tag{329}$$
$$C \vdash \Delta \bowtie \rho_1 \tag{330}$$

Applying the induction hypothesis we get

$$C; \Gamma \vdash e_1' : \sigma_2 \tag{331}$$
$$C \Vdash \sigma_2 \leqslant_{\delta\chi} \sigma_1 \tag{332}$$

Applying [T-Bind] with $e_1 \mapsto e_1'$ and $\sigma_1 \mapsto \sigma_2$ we find that $C; \Gamma \vdash \mathbf{bind}\ \rho_1\ \mathbf{in}\ e_1' : \sigma_2$ with $C \Vdash \sigma_2 \leqslant_{\delta\chi} \sigma_1$. The preconditions on [T-Bind] follow immediately from (331) and (330).

Case [E-Bind2]: We need to show that if

$$C; \Gamma \vdash \mathbf{bind}\ \rho_1\ \mathbf{in}\ v_1^{\ell_1} : \sigma_1$$

and $\rho \vdash \mathbf{bind}\ \rho_1\ \mathbf{in}\ v_1^{\ell_1} \longrightarrow v_1^{\ell_1}$ then $C; \Gamma \vdash v_1^{\ell_1} : \sigma_2$ with $C \Vdash \sigma_2 \leqslant_{\delta\chi} \sigma_1$.

Applying the inversion lemma to $C; \Gamma \vdash \mathbf{bind}\ \rho_1\ \mathbf{in}\ v_1^{\ell_1} : \sigma_1$ reveals it can only have been constructed by [T-Bind], thus we have $C; \Gamma \vdash v_1^{\ell_1} : \sigma_1$, with $C \Vdash \sigma_1 \leqslant_{\delta\chi} \sigma_1$ following from the reflexivity of the subtype relation.