Fully Reflexive Intensional Type Analysis*†

Bratin Saha   Valery Trifonov   Zhong Shao
Department of Computer Science
Yale University
New Haven, CT 06520-8285
{saha,trifonov,shao}@cs.yale.edu

Technical Report YALEU/DCS/TR-1194
March 20, 2000

Abstract

Compilers for polymorphic languages can use runtime type inspection to support advanced implementation techniques such as tagless garbage collection, polymorphic marshalling, and flattened data structures. Intensional type analysis is a type-theoretic framework for expressing and certifying such type-analyzing computations. Unfortunately, existing approaches to intensional analysis do not work well on types with universal, existential, or fixpoint quantifiers. This makes it impossible to code applications such as garbage collection, persistency, or marshalling which must be able to examine the type of any runtime value.

We present a typed intermediate language that supports *fully reflexive* intensional type analysis. By fully reflexive, we mean that type-analyzing operations are applicable to the type of any runtime value in the language. In particular, we provide both type-level and term-level constructs for analyzing quantified types. Our system supports structural induction on quantified types yet type checking remains decidable. We show how to use reflexive type analysis to support type-safe marshalling and how to generate certified type-analyzing object code.

Keywords: certified code, runtime type dispatch, typed intermediate language.

1 Introduction

Runtime type analysis is used extensively in various applications and programming situations. Runtime services such as garbage collection and dynamic linking, applications such as marshalling and pickling, type-safe persistent programming, and unboxing implementations of polymorphic languages all analyze types to various degrees at runtime. Most existing compilers use untyped intermediate languages for compilation; therefore, they support runtime type inspection in a type-unsafe manner. In this paper, we present a statically typed intermediate language that allows runtime type analysis to be coded within the language. This allows us to leverage the power of dynamically typed languages, yet retain the advantages of static type checking.

Supporting runtime type analysis in a type-safe manner has been an active area of research. This paper builds on existing work [7] but makes the following new contributions:

- *We support fully reflexive type analysis at the term level.* Consequently, programs can analyze any runtime value such as function closures and polymorphic data structures.
- *We support fully reflexive type analysis at the type level.* Therefore, type transformations operating on arbitrary types can be encoded in our language.
- *We prove that the language is sound and that type reduction is strongly normalising and confluent.*
- *We show a translation into a type erasure semantics.* In a type preserving compiler, this simplifies typed closure conversion which allows generation of certified object code.

2 Motivation

The core issue that we address in this paper is the design of a statically typed intermediate language that supports runtime type analysis. Why is this important? Modern programming paradigms are increasingly giving rise to applications that rely critically on type information at runtime, for example:

- Java adopts dynamic linking as a key feature, and to ensure safe linking, an external module must be dynamically verified to satisfy the expected interface type.
- A garbage collector must keep track of all live heap objects, and for that type information must be kept at runtime to allow traversal of data structures.
- In a distributed computing environment, code and data on one machine may need to be pickled for transmission to a different machine, where the unpickler reconstructs the data structures from the bit stream. If the type of the data is not statically known at the destination (as is the case for the environment components of function closures), the unpickler must use type information, encoded in the bit stream, to correctly interpret the encoded value.
- Type-safe persistent programming also requires language support for dynamic typing: while reading data from a persistent store, the program must ensure that the data is of the expected type.

*This research was sponsored in part by the Defense Advanced Research Projects Agency ISO under the title "Scaling Proof-Carrying Code to Production Compilers and Security Policies," ARPA Order No. H559, issued under Contract No. F30602-99-1-0519, and in part by NSF Grants CCR-9633390 and CCR-9901011. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

†Extended version

- Finally, in polymorphic languages like ML, the type of a value may not be known statically; therefore, compilers have traditionally used inefficient, uniformly boxed data representation. To avoid this, several modern compilers [23, 19, 25] use runtime type information to support unboxed data representation.

Most existing compilers cannot typecheck the code that involves runtime type inspections. They use untyped intermediate languages, and reify runtime types into values at some early stage. However, since type information is lost during the compilation, this approach is at a serious disadvantage when it comes to generating certified code [13].

Code certification is appealing for a number of reasons. If a compiler can generate certified code, we no longer need to trust the correctness of the compiler; instead, we can verify the correctness of the generated code. Checking the correctness of a compiler-generated proof (of a program property) is much easier than proving the correctness of the compiler. Secondly, with the growth of web-based computing, programs are increasingly being developed at remote sites and shipped to clients for execution. Client programs may also download modules dynamically as they need them. For such a system to be practical, a client should be able to accept code from untrusted sources, but have a means of verifying it before execution. This again requires compilers that generate certified code.

A necessary step in building a certifying compiler is to have the compiler generate code that can be type-checked before execution. The type system ensures that the code accesses only the provided resources, makes legal function calls, etc. A certifying compiler can support runtime type analysis only in a typed framework.

The safety of such a system depends not only on the downloaded code, but also on the correctness of all the code that is executed by the system after type checking. This typically includes the runtime services like garbage collection, linking, etc. This code constitutes the trusted computing base of the system. Reducing the trusted computing base makes the system more reliable; for this, we must independently verify the correctness of this code. This implies that as many of the runtime services as possible should be written in a type-safe language, which requires support for runtime type analysis in a typed framework.

Finally, why is it important to have fully reflexive type analysis? Why do we want to analyze quantified types? Many type-analyzing applications mentioned above must handle arbitrary runtime values. For example, a pickler must be able to pickle any value, including closures (which have existential types), polymorphic functions, or recursive data structures. A garbage collector has to be able to traverse all data structures in the heap to track live objects. Therefore the language must support type analysis over any runtime value in the language.

2.1 Background

Harper and Morrisett [7] proposed intensional type analysis and presented a type-theoretic framework for expressing computations that analyze types at runtime. They introduced two explicit type-analysis operators: one for the term level (i.e., typecase) and another for the type level (i.e., Typerec); both use induction over the structure of types. Type-dependent primitive functions use these operators to analyze types and select the appropriate code. For example, a polymorphic subscript function for arrays might be written as the following pseudo-code:

$$\begin{array}{l} \mathsf{sub} = \Lambda\alpha.\,\mathsf{typecase}\;\alpha\;\mathsf{of} \\ \qquad\qquad \begin{array}{ll} \mathsf{int} & \Rightarrow \mathsf{intsub} \\ \mathsf{real} & \Rightarrow \mathsf{realsub} \\ \beta & \Rightarrow \mathsf{boxedsub}\,[\beta] \end{array} \end{array}$$

$$\begin{array}{llcl} (\mathit{kinds}) & \kappa & ::= & \Omega \mid \kappa \to \kappa' \\ (\mathit{cons}) & \tau & ::= & \mathsf{int} \mid \tau \to \tau' \mid \alpha \mid \lambda\alpha{:}\kappa.\,\tau \mid \tau\,\tau' \\ & & \mid & \mathsf{Typerec}\;\tau\;\mathsf{of}\;(\tau_{\mathsf{int}};\,\tau_{\to}) \\ (\mathit{types}) & \sigma & ::= & \tau \mid \forall\alpha{:}\kappa.\,\sigma \end{array}$$

Figure 1: The Harper-Morrisett type language

Here sub analyzes the type α of the array elements and returns the appropriate subscript function. We assume that arrays of type int and real have specialized representations, say intarray and realarray (and therefore special subscript functions); and all other arrays use the default boxed representation.

Typing this subscript function is more interesting, because it must have all of the types $\mathsf{intarray} \to \mathsf{int} \to \mathsf{int}$, $\mathsf{realarray} \to \mathsf{int} \to \mathsf{real}$, and $\forall\alpha.\,\mathsf{boxedarray}\,(\alpha) \to \mathsf{int} \to \alpha$. To assign a type to the subscript function, we need a construct at the type level that parallels the typecase analysis at the term level. In general, this facility is crucial since many type-analyzing operations like flattening and marshalling transform types in a non-uniform way. The subscript operation would then be typed as

$$\begin{array}{rrcl} & \mathsf{sub} & : & \forall\alpha.\,\mathsf{Array}\,(\alpha) \to \mathsf{int} \to \alpha \\ \text{where} & \mathsf{Array} & = & \lambda\alpha.\,\mathsf{Typecase}\;\alpha\;\mathsf{of} \\ & & & \qquad \begin{array}{ll} \mathsf{int} & \Rightarrow \mathsf{intarray} \\ \mathsf{real} & \Rightarrow \mathsf{realarray} \\ \beta & \Rightarrow \mathsf{boxedarray}\,\beta \end{array} \end{array}$$

The Typecase construct in the above example is a special case of the Typerec construct in Harper and Morrisett [7] which also supports primitive recursion over types.

2.2 The problem

Harper and Morrisett's system [7] only allows the analysis of monotypes; it does not support analysis of types with binding structure (e.g., polymorphic, existential or recursive types). Therefore, type analyzing primitives that handle polymorphic code blocks, closures (since closures are represented as existentials [11]), or recursive structures, cannot be written in their language. The types in their language (in essence shown in Figure 1) are separated into two universes, *constructors* and *types*. The constructor calculus is a simply typed lambda calculus, with no polymorphic types. The Typerec operator analyzes only constructors of base kind Ω:

$$\begin{array}{lcl} \mathsf{int} & : & \Omega \\ \to & : & \Omega \to \Omega \to \Omega \end{array}$$

The kinds of these constructors' arguments do not contain any negative occurence of the kind Ω, so int and $\to$ can be used to define Ω inductively. The Typerec operator is essentially an iterator over this inductive definition; its reduction rules can be written as:

$$\begin{array}{l} \mathsf{Typerec}\;\mathsf{int}\;\mathsf{of}\;(\tau_{\mathsf{int}};\,\tau_{\to}) \leadsto \tau_{\mathsf{int}} \\ \mathsf{Typerec}\;(\tau_1 \to \tau_2)\;\mathsf{of}\;(\tau_{\mathsf{int}};\,\tau_{\to}) \leadsto \\ \quad \tau_{\to}\;\tau_1\;\tau_2\;(\mathsf{Typerec}\;\tau_1\;\mathsf{of}\;(\tau_{\mathsf{int}};\,\tau_{\to}))\;(\mathsf{Typerec}\;\tau_2\;\mathsf{of}\;(\tau_{\mathsf{int}};\,\tau_{\to})) \end{array}$$

Here the Typerec operator examines the head constructor of the type being analyzed and chooses a branch accordingly. If the type is int, it reduces to the τ_{int} branch. If the type is $\tau_1 \to \tau_2$, the analysis proceeds recursively on the subtypes τ_1 and τ_2. The Typerec operator then applies the $\tau_{\to}$ branch to the original component types, and to the result of analysing the components; thus providing a form of primitive recursion.

Types with binding structure can be constructed using higher-order abstract syntax. For example, the polymorphic type constructor $\forall$ can be given the kind $(\Omega \to \Omega) \to \Omega$, so that the type $\forall\alpha : \Omega.\, \alpha \to \alpha$ is represented as $\forall\,(\lambda\alpha : \Omega.\, \alpha \to \alpha)$. It would seem plausible to define an iterator with the reduction rule:

$$\begin{array}{l}\mathsf{Typerec}\ (\forall\,\tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall})\\ \leadsto\ \tau_{\forall}\,\tau\,(\lambda\alpha{:}\Omega.\,\mathsf{Typerec}\ \tau\,\alpha\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall}))\end{array}$$

However the negative occurence of Ω in the kind of the argument of $\forall$ poses a problem: this iterator may fail to terminate! Consider the following example, assuming $\tau = \lambda\alpha{:}\Omega.\,\alpha$ and

$$\tau_{\forall} = \lambda\beta_1 : \Omega \to \Omega.\, \lambda\beta_2 : \Omega \to \Omega.\, \beta_2\,(\forall\beta_1)$$

the following reduction sequence will go on indefinitely:

$$\begin{array}{l}\mathsf{Typerec}\ (\forall\tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall})\\ \leadsto\ \tau_{\forall}\,\tau\,(\lambda\alpha{:}\Omega.\,\mathsf{Typerec}\ \tau\,\alpha\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall}))\\ \leadsto\ \mathsf{Typerec}\ (\tau\,(\forall\,\tau))\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall})\\ \leadsto\ \mathsf{Typerec}\ (\forall\tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall})\\ \leadsto\ \ldots\end{array}$$

Clearly this makes typechecking Typerec undecidable.

Another serious problem in analysing quantified types involves both the type-level and the term-level operators. Typed intermediate languages like FLINT [20] and TIL [24] are based on the calculus F_ω [5, 18], which has higher order type constructors. In a quantified type, say $\exists\alpha{:}\kappa.\,\tau$, the quantified variable α is no longer restricted to a base kind Ω, but can have an arbitrary kind κ. Consider the term level typecase in such a scenario:

$$\begin{array}{lll}\mathsf{sub} = \Lambda\alpha.\,\mathsf{typecase}\ \alpha\ \mathsf{of} & & \\ & \mathsf{int} & \Rightarrow e_{\mathsf{int}}\ldots \\ & & \\ & \exists\alpha{:}\kappa.\,\tau & \Rightarrow e_{\exists}\end{array}$$

To do anything useful in the $e_\exists$ branch, even to open a package of this type, we need to know the kind κ. We can get around this by having an infinite number of branches in the typecase, one for each kind; or by restricting type analysis to a finite set of kinds. Both of these approaches are clearly impractical. Recent work on typed compilation of ML and Java has shown that both would require an F_ω-like calculus with arbitrarily complex kinds [21, 22, 9].

2.3 Requirements for a solution

Before we discuss our solution, let us look at the properties we want it to have.

First, our language must support type analysis in the manner of Harper and Morrisett [7]. That is, we want to include type analysis primitives that will analyze the entire syntax tree representing a type. Second, we want the analysis to continue inside the body of a quantified type; handling quantified types parametrically, or in a uniform way by providing a default case, is insufficient. As we will see later, many interesting type-directed operations require these two properties. Third, we do not want to restrict the kind of the (quantified) type variable in a quantified type; we want to analyze types where the quantification is over a variable of arbitrary kind.

Consider a type-directed pickler that converts a value of arbitrary type into an external representation. Suppose we want to pickle a closure. With a type-preserving compiler, the type of a closure would be represented as an existential with the environment held abstract. Even if the code is handled uniformly, the function must inspect the type of the environment (which is also the witness type of the existential package) to pickle it. This shows that at the term level, the analysis must proceed inside a quantified type. In Section 3.3, we show the encoding of a polymorphic equality function in our calculus; the comparison of existential values requires a similar technique.

The reason for not restricting the quantified type variable to a finite set of kinds is twofold. Restricting type analysis to a finite number of kinds would be ad hoc and there is no way of satisfactorily predetermining this finite set (this is even more of the case when we compile Java into a typed intermediate language [9]). More importantly, if the kind of the bound variable is a known constant in the corresponding branch of the Typerec construct, it is easy to generalize the non-termination example of the previous section and break the decidability of the type system.

2.4 Our solution

The key problem in analysing quantified types such as the polymorphic type $\forall\alpha : \Omega.\, \alpha \to \alpha$ is to determine what happens when the iteration reaches the quantified type variable α, or (in the general case of type variables of higher kinds) a normal form which is an application with a type variable in the head.

One approach would be to leave the type variable untouched while analyzing the body of the quantified type. The equational theory of the type language then includes the reduction of form $(\mathsf{Typerec}\ \alpha\ \mathsf{of}\ \ldots) \leadsto \alpha$ so that the iterator vanishes when it reaches a type variable. However this would break the confluence of the type language, since the application $(\lambda\alpha : \Omega.\, \mathsf{Typerec}\ \alpha\ \mathsf{of} \ldots)\,\tau'$ would reduce in general to different types depending on whether we first perform the β-reduction or eliminate the iterator.

Crary and Weirich [1] propose another method for solving this problem. Their language LX allows the representation of terms with bound variables using deBruijn notation and an encoding of natural numbers as types. To analyze quantified types, the iterator carries an environment mapping indices to types; when the iterator reaches a type variable, it returns the corresponding type from the environment. This method has several disadvantages:

- First, their method is not fully reflexive since they cannot analyze all quantified types—their analysis is restricted to types where the quantification is only over variables of kind Ω.
- Second, their technique is "limited to parametrically polymorphic functions, and cannot account for functions that perform intensional type analysis" [1, Section 4.1]. For example, polymorphic types such as $\forall\alpha{:}\Omega.\,(\mathsf{Typerec}\ \alpha\ \mathsf{of}\ \ldots)$ are not analyzable in their framework.
- Third, the correctness of the structure of a type encoded using deBruijn notation cannot be verified by the kind language (indices not corresponding to bound variables go undetected, so the environment must provide a default type for them), which does not break the type soundness but opens the door for programmer mistakes.

To account for non-parametrically polymorphic functions, we must analyze the quantified type variable. Moreover, we want to have confluence of the type language, so β-reduction should be transparent to the iterator. This is possible only if the analysis gets suspended when it reaches a type variable (or its application of base kind) and resumes when the variable gets instantiated. Therefore, we consider $(\mathsf{Typerec}\ \alpha\ \mathsf{of}\ \ldots)$ to be a normal form. For example, the result of analyzing the body $(\alpha \to \mathsf{int})$ of the polymorphic type

$\forall\alpha\!:\!\kappa.\,\alpha \rightarrow \mathsf{int}$ is

$$\begin{array}{l}\mathsf{Typerec}\ (\alpha \rightarrow \mathsf{int})\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall}) \leadsto \\ \tau_{\rightarrow}\ \alpha\ \mathsf{int}\ (\mathsf{Typerec}\ \alpha\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall}))\ (\tau_{\mathsf{int}})\end{array}$$

We formalize the analysis of quantified types when we present the type reduction rules of the Typerec construct (Figure 5).

The other problem is to analyze quantified types when the quantified variable can be of an arbitrary kind. In our language the solution is similar at both the type and the term levels: we use kind polymorphism! We introduce kind abstractions at the type level ($\Lambda\chi.\,\tau$) and at the term level ($\Lambda^{+}\chi.\,e$) to bind the kind of the quantified variable. (See Section 3 for details.)

Kind polymorphism also ensures the termination of the Typerec constructor. Consider again the analysis of the polymorphic type:

$$\begin{array}{l}\mathsf{Typerec}\ (\forall\,\tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall}) \\ \leadsto\ \tau_{\forall}\ \tau\ (\lambda\alpha\!:\!\Omega.\ \mathsf{Typerec}\ \tau\ \alpha\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall}))\end{array}$$

Informally, we must ensure that the type being analysed decreases in size at every iteration. That is $\tau\alpha$ is smaller than $\forall\tau$. (Note that the previous non-terminating example violates this requirement). This will be true if we can ensure that α is always substituted by a single variable. Therefore, we make the kind of α abstract by using kind polymorphism; α now has the kind bound in the $\tau_{\forall}$ branch. The only way to construct another type of this kind is to bind a type variable of the same kind in the $\tau_{\forall}$ branch. This ensures that α can only be substituted by another type variable.

It is important to note that our language provides no facilities for kind analysis. Analyzing the kind κ of the bound variable α in the type $\forall\,(\lambda\alpha:\kappa.\,\tau)$ would let us synthesize a type argument of the same kind, for every kind κ. The synthesized type can then be used in the style of the non-termination example of the previous section. Intuitively, we would not be able to guarantee that the type being analysed decreases at every step.

The rest of the paper is organized as follows. Section 3 describes the language λ_i^P supporting analysis of polymorphic and existential types. Section 4 presents the language λ_i^Q that also includes support for analysis of recursive types. Section 5 shows a translation into a language with type erasure semantics.

3 Analyzing polymorphic types

3.1 The language λ_i^P

In the impredicative F_ω calculus, the polymorphic types $\forall\alpha:\kappa.\,\tau$ can be viewed as generated by an infinite set of type constructors $\forall_\kappa$ of kind $(\kappa \rightarrow \Omega) \rightarrow \Omega$, one for each kind κ. The type $\forall\alpha\!:\!\kappa.\,\tau$ is then represented as $\forall_\kappa\,(\lambda\alpha\!:\!\kappa.\,\tau)$. The kinds of constructors that can generate types of kind Ω then would be

$$\begin{array}{lcl}\mathsf{int} & : & \Omega \\ \twoheadrightarrow & : & \Omega \rightarrow \Omega \rightarrow \Omega \\ \forall_\Omega & : & (\Omega \rightarrow \Omega) \rightarrow \Omega \\ \ldots & & \\ \forall_\kappa & : & (\kappa \rightarrow \Omega) \rightarrow \Omega \\ \ldots & & \end{array}$$

We can avoid the infinite number of $\forall_\kappa$ constructors by defining a single constructor $\forall$ of polymorphic kind $\forall\chi.\,(\chi \rightarrow \Omega) \rightarrow \Omega$ and then instantiating it to a specific kind before forming polymorphic types. More importantly, this technique also removes the negative occurrence of Ω from the kind of the argument of the constructors $\forall_\Omega$. Hence in our new λ_i^P calculus (see Figure 2 to 3), we extend F_ω with polymorphic kinds and add a type constant $\forall$ of kind $\forall\chi.\,(\chi \rightarrow \Omega) \rightarrow \Omega$ to the type language. The polymorphic type $\forall\alpha:\kappa.\,\tau$ is now represented as $\forall\,[\kappa]\,(\lambda\alpha:\kappa.\,\tau)$. Existential quantifiers can be treated in the same way.

$$\begin{array}{llcl}(\mathit{kinds}) & \kappa & ::= & \Omega \mid \kappa \rightarrow \kappa' \mid \chi \mid \forall\chi.\,\kappa \\ \\ (\mathit{types}) & \tau & ::= & \mathsf{int} \mid \twoheadrightarrow \mid \forall \mid \forall^{+} \\ & & \mid & \alpha \mid \Lambda\chi.\,\tau \mid \lambda\alpha\!:\!\kappa.\,\tau \mid \tau\,[\kappa] \mid \tau\,\tau' \\ & & \mid & \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^{+}}) \\ \\ (\mathit{values}) & v & ::= & i \mid \Lambda^{+}\chi.\,e \mid \Lambda\alpha\!:\!\kappa.\,e \mid \lambda x\!:\!\tau.\,e \mid \mathsf{fix}\,x\!:\!\tau.\,v \\ \\ (\mathit{terms}) & e & ::= & v \mid x \mid e\,[\kappa]^{+} \mid e\,[\tau] \mid e\,e' \\ & & \mid & \mathsf{typecase}[\tau]\ \tau'\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\rightarrow};\ e_{\forall};\ e_{\forall^{+}})\end{array}$$

Figure 2: Syntax of the λ_i^P language

$$\begin{array}{c}\tau \rightarrow \tau' \equiv ((\twoheadrightarrow)\,\tau)\,\tau' \\ \forall\alpha\!:\!\kappa.\,\tau \equiv (\forall\,[\kappa])\,(\lambda\alpha\!:\!\kappa.\,\tau) \\ \forall^{+}\chi.\,\tau \equiv \forall^{+}(\Lambda\chi.\,\tau)\end{array}$$

Figure 3: Syntactic sugar for λ_i^P types

We define the syntax of the λ_i^P calculus in Figure 2. The static semantics of λ_i^P uses the following three environments:

$$\begin{array}{llcl}\text{kind environment} & \mathcal{E} & ::= & \varepsilon \mid \mathcal{E},\chi \\ \text{type environment} & \Delta & ::= & \varepsilon \mid \Delta,\alpha\!:\!\kappa \\ \text{value environment} & \Gamma & ::= & \varepsilon \mid \Gamma,x\!:\!\tau\end{array}$$

The detailed typing rules are given in Figures 4 and 5.

The Typerec operator analyzes polymorphic types with bound variables of arbitrary kind. The corresponding branch of the operator must bind the kind of the quantified type variable; we therefore provide kind abstraction ($\Lambda\chi.\,\tau$) and kind application ($\tau\,[\kappa]$) at the type level. The formation rules for these constructs, excerpted from Figure 4, are

$$\frac{\mathcal{E},\chi;\Delta \vdash \tau\ :\ \kappa}{\mathcal{E};\Delta \vdash \Lambda\chi.\,\tau\ :\ \forall\chi.\,\kappa} \qquad \frac{\mathcal{E};\Delta \vdash \tau\ :\ \forall\chi.\,\kappa \qquad \mathcal{E} \vdash \kappa'}{\mathcal{E};\Delta \vdash \tau\,[\kappa']\ :\ \kappa\{\kappa'/\chi\}}$$

Similarly, while analyzing a polymorphic type, the term level typecase operator must bind the kind of the quantified type variable. Therefore, we introduce kind abstraction ($\Lambda^{+}\chi.\,e$) and kind application ($e\,[\kappa]^{+}$) at the term level. To type the term level kind abstraction, we need a type construct $\forall^{+}\chi.\,\tau$ that binds the kind variable χ in the type τ. The formation rules are shown below (Γ denotes the variable environment).

$$\frac{\mathcal{E},\chi;\Delta;\Gamma \vdash v\ :\ \tau}{\mathcal{E};\Delta;\Gamma \vdash \Lambda^{+}\chi.\,v\ :\ \forall^{+}\chi.\,\tau} \qquad \frac{\mathcal{E};\Delta;\Gamma \vdash e\ :\ \forall^{+}\chi.\,\tau \quad \mathcal{E} \vdash \kappa}{\mathcal{E};\Delta;\Gamma \vdash e\,[\kappa]^{+}\ :\ \tau\{\kappa/\chi\}}$$

However, since our goal is fully reflexive type analyses, we need to analyze kind polymorphic types as well. As with polymorphic types, we can represent the type $\forall^{+}\chi.\,\tau$ as the application of a type constructor $\forall^{+}$ of kind $(\forall\chi.\,\Omega) \rightarrow \Omega$ to a kind abstraction $\Lambda\chi.\,\tau$. Thus the kinds of the λ_i^P constructors generating types of

Kind formation $\quad \mathcal{E} \vdash \kappa$

$$\mathcal{E} \vdash \Omega \qquad \frac{\chi \in \mathcal{E}}{\mathcal{E} \vdash \chi} \qquad \frac{\mathcal{E} \vdash \kappa \quad \mathcal{E} \vdash \kappa'}{\mathcal{E} \vdash \kappa \to \kappa'} \qquad \frac{\mathcal{E}, \chi \vdash \kappa}{\mathcal{E} \vdash \forall \chi.\, \kappa}$$

Type environment formation $\quad \mathcal{E} \vdash \Delta$

$$\mathcal{E} \vdash \varepsilon \qquad \frac{\mathcal{E} \vdash \Delta \quad \mathcal{E} \vdash \kappa}{\mathcal{E} \vdash \Delta, \alpha{:}\kappa}$$

Type formation $\quad \mathcal{E}; \Delta \vdash \tau \,:\, \kappa$

$$\frac{\mathcal{E} \vdash \Delta}{\begin{array}{lll} \mathcal{E};\Delta \vdash \mathsf{int} & : & \Omega \\ \mathcal{E};\Delta \vdash (\twoheadrightarrow) & : & \Omega \to \Omega \to \Omega \\ \mathcal{E};\Delta \vdash \boldsymbol{\forall} & : & \forall \chi.\, (\chi \to \Omega) \to \Omega \\ \mathcal{E};\Delta \vdash \boldsymbol{\forall}^{+} & : & (\forall \chi.\, \Omega) \to \Omega \end{array}} \qquad \frac{\mathcal{E} \vdash \Delta \quad \alpha{:}\kappa \text{ in } \Delta}{\mathcal{E};\Delta \vdash \alpha \,:\, \kappa}$$

$$\frac{\mathcal{E}, \chi; \Delta \vdash \tau \,:\, \kappa}{\mathcal{E};\Delta \vdash \Lambda \chi.\, \tau \,:\, \forall \chi.\, \kappa} \qquad \frac{\mathcal{E};\Delta \vdash \tau \,:\, \forall \chi.\, \kappa \quad \mathcal{E} \vdash \kappa'}{\mathcal{E};\Delta \vdash \tau\,[\kappa'] \,:\, \kappa\{\kappa'/\chi\}}$$

$$\frac{\mathcal{E};\Delta, \alpha{:}\kappa \vdash \tau \,:\, \kappa'}{\mathcal{E};\Delta \vdash \lambda \alpha{:}\kappa.\, \tau \,:\, \kappa \to \kappa'} \qquad \frac{\mathcal{E};\Delta \vdash \tau \,:\, \kappa' \to \kappa \quad \mathcal{E};\Delta \vdash \tau' \,:\, \kappa'}{\mathcal{E};\Delta \vdash \tau\, \tau' \,:\, \kappa}$$

$$\frac{\begin{array}{lll} \mathcal{E};\Delta \vdash \tau & : & \Omega \\ \mathcal{E};\Delta \vdash \tau_{\mathsf{int}} & : & \kappa \\ \mathcal{E};\Delta \vdash \tau_{\to} & : & \Omega \to \Omega \to \kappa \to \kappa \to \kappa \\ \mathcal{E};\Delta \vdash \tau_{\forall} & : & \forall \chi.\, (\chi \to \Omega) \to (\chi \to \kappa) \to \kappa \\ \mathcal{E};\Delta \vdash \tau_{\forall^{+}} & : & (\forall \chi.\, \Omega) \to (\forall \chi.\, \kappa) \to \kappa \end{array}}{\mathcal{E};\Delta \vdash \mathsf{Typerec}[\kappa]\ \tau \text{ of } (\tau_{\mathsf{int}};\, \tau_{\to};\, \tau_{\forall};\, \tau_{\forall^{+}}) \,:\, \kappa}$$

Term environment formation $\quad \mathcal{E}; \Delta \vdash \Gamma$

$$\frac{\mathcal{E} \vdash \Delta}{\mathcal{E};\Delta \vdash \varepsilon} \qquad \frac{\mathcal{E};\Delta \vdash \Gamma \quad \mathcal{E};\Delta \vdash \tau \,:\, \Omega}{\mathcal{E};\Delta \vdash \Gamma, x{:}\tau}$$

Term formation $\quad \mathcal{E}; \Delta; \Gamma \vdash e \,:\, \tau$

$$\frac{\mathcal{E};\Delta;\Gamma \vdash e \,:\, \tau \quad \mathcal{E};\Delta \vdash \tau \leadsto \tau' \,:\, \Omega}{\mathcal{E};\Delta;\Gamma \vdash e \,:\, \tau'} \qquad \frac{\mathcal{E};\Delta \vdash \Gamma}{\mathcal{E};\Delta;\Gamma \vdash i \,:\, \mathsf{int}}$$

$$\frac{\mathcal{E};\Delta \vdash \Gamma \quad x{:}\tau \text{ in } \Gamma}{\mathcal{E};\Delta;\Gamma \vdash x \,:\, \tau} \qquad \frac{\mathcal{E}, \chi;\Delta;\Gamma \vdash v \,:\, \tau}{\mathcal{E};\Delta;\Gamma \vdash \Lambda^{+}\chi.\, v \,:\, \forall^{+}\chi.\, \tau}$$

$$\frac{\mathcal{E};\Delta, \alpha{:}\kappa;\Gamma \vdash v \,:\, \tau}{\mathcal{E};\Delta;\Gamma \vdash \Lambda \alpha{:}\kappa.\, v \,:\, \forall \alpha{:}\kappa.\, \tau} \qquad \frac{\mathcal{E};\Delta;\Gamma, x{:}\tau \vdash e \,:\, \tau'}{\mathcal{E};\Delta;\Gamma \vdash \lambda x{:}\tau.\, e \,:\, \tau \to \tau'}$$

$$\frac{\begin{array}{c} \mathcal{E};\Delta;\Gamma, x{:}\tau \vdash v \,:\, \tau \\ \tau = \forall^{+}\chi_1 \ldots \chi_n.\, \forall \alpha_1{:}\kappa_1 \ldots \alpha_m{:}\kappa_m{:}\tau_1 \to \tau_2. \\ n \geq 0, m \geq 0 \end{array}}{\mathcal{E};\Delta;\Gamma \vdash \mathsf{fix}\, x{:}\tau.\, v \,:\, \tau}$$

$$\frac{\mathcal{E};\Delta;\Gamma \vdash e \,:\, \boldsymbol{\forall}^{+}\tau \quad \mathcal{E} \vdash \kappa}{\mathcal{E};\Delta;\Gamma \vdash e\,[\kappa]^{+} \,:\, \tau\,[\kappa]}$$

$$\frac{\mathcal{E};\Delta;\Gamma \vdash e \,:\, \boldsymbol{\forall}\,[\kappa]\,\tau \quad \mathcal{E};\Delta \vdash \tau' \,:\, \kappa}{\mathcal{E};\Delta;\Gamma \vdash e\,[\tau'] \,:\, \tau\,\tau'}$$

$$\frac{\mathcal{E};\Delta;\Gamma \vdash e \,:\, \tau' \to \tau \quad \mathcal{E};\Delta;\Gamma \vdash e' \,:\, \tau'}{\mathcal{E};\Delta;\Gamma \vdash e\, e' \,:\, \tau}$$

$$\frac{\begin{array}{lll} \mathcal{E};\Delta \vdash \tau & : & \Omega \to \Omega \\ \mathcal{E};\Delta \vdash \tau' & : & \Omega \\ \mathcal{E};\Delta;\Gamma \vdash e_{\mathsf{int}} & : & \tau\, \mathsf{int} \\ \mathcal{E};\Delta;\Gamma \vdash e_{\to} & : & \forall \alpha{:}\Omega.\, \forall \alpha'{:}\Omega.\, \tau\,(\alpha \to \alpha') \\ \mathcal{E};\Delta;\Gamma \vdash e_{\forall} & : & \forall^{+}\chi.\, \forall \alpha{:}\chi \to \Omega.\, \tau\,(\boldsymbol{\forall}\,[\chi]\,\alpha) \\ \mathcal{E};\Delta;\Gamma \vdash e_{\forall^{+}} & : & \forall \alpha{:}(\forall \chi.\, \Omega).\, \tau\,(\boldsymbol{\forall}^{+}\alpha) \end{array}}{\mathcal{E};\Delta;\Gamma \vdash \mathsf{typecase}[\tau]\ \tau' \text{ of } (e_{\mathsf{int}};\, e_{\to};\, e_{\forall};\, e_{\forall^{+}}) \,:\, \tau\,\tau'}$$

Figure 4: Formation rules of λ_i^P

Type reduction $\quad \mathcal{E}; \Delta \vdash \tau_1 \leadsto \tau_2 \,:\, \kappa$

$$\frac{\mathcal{E};\Delta, \alpha{:}\kappa' \vdash \tau \,:\, \kappa \quad \mathcal{E};\Delta \vdash \tau' \,:\, \kappa'}{\mathcal{E};\Delta \vdash (\lambda \alpha{:}\kappa'.\, \tau)\, \tau' \leadsto \tau\{\tau'/\alpha\} \,:\, \kappa}$$

$$\frac{\mathcal{E}, \chi;\Delta \vdash \tau \,:\, \forall \chi.\, \kappa \quad \mathcal{E} \vdash \kappa'}{\mathcal{E};\Delta \vdash (\Lambda \chi.\, \tau)\,[\kappa'] \leadsto \tau\{\kappa'/\chi\} \,:\, \kappa\{\kappa'/\chi\}}$$

$$\frac{\mathcal{E};\Delta \vdash \tau \,:\, \kappa \to \kappa' \quad \alpha \notin \mathit{ftv}(\tau)}{\mathcal{E};\Delta \vdash \lambda \alpha{:}\kappa.\, \tau\, \alpha \leadsto \tau \,:\, \kappa \to \kappa'}$$

$$\frac{\mathcal{E};\Delta \vdash \tau \,:\, \forall \chi'.\, \kappa \quad \chi \notin \mathit{fkv}(\tau)}{\mathcal{E};\Delta \vdash \Lambda \chi.\, \tau\,[\chi] \leadsto \tau \,:\, \forall \chi'.\, \kappa}$$

$$\frac{\mathcal{E};\Delta \vdash \mathsf{Typerec}[\kappa]\ \mathsf{int} \text{ of } (\tau_{\mathsf{int}};\, \tau_{\to};\, \tau_{\forall};\, \tau_{\forall^{+}}) \,:\, \kappa}{\mathcal{E};\Delta \vdash \mathsf{Typerec}[\kappa]\ \mathsf{int} \text{ of } (\tau_{\mathsf{int}};\, \tau_{\to};\, \tau_{\forall};\, \tau_{\forall^{+}}) \leadsto \tau_{\mathsf{int}} \,:\, \kappa}$$

$$\frac{\begin{array}{c} \mathcal{E};\Delta \vdash \mathsf{Typerec}[\kappa]\ \tau_1 \text{ of } (\tau_{\mathsf{int}};\, \tau_{\to};\, \tau_{\forall};\, \tau_{\forall^{+}}) \leadsto \tau_1' \,:\, \kappa \\ \mathcal{E};\Delta \vdash \mathsf{Typerec}[\kappa]\ \tau_2 \text{ of } (\tau_{\mathsf{int}};\, \tau_{\to};\, \tau_{\forall};\, \tau_{\forall^{+}}) \leadsto \tau_2' \,:\, \kappa \end{array}}{\mathcal{E};\Delta \vdash \mathsf{Typerec}[\kappa]\ ((\twoheadrightarrow)\, \tau_1\, \tau_2) \text{ of } (\tau_{\mathsf{int}};\, \tau_{\to};\, \tau_{\forall};\, \tau_{\forall^{+}}) \leadsto \tau_{\to}\, \tau_1\, \tau_2\, \tau_1'\, \tau_2' \,:\, \kappa}$$

$$\frac{\mathcal{E};\Delta, \alpha{:}\kappa' \vdash \mathsf{Typerec}[\kappa]\ (\tau\, \alpha) \text{ of } (\tau_{\mathsf{int}};\, \tau_{\to};\, \tau_{\forall};\, \tau_{\forall^{+}}) \leadsto \tau' \,:\, \kappa}{\begin{array}{c} \mathcal{E};\Delta \vdash \mathsf{Typerec}[\kappa]\ (\boldsymbol{\forall}\,[\kappa']\,\tau) \text{ of } (\tau_{\mathsf{int}};\, \tau_{\to};\, \tau_{\forall};\, \tau_{\forall^{+}}) \\ \leadsto \tau_{\forall}\,[\kappa']\,\tau\,(\lambda \alpha{:}\kappa'.\, \tau') \,:\, \kappa \end{array}}$$

$$\frac{\mathcal{E}, \chi;\Delta \vdash \mathsf{Typerec}[\kappa]\ (\tau\,[\chi]) \text{ of } (\tau_{\mathsf{int}};\, \tau_{\to};\, \tau_{\forall};\, \tau_{\forall^{+}}) \leadsto \tau' \,:\, \kappa}{\mathcal{E};\Delta \vdash \mathsf{Typerec}[\kappa]\ (\boldsymbol{\forall}^{+}\tau) \text{ of } (\tau_{\mathsf{int}};\, \tau_{\to};\, \tau_{\forall};\, \tau_{\forall^{+}}) \leadsto \tau_{\forall^{+}}\, \tau\, (\Lambda \chi.\, \tau') \,:\, \kappa}$$

Figure 5: Selected λ_i^P type reduction rules

kind Ω are

$$\begin{array}{rcl} \mathsf{int} & : & \Omega \\ \boldsymbol{\rightarrow} & : & \Omega \rightarrow \Omega \rightarrow \Omega \\ \boldsymbol{\forall} & : & \forall \chi.\, (\chi \rightarrow \Omega) \rightarrow \Omega \\ \boldsymbol{\forall}^{+} & : & (\forall \chi.\, \Omega) \rightarrow \Omega \end{array}$$

None of these constructors' arguments have the kind Ω in a negative position; hence the kind Ω can now be defined inductively in terms of these constructors. The $\mathsf{Typerec}$ construct is then the iterator over this kind. The formation rule for $\mathsf{Typerec}$ follows naturally from the type reduction rules (Figure 5). Depending on the head constructor of the type being analyzed, $\mathsf{Typerec}$ chooses one of the branches. At the int type, it returns the τ_{int} branch. At the function type $\tau \rightarrow \tau'$, it applies the $\tau_{\rightarrow}$ branch to the components τ and τ' and to the result of the iteration over τ and τ'.

When analyzing a polymorphic type, the implied reduction rule is

$$\begin{array}{l} \mathsf{Typerec}[\kappa]\ (\forall \alpha : \kappa'.\, \tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+}) \leadsto \\ \tau_{\forall}\ [\kappa']\ (\lambda \alpha : \kappa'.\, \tau)\ (\lambda \alpha : \kappa'.\, \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})) \end{array}$$

Thus the $\forall$-branch of $\mathsf{Typerec}$ gets as arguments the kind of the bound variable, the abstraction representing the quantified type, and a type function encapsulating the result of the iteration on the body of the quantified type. Since $\tau_{\forall}$ must be parametric in the kind κ' (there are no facilities for kind analysis in the language), it can only apply its second and third arguments to locally introduced type variables of kind κ'. We believe this restriction, which is crucial for preserving strong normalization of the type language, is quite reasonable in practice. For instance $\tau_{\forall}$ can yield a quantified type based on the result of the iteration.

When analyzing a kind polymorphic type, the implied reduction rule is

$$\begin{array}{l} \mathsf{Typerec}[\kappa]\ (\forall^{+} \chi.\, \tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+}) \leadsto \\ \tau_{\forall^+}\ (\Lambda \chi.\, \tau)\ (\Lambda \chi.\, \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})) \end{array}$$

The $\boldsymbol{\forall}^{+}$-branch of $\mathsf{Typerec}$ gets as arguments the body of the quantified type and a kind function encapsulating the result of the iteration on the body of the quantified type.

For ease of presentation, we will use ML-style pattern matching syntax to define a type involving $\mathsf{Typerec}$. Instead of writing

$$\begin{array}{l} \tau = \lambda \alpha : \Omega.\, \mathsf{Typerec}[\kappa]\ \alpha\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\boldsymbol{\forall}};\ \tau_{\forall^+}) \\ \text{where} \\ \tau_{\rightarrow} = \lambda \alpha_1 : \Omega.\, \lambda \alpha_2 : \Omega.\, \lambda \alpha_1' : \kappa.\, \lambda \alpha_2' : \kappa.\, \tau_{\rightarrow}' \\ \tau_{\boldsymbol{\forall}} = \Lambda \chi.\, \lambda \alpha : \chi \rightarrow \Omega.\, \lambda \alpha' : \chi \rightarrow \kappa.\, \tau_{\boldsymbol{\forall}}' \\ \tau_{\forall^+} = \lambda \alpha : (\forall \chi.\, \Omega).\, \lambda \alpha' : (\forall \chi.\, \kappa).\, \tau_{\forall^+}' \end{array}$$

we will write

$$\begin{array}{lcl} \tau\ (\mathsf{int}) & = & \tau_{\mathsf{int}} \\ \tau\ (\alpha_1 \rightarrow \alpha_2) & = & \tau_{\rightarrow}'\{\tau\ (\alpha_1), \tau\ (\alpha_2) / \alpha_1', \alpha_2'\} \\ \tau\ (\boldsymbol{\forall}\ [\chi]\ \alpha_1) & = & \tau_{\boldsymbol{\forall}}'\{\lambda \alpha : \chi.\, \tau\ (\alpha_1\ \alpha) / \alpha'\} \\ \tau\ (\boldsymbol{\forall}^{+} \alpha_1) & = & \tau_{\forall^+}'\{\Lambda \chi.\, \tau\ (\alpha_1\ [\chi]) / \alpha'\} \end{array}$$

To illustrate the type level analysis we will use the $\mathsf{Typerec}$ operator to define the class of types admitting equality comparisons. To prepare for a later section, we extend the language with a product type constructor ($\boldsymbol{\times}$) of the same kind as ($\boldsymbol{\rightarrow}$) and with existential types. The type constructor is $\exists$ with kind identical to that of $\boldsymbol{\forall}$, and $\exists \alpha : \kappa.\, \tau \equiv \exists\ [\kappa]\ (\lambda \alpha : \kappa.\, \tau)$. Correspondingly we extend $\mathsf{Typerec}$ with a product branch $\tau_{\times}$ and an existential branch $\tau_{\exists}$ which behave in exactly the same way as the $\tau_{\rightarrow}$ branch and the $\tau_{\boldsymbol{\forall}}$ branch respectively. Also, we use Bool instead of int as the base type.

A polymorphic function eq comparing two objects for equality is not defined on values of function or polymorphic types. We can enforce this restriction statically if we define a type operator $\mathsf{Eq} : \Omega \rightarrow \Omega$ which maps function and polymorphic types to the type Void, and require the arguments of eq to be of type $\mathsf{Eq}\,\tau$ for any type τ. (Here $\mathsf{Void} \equiv \forall \alpha : \Omega.\, \alpha$ is a type with no values.) Given any type τ, the function Eq serves to verify that a non-equality type does not occur inside τ.

$$\begin{array}{lcl} \mathsf{Eq}\ (\mathsf{Bool}) & = & \mathsf{Bool} \\ \mathsf{Eq}\ (\alpha_1 \rightarrow \alpha_2) & = & \mathsf{Void} \\ \mathsf{Eq}\ (\alpha_1 \times \alpha_2) & = & \mathsf{Eq}\ (\alpha_1) \times \mathsf{Eq}\ (\alpha_2) \\ \mathsf{Eq}\ (\boldsymbol{\forall}\ [\chi]\ \alpha) & = & \mathsf{Void} \\ \mathsf{Eq}\ (\boldsymbol{\forall}^{+} \alpha) & = & \mathsf{Void} \\ \mathsf{Eq}\ (\exists\ [\chi]\ \alpha) & = & \exists\ [\chi]\ (\lambda \alpha_1 : \chi.\, \mathsf{Eq}\ (\alpha\ \alpha_1)) \end{array}$$

Note that $\mathsf{Eq}\ ((\mathsf{Bool} \rightarrow \mathsf{Bool}) \times (\mathsf{Bool} \rightarrow \mathsf{Bool}))$ reduces to $(\mathsf{Void} \times \mathsf{Void})$; a more complicated definition is necessary to map this type to Void.

At the term level type analysis is carried out by the $\mathsf{typecase}$ construct; however, it is not iterative since the term language has a recursion primitive, fix. The $e_{\forall}$ branch of $\mathsf{typecase}$ binds the kind and the type function carried by the $\boldsymbol{\forall}$ type constructor, while the $e_{\forall^+}$ branch binds the kind function carried by the $\boldsymbol{\forall}^{+}$ type constructor.

$$\begin{array}{l} \mathsf{typecase}[\tau]\ (\boldsymbol{\forall}\ [\kappa]\ \tau')\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\rightarrow};\ e_{\forall};\ e_{\forall^+}) \leadsto e_{\forall}\ [\kappa]^{+}\ [\tau'] \\ \mathsf{typecase}[\tau]\ (\boldsymbol{\forall}^{+} \tau')\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\rightarrow};\ e_{\forall};\ e_{\forall^+}) \leadsto e_{\forall^+}\ [\tau'] \end{array}$$

The semantics of the resulting language λ_i^P is presented in Figures 4, 5, and in Figure 6. We occasionally use the conventional notations $\tau \rightarrow \tau'$ for function types and $\forall \alpha : \kappa.\, \tau$ for polymorphic types, as well as $\forall^{+} \chi.\, \tau$ for kind-polymorphic types, instead of their corresponding forms in λ_i^P shown in Figure 3.

The language λ_i^P obeys the following properties. For detailed proofs, see Appendix B.

Theorem 3.1 (Strong Normalisation) *Reduction of well formed types is strongly normalizing.*

We prove strong normalization using Girard's candidates method [6]. We impose the same properties on a candidate. The

$$(\lambda x : \tau.\, e)\ v \leadsto e\{v/x\} \qquad (\mathsf{fix}\ x : \tau.\, v)\ v' \leadsto (v\{\mathsf{fix}\ x : \tau.\, v/x\})\ v'$$

$$(\Lambda \alpha : \kappa.\, v)\ [\tau] \leadsto v\{\tau/\alpha\} \qquad (\mathsf{fix}\ x : \tau.\, v)\ [\tau] \leadsto (v\{\mathsf{fix}\ x : \tau.\, v/x\})\ [\tau]$$

$$(\Lambda^{+} \chi.\, v)\ [\kappa]^{+} \leadsto v\{\kappa/\chi\} \qquad (\mathsf{fix}\ x : \tau.\, v)\ [\kappa]^{+} \leadsto (v\{\mathsf{fix}\ x : \tau.\, v/x\})\ [\kappa]^{+}$$

$$\mathsf{typecase}[\tau]\ \mathsf{int}\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\rightarrow};\ e_{\forall};\ e_{\forall^+}) \leadsto e_{\mathsf{int}}$$

$$\mathsf{typecase}[\tau]\ (\tau_1 \rightarrow \tau_2)\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\rightarrow};\ e_{\forall};\ e_{\forall^+}) \leadsto e_{\rightarrow}\ [\tau_1]\ [\tau_2]$$

$$\mathsf{typecase}[\tau]\ (\boldsymbol{\forall}\ [\kappa]\ \tau)\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\rightarrow};\ e_{\forall};\ e_{\forall^+}) \leadsto e_{\forall}\ [\kappa]^{+}\ [\tau]$$

$$\mathsf{typecase}[\tau]\ (\boldsymbol{\forall}^{+} \tau)\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\rightarrow};\ e_{\forall};\ e_{\forall^+}) \leadsto e_{\forall^+}\ [\tau]$$

$$\frac{e \leadsto e_1}{e\ e' \leadsto e_1\ e'} \qquad \frac{e \leadsto e_1}{v\ e \leadsto v\ e_1} \qquad \frac{e \leadsto e_1}{e\ [\tau] \leadsto e_1\ [\tau]} \qquad \frac{e \leadsto e_1}{e\ [\kappa]^{+} \leadsto e_1\ [\kappa]^{+}}$$

$$\frac{\varepsilon; \varepsilon \vdash \tau' \leadsto^{*} \nu' : \Omega \qquad \nu' \text{ is normal form}}{\begin{array}{l} \mathsf{typecase}[\tau]\ \tau'\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\rightarrow};\ e_{\forall};\ e_{\forall^+}) \leadsto \\ \quad \mathsf{typecase}[\tau]\ \nu'\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\rightarrow};\ e_{\forall};\ e_{\forall^+}) \end{array}}$$

Figure 6: Operational semantics of λ_i^P

set of neutral types now includes a $\mathsf{Typerec}$. We define a new set R_Ω as follows – it is the set of types τ of kind Ω such that $\mathsf{Typerec}[\kappa]\ \tau$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ belongs to a candidate for kind κ; given that the branches belong to candidates of the corresponding kind. We then prove that this set is a candidate. We then define an operation $\mathcal{S}_\kappa$ that, given a kind κ, forms a set of types of kind κ. For the kind Ω, this operation returns R_Ω. For function, polymorphic, and variable kinds the operation acts as in [6]. We prove that $\mathcal{S}_\kappa$ always returns a candidate. We then prove that a type of kind κ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, where the kind variable χ_i maps to the candidate $\mathcal{C}_i$.

Theorem 3.2 (Confluence) *Reduction of well formed types is confluent.*

Confluence of type reduction now follows from local confluence which we prove by case analysis of the type reduction relation ($\rightsquigarrow$). We consider types with two holes and show that the reduction is locally confluent in each case.

Theorem 3.3 (Type Safety) *If* $\vdash e : \tau$ *and* $e \rightsquigarrow^* e'$ *then* e' *is not stuck.*

We say that a term is stuck if it is not a value and no rule of the operational semantics applies. We prove soundness of the system using a contextual semantics in Wright/Felleisen style [26]. It uses the usual progress, subject reduction, and substitution lemmas and also that our type system is strongly normalising and confluent.

3.2 Example: Marshalling

One of the examples that Harper and Morrisett [7] use to illustrate the power of intensional type analysis is based on the extension of ML for distributed computing proposed by Ohori and Kato [14]. The idea is to convert values into a form which can be used for transmission over a network. An integer value may be transmitted directly, but a function may not; instead, a globally unique identifier is transmitted that serves as a proxy at the remote site. These identifiers are associated with their functions by a name server that may be contacted through a primitive addressing scheme. The remote sites use the identifiers to make remote calls to the function. Harper and Morrisett show how to define types of transmissible values as well as functions for marshalling to and unmarshalling from these types using intensional type analysis. However, the predicativity of their calculus prevents them from handling the full calculus of Ohori and Kato, which also includes the remote representation of polymorphic functions and remote type application.

In λ_i^P marshalling of polymorphic values is straightforward; in fact it offers more flexibility than the calculus of Ohori and Kato needs, since polymorphic functions become first-class values, and polymorphic types can be used in remote type applications. Adapting the constructs of [7] to λ_i^P, we introduce a type constructor $\mathsf{Id} : \Omega \rightarrow \Omega$. A value of type τ has a global identifier of type $\mathsf{Id}\ \tau$. The $\mathsf{Typerec}$ and $\mathsf{typecase}$ operators are extended in an obvious way. For example, the following type reduction relation is added:

$$\begin{array}{l}\mathsf{Typerec}[\kappa]\ (\mathsf{Id}\ \tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{Id}) \rightsquigarrow \\ \quad \tau_{Id}\ \tau\ (\mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{Id}))\end{array}$$

The type of the remote representation of values of type τ is $\mathsf{Tran}\ \tau$, defined in [7] using intensional analysis of τ. Values of type $\mathsf{Tran}\ \tau$ do not contain any abstractions; all the abstractions are wrapped inside an Id constructor. We can extend the Harper/Morrisett definition of Tran to handle the quantified types of λ_i^P as follows:

$$\begin{array}{lcl}\mathsf{Tran}\ (\mathsf{int}) & = & \mathsf{int} \\ \mathsf{Tran}\ (\alpha_1 \rightarrow \alpha_2) & = & \mathsf{Id}\ (\mathsf{Tran}\ \alpha_1 \rightarrow \mathsf{Tran}\ \alpha_2) \\ \mathsf{Tran}\ (\forall\ [\chi]\ \alpha) & = & \mathsf{Id}\ (\forall \alpha' : \chi.\ (\lambda \alpha_1 : \chi.\ \mathsf{Tran}\ (\alpha\ \alpha_1))\ \alpha') \\ \mathsf{Tran}\ (\forall^+ \alpha) & = & \mathsf{Id}\ (\forall^+ \chi'.\ (\Lambda \chi.\ \mathsf{Tran}\ (\alpha\ [\chi]))\ [\chi']) \\ \mathsf{Tran}\ (\mathsf{Id}\ \alpha) & = & \mathsf{Id}\ \alpha\end{array}$$

At the term level the system provides primitives for creating global identifiers and performing remote invocations:[1]

$$\begin{array}{rl}\mathsf{newid} : & \forall \alpha_1 : \Omega.\ \forall \alpha_2 : \Omega.\ (\mathsf{Tran}\ \alpha_1 \rightarrow \mathsf{Tran}\ \alpha_2) \rightarrow \mathsf{Tran}\ (\alpha_1 \rightarrow \alpha_2) \\ \mathsf{rapp} : & \forall \alpha_1 : \Omega.\ \forall \alpha_2 : \Omega.\ \mathsf{Tran}\ (\alpha_1 \rightarrow \alpha_2) \rightarrow \mathsf{Tran}\ \alpha_1 \rightarrow \mathsf{Tran}\ \alpha_2 \\ \mathsf{newpid} : & \forall^+ \chi.\ \forall \alpha : \chi \rightarrow \Omega.\ (\forall \alpha' : \chi.\ \mathsf{Tran}\ (\alpha\ \alpha')) \rightarrow \mathsf{Tran}\ (\forall\ [\chi]\ \alpha) \\ \mathsf{rtapp} : & \forall^+ \chi.\ \forall \alpha : \chi \rightarrow \Omega.\ \mathsf{Tran}\ (\forall\ [\chi]\ \alpha) \rightarrow \forall \alpha' : \chi.\ \mathsf{Tran}\ (\alpha\ \alpha')\end{array}$$

For completeness in our system we also need to handle kind polymorphism and remote kind applications:

$$\begin{array}{rl}\mathsf{newkid} : & \forall \alpha : (\forall \chi.\ \Omega).\ (\forall^+ \chi.\ \mathsf{Tran}\ (\alpha\ [\chi])) \rightarrow \mathsf{Tran}\ (\forall^+ \alpha) \\ \mathsf{rkapp} : & \forall \alpha : (\forall \chi.\ \Omega).\ \mathsf{Tran}\ (\forall^+ \alpha) \rightarrow \forall^+ \chi.\ \mathsf{Tran}\ (\alpha\ [\chi])\end{array}$$

Operationally, the newid's take a function between transmissible values and generate a new, globally unique identifier and tell the name server to associate that identifier with the function on the local machine. The remote applications take a proxy identifier of a remote function and a transmissible argument value. The name server is contacted to get the site where the remote value exists; the argument is sent to this machine, and the result of the function transmitted back as the result of the operation.

Marshalling and unmarshalling of values from transmissible representations are performed by the mutually recursive functions $\mathsf{M} : \forall \alpha : \Omega.\ \alpha \rightarrow \mathsf{Tran}\ \alpha$ and $\mathsf{U} : \forall \alpha : \Omega.\ \mathsf{Tran}\ \alpha \rightarrow \alpha$. They are defined below by a pattern-matching syntax and implicit recursion instead of $\mathsf{typecase}$ and fix. We assume that, to be transmitted, a type or a kind does not need to be transformed.

$$\begin{array}{lcl}\mathsf{M}\ [\mathsf{int}] & = & \lambda x : \mathsf{int}.\ x \\ \mathsf{M}\ [\alpha_1 \rightarrow \alpha_2] & = & \lambda x : \alpha_1 \rightarrow \alpha_2. \\ & & \quad \mathsf{newid}\ [\alpha_1]\ [\alpha_2] \\ & & \qquad (\lambda x' : \mathsf{Tran}\ \alpha_1.\ \mathsf{M}\ [\alpha_2]\ (x\ (\mathsf{U}\ [\alpha_1]\ x'))) \\ \mathsf{M}\ [\forall\ [\chi]\ \alpha] & = & \lambda x : \forall\ [\chi]\ \alpha. \\ & & \quad \mathsf{newpid}\ [\chi]^+\ [\alpha]\ (\Lambda \alpha' : \chi.\ \mathsf{M}\ [\alpha\ \alpha']\ (x\ [\alpha'])) \\ \mathsf{M}\ [\forall^+ \alpha] & = & \lambda x : \forall^+ \alpha.\ \mathsf{newkid}\ [\alpha]\ (\Lambda^+ \chi.\ \mathsf{M}\ [\alpha\ [\chi]]\ (x\ [\chi]^+)) \\ \mathsf{M}\ [\mathsf{Id}\ \alpha] & = & \lambda x : \mathsf{Id}\ \alpha.\ x\end{array}$$

$$\begin{array}{rcl}\mathsf{U}\ [\mathsf{int}] & = & \lambda x : \mathsf{Tran}\ (\mathsf{int}).\ x \\ \mathsf{U}\ [\alpha_1 \rightarrow \alpha_2] & = & \lambda x : \mathsf{Tran}\ (\alpha_1 \rightarrow \alpha_2).\ \lambda x' : \alpha_1. \\ & & \quad \mathsf{U}\ [\alpha_2]\ (\mathsf{rapp}\ [\alpha_1]\ [\alpha_2]\ x\ (\mathsf{M}\ [\alpha_1]\ x')) \\ \mathsf{U}\ [\forall\ [\chi]\ \alpha] & = & \lambda x : \mathsf{Tran}\ (\forall\ [\chi]\ \alpha).\ \Lambda \alpha' : \chi. \\ & & \quad \mathsf{U}\ [\alpha\ \alpha']\ (\mathsf{rtapp}\ [\chi]^+\ [\alpha]\ x\ [\alpha']) \\ \mathsf{U}\ [\forall^+ \alpha] & = & \lambda x : \mathsf{Tran}\ (\forall^+ \alpha).\ \Lambda^+ \chi.\ \mathsf{U}\ [\alpha\ [\chi]]\ (\mathsf{rkapp}\ [\alpha]\ x\ [\chi]^+) \\ \mathsf{U}\ [\mathsf{Id}\ \alpha] & = & \lambda x : \mathsf{Tran}\ (\mathsf{Id}\ \alpha).\ x\end{array}$$

3.3 Example: Polymorphic equality

A more interesting example involves the comparison of values of existential type. The term constructs for introducing and eliminat-

[1] Ohori and Kato [14] define one primitive for creating identifiers for both term and type abstraction.

```
letrec
  heq : ∀α₁:Ω. ∀α₂:Ω. Eq α₁ → Eq α₂ → Bool
  = Λα₁:Ω. Λα₂:Ω.
    typecase[λγ:Ω. Eq γ → Eq α₂ → Bool] α₁ of
      Bool     ⇒ λx:Bool.
                    typecase[λγ:Ω. Eq γ → Bool] α₂ of
                      Bool     ⇒ λy:Bool. primEqBool x y
                      ...      ⇒ ... false
      β₁ × β₂  ⇒ λx:Eq β₁ × Eq β₂.
                    typecase[λγ:Ω. Eq γ → Bool] α₂ of
                      β₁' × β₂'  ⇒ λy:Eq β₁' × Eq β₂'.
                                     heq [β₁] [β₁'] (x.1) (y.1) and
                                     heq [β₂] [β₂'] (x.2) (y.2)
                      ...      ⇒ ... false
      ∃ [χ] β  ⇒ λx:∃β₁:χ. Eq (β β₁).
                    typecase[λγ:Ω. Eq γ → Bool] α₂ of
                      ∃ [χ'] β' ⇒ λy:∃β₁':χ'. β' β₁'.
                                   open x as ⟨β₁:χ, xc:Eq (β β₁)⟩ in
                                   open y as ⟨β₁':χ', yc:Eq (β' β₁')⟩ in
                                     heq [β β₁] [β' β₁'] xc yc
                      ...      ⇒ ... false
      ...
in let eq = Λα:Ω. λx:Eq α. λy:Eq α. heq [α] [α] x y
in ...
```

Figure 7: Polymorphic equality in λ_i^P

ing existential types have the following typing rules.

$$\frac{\mathcal{E};\Delta;\Gamma \vdash e \ :\ (\lambda\alpha{:}\kappa.\,\tau)\,\tau'}{\mathcal{E};\Delta;\Gamma \vdash \langle\alpha{:}\kappa = \tau',\, e{:}\tau\rangle \ :\ \exists\alpha{:}\kappa.\,\tau}$$

$$\frac{\mathcal{E};\Delta;\Gamma \vdash e \ :\ \exists\,[\kappa]\,\tau \qquad \mathcal{E};\Delta \vdash \tau' \ :\ \Omega \qquad \mathcal{E};\Delta,\alpha{:}\kappa;\Gamma,x{:}\tau\,\alpha \vdash e' \ :\ \tau'}{\mathcal{E};\Delta;\Gamma \vdash \mathsf{open}\ e\ \mathsf{as}\ \langle\alpha{:}\kappa,\ x{:}\tau\,\alpha\rangle\ \mathsf{in}\ e' \ :\ \tau'}$$

The polymorphic equality function eq is defined in Figure 7 (we use a letrec construct derived from our fix). The domain type of the function is restricted to types of the form $\mathsf{Eq}\,\tau$ which ensures that only values of types admitting equality are compared.

Consider the two packages $v = \langle\alpha:\Omega = \mathsf{Bool},\ \mathsf{false}:\alpha\rangle$ and $v' = \langle\alpha:\Omega = \mathsf{Bool}\times\mathsf{Bool},\ \langle\mathsf{true},\ \mathsf{true}\rangle:\alpha\rangle$. Both are of type $\exists\alpha{:}\Omega.\,\alpha$, which makes the invocation $\mathsf{eq}\ [\exists\alpha{:}\Omega.\,\alpha]\ v\ v'$ legal. But when the packages are open, the types of the packaged values may (as in this example) turn out to be different.

Therefore we need the auxiliary function heq to compare values of possibly different types by comparing their types first. It represents a matrix on the types of the two arguments, where the diagonal elements compare recursively the constituent values. Off-diagonal elements return false and are abbreviated in the figure.

The only interesting case is that of values of an existential type. Opening the packages provides access to the witness types β_1 and β_1' of the arguments x and y. As shown in the typing rules, the actual types of the packaged values, x and y, are obtained by applying the corresponding type functions β and β' to the respective witness types. Note that when the arguments are of existential types, the semantics of type comparison may be somewhat unexpected. Consider the call to the eq function shown below which evaluates to true.

$$\begin{array}{l} \mathsf{eq}\ [\exists\alpha{:}\Omega.\,\alpha] \\ \langle\alpha{:}\Omega = \exists\beta{:}\Omega.\,\beta,\ \langle\beta{:}\Omega = \mathsf{Bool},\ \mathsf{true}{:}\mathsf{Eq}\,\beta\rangle : \mathsf{Eq}\,\alpha\rangle \\ \langle\alpha{:}\Omega = \exists\beta{:}\Omega\to\Omega.\,\beta\,\mathsf{Bool}, \\ \qquad \langle\beta{:}\Omega\to\Omega = \lambda\gamma{:}\Omega.\,\gamma,\ \mathsf{true}{:}\mathsf{Eq}\,(\beta\,\mathsf{Bool})\rangle : \mathsf{Eq}\,\alpha\rangle \end{array}$$

At runtime, after the packages are opened, the call to heq is

$$\begin{array}{l} \mathsf{heq}\ [\exists\beta{:}\Omega.\,\beta]\ [\exists\beta{:}\Omega\to\Omega.\,\beta\,\mathsf{Bool}] \\ \langle\beta{:}\Omega = \mathsf{Bool},\ \mathsf{true}{:}\mathsf{Eq}\,\beta\rangle \\ \langle\beta{:}\Omega\to\Omega = \lambda\gamma{:}\Omega.\,\gamma,\ \mathsf{true}{:}\mathsf{Eq}\,(\beta\,\mathsf{Bool})\rangle \end{array}$$

This term evaluates to true even though the types are different. This is because the types compared are the actual types of the values before their packing. This is seen at the recursive call to heq where the original types are used.

$$\mathsf{heq}\ [\mathsf{Bool}]\ [(\lambda\gamma{:}\Omega.\,\gamma)\,\mathsf{Bool}]\ \mathsf{true}\,\mathsf{true}$$

On the other hand, the two packages of type $\exists\alpha:\Omega.\,\alpha$ above will indeed behave identically in all contexts. A informal argument in support of this claim is that the most any context could do with such a package is open it and inspect the type of the value using a typecase, but this will only provide access to a *type function* τ representing the internal existential type. Since the kind of the argument of τ is unknown statically, the only non-trivial operation on τ is its application to the witness type of the package. As we saw above, this operation will produce the same result (namely Bool) for both values. Thus, since the two arguments to eq were indistinguishable in all λ_i^P contexts, the above result is perfectly sensible.

3.4 A discussion of λ_i^P

Before we move on, it would be worthwhile to analyze the λ_i^P language. Specifically, what is the price in terms of the complexity of the type theory that can be attributed to the requirements that we imposed?

We saw in Section 2.3 that an iterative type operator is essential to type the transformations of most type directed operations. Even if we were to consider compiling just ML, we would still have to consider polymorphic types of the form $\forall\alpha:\Omega.\,\tau$; the polymorphic type is still non-inductive. Therefore, even for this simple case, we need kind polymorphism in an essential way to handle the negative occurrence. In turn, kind polymorphism allows us to analyze arbitrarily quantified types at the type level; hence the extra expressivity comes for free. Moreover, adding kind polymorphism does not entail any heavy type-theoretic machinery. The kind and type language of λ_i^P is a minor extension (with primitive recursion) of the F_2 calculus which has been widely studied; we use the basic techniques developed for F_2 [6] to prove properties of our type language.

For analysis of quantified types at the term level we have the new construct $\Lambda^{+}\chi.\,e$. First, this does not result in any additional complexity at the type level. Although we introduce a new type constructor $\forall^{+}$, the kind of this construct is defined completely by the original kind calculus; the kind and type calculus is still essentially F_2. We could restrict the type analysis at the term level to a finite set of kinds and avoid introducing the new term. However, even in this case, we would still need kind abstraction to implement a type erasure semantics (Section 5) which can simplify certain phases of the compiler. On the other hand, kind abstraction at the term level in λ_i^P does not in any way complicate the transition to type erasure semantics.

4 Analyzing recursive types

Next we turn our attention to the problem of analyzing recursive types. Following the general scheme described in the previous section, we need to introduce a type constructor μ, creating a type which is isomorphic to the least fixpoint of a given type function. Since the types we analyze are of kind Ω, the kind of μ of interest is

$$\mu : (\Omega \to \Omega) \to \Omega$$

Unfortunately there is a negative occurrence of Ω in the domain of this kind, which—as it was with universally-quantified types in Section 3—prevents defining an iterator over this kind while maintaining strong normalization of the type language. In the case of quantified types we were able to resolve this problem by generalizing the negative occurrence of Ω to an arbitrary kind; however such an approach is doomed in the case of recursive types since the type function must have identical argument and result kinds.

One possibility is to follow the approach outlined by Crary and Weirich in [1] for quantified types; since type variables bound by the fixpoint operator must be of kind Ω, an environment can be used to map them to types of kind Ω without kind mismatches. While plausible and perhaps efficient, this approach (as pointed out in Section 2.4) gives no protection against some programming errors.

4.1 A restricted Typerec

To handle recursive types, we introduce a new constructor Place that acts as the right inverse of the Typerec. We will first give an informal explanation of how the Place constructor is used in our solution by considering a restricted form of the Typerec. This approach does not guarantee termination; we use it to ease the presentation of the λ_i^Q calculus.

Consider the iteration $\mathsf{Typerec}[\Omega]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu)$ in the case when τ is a recursive type, say $\mu\,(\lambda\alpha:\Omega.\ \mathsf{int} \to \alpha)$. In many cases, the desired result will be another recursive type, say $\mu\,(\lambda\alpha:\Omega.\,\tau')$ where τ' is the result of analysing the body. If we followed the approach we used in the case of polymorphic types (i.e. if the iterator's action on the type variable is suspended until the variable is replaced by a type upon unfolding the fixpoint), then the result would be:

$$\mu\,(\lambda\alpha:\Omega.\ \tau_{\to}\ \mathsf{int}\ \alpha\ \tau_{\mathsf{int}}\ (\mathsf{Typerec}[\Omega]\ \alpha\ \mathsf{of}\ \ldots))$$

In this case, the iterator ends up being applied n times to the nth unfolding of the fixpoint, which does not correspond to the desired fixpoint. Instead the iterator must be applied to the body of the type function, but—in contrast with the behavior in the case of a quantified type—the iterator should *disappear* when applied to the type variable α. The fixpoint notation represents a type isomorphic to an infinite unfolding of the body, thus the traversal of the entire infinite tree is complete with one iteration over the body. In other words the iterator must satisfy an equation of the form $\mathsf{Typerec}[\Omega]\ \alpha\ \mathsf{of}\ \ldots = \alpha$ so that the result of analysing the body is $\lambda\alpha:\Omega.\ \tau_{\to}\ \mathsf{int}\ \alpha\ \tau_{\mathsf{int}}\ \alpha$.

Therefore, we need to distinguish between type variables bound by a polymorphic or existential quantifier and those bound in a recursive type. This reasoning leads us to a solution based on the work of Fegaras and Sheard on catamorphisms over non-inductive datatypes [4]. The main idea is to introduce an auxiliary type constructor Place of kind $\Omega \to \Omega$ which is the right inverse of the iterator, i.e. it holds that

$$\mathsf{Typerec}[\Omega]\ (\mathsf{Place}\,\tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu) \leadsto \tau$$

$$
\begin{array}{lll}
(\mathit{kinds}) & \kappa ::= & \chi \mid \natural\kappa \mid \kappa \to \kappa' \mid \forall\chi.\,\kappa \\[1ex]
(\mathit{types}) & \tau ::= & \alpha \mid \mathsf{int} \mid \mathring{\twoheadrightarrow} \mid \mathring{\forall} \mid \mathring{\forall}^{+} \mid \mathring{\mu} \mid \mathsf{Place} \\
 & & \mid\ \lambda\alpha:\kappa.\,\tau \mid \tau\,\tau' \mid \Lambda\chi.\,\tau \mid \tau\,[\kappa] \\
 & & \mid\ \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu) \\[1ex]
(\mathit{values}) & v ::= & i \mid \Lambda^{+}\chi.\,v \mid \Lambda\alpha:\kappa.\,v \mid \lambda x:\tau.\,e \mid \mathsf{fix}\,x:\tau.\,v \\
 & & \mid\ \mathsf{fold}\ v\ \mathsf{as}\ \tau \\[1ex]
(\mathit{terms}) & e ::= & v \mid x \mid e\,[\kappa]^{+} \mid e\,[\tau] \mid e\,e' \\
 & & \mid\ \mathsf{fold}\ e\ \mathsf{as}\ \tau \mid \mathsf{unfold}\ e\ \mathsf{as}\ \tau \\
 & & \mid\ \mathsf{typecase}[\tau]\ \tau'\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\to};\ e_{\forall};\ e_{\forall^+};\ e_\mu)
\end{array}
$$

Figure 8: The λ_i^Q language

$$
\begin{array}{rcl}
\Omega & \equiv & \forall\chi.\,\natural\chi \\
\tau\$\tau' & \equiv & \Lambda\chi.\,\tau\,[\chi]\,(\tau'\,[\chi])\ \text{for}\ \chi \notin \mathit{fkv}(\tau) \cup \mathit{fkv}(\tau') \\
\tau \to \tau' & \equiv & (\twoheadrightarrow)\,\tau\,\tau' \\
\forall\alpha:\kappa.\,\tau & \equiv & \forall\,[\kappa]\,(\lambda\alpha:\kappa.\,\tau) \\
\forall^{+}\chi.\,\tau & \equiv & \forall^{+}(\Lambda\chi.\,\tau)
\end{array}
$$

$$
\begin{array}{lll}
(\twoheadrightarrow) : \Omega \to \Omega \to \Omega & = & \lambda\alpha:\Omega.\,\lambda\alpha':\Omega.\,((\mathring{\twoheadrightarrow})\$\alpha)\$\alpha' \\
\forall\ : \forall\chi.\,(\chi \to \Omega) \to \Omega & = & \Lambda\chi.\,\lambda\alpha:\chi \to \Omega.\,\Lambda\chi'. \\
 & & \quad \mathring{\forall}\,[\chi']\,[\chi]\,(\lambda\alpha':\chi.\,\alpha\,\alpha'\,[\chi']) \\
\forall^{+}\ : (\forall\chi.\,\Omega) \to \Omega & = & \lambda\alpha:\forall\chi.\,\Omega.\,\Lambda\chi'.\,\mathring{\forall}^{+}[\chi']\,(\Lambda\chi.\,\alpha\,[\chi]\,[\chi']) \\
\mu\ : (\forall\chi.\,\natural\chi \to \natural\chi) \to \Omega & = & \lambda\alpha:\forall\chi.\,\natural\chi \to \natural\chi.\,\mathring{\mu}\$\alpha
\end{array}
$$

Figure 9: Syntactic sugar for λ_i^Q

The iterator processes the body of a recursive type with the μ-bound type variable protected under Place. While processing the body, the iterator eventually reduces to instances of the form $(\mathsf{Typerec}[\Omega]\ (\mathsf{Place}\,\alpha)\ \mathsf{of}\ \ldots)$, which reduce to α. The reduction rule for the iterator over a recursive type is

$$
\begin{array}{l}
\mathsf{Typerec}[\Omega]\ (\mu\,\tau')\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu) \leadsto \\
\tau_\mu\ \tau' \\
\quad (\lambda\alpha:\Omega.\ \mathsf{Typerec}[\Omega]\ (\tau'\,(\mathsf{Place}\,\alpha))\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu))
\end{array}
$$

4.2 The general case

The previous approach does not generalise to the case when the result of the Typerec may be of an arbitrary kind. In the general case, the type reductions are:

$$
\begin{array}{l}
\mathsf{Typerec}[\kappa]\ (\mathsf{Place}\,\tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu) \leadsto \tau \\
\mathsf{Typerec}[\kappa]\ (\mu\,\tau')\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu) \leadsto \\
\tau_\mu\ \tau' \\
\quad (\lambda\alpha:\kappa.\ \mathsf{Typerec}[\kappa]\ (\tau'\,(\mathsf{Place}\,\alpha))\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu))
\end{array}
$$

The constructor Place can now be applied to a type of arbitrary kind, but its return result must be Ω. This implies that Place has the kind $\forall\chi.\,\chi \to \Omega$. But this is unsound since we can not constrain the kind of τ above (the argument of Place) to match the result kind κ of the Typerec.

Adopting the solution given by Fegaras/Sheard, we modify the domain of intensional analysis: in place of Ω we introduce a parameterized kind $\natural$, and require that the kind of τ in $\mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu)$ is $\natural\kappa$. The constructor Place must then have the polymorphic kind $\forall\chi.\,\chi \to \natural\chi$, and the fix-point constructor $\mathring{\mu}$ the kind $\forall\chi.\,(\natural\chi \to \natural\chi) \to \natural\chi$.

$\boxed{\text{Kind formation} \quad \mathcal{E} \vdash \kappa}$

$$\frac{\chi \in \mathcal{E}}{\mathcal{E} \vdash \chi} \qquad \frac{\mathcal{E} \vdash \kappa}{\mathcal{E} \vdash \natural\kappa} \qquad \frac{\mathcal{E} \vdash \kappa_1 \quad \mathcal{E} \vdash \kappa_2}{\mathcal{E} \vdash \kappa_1 \to \kappa_2} \qquad \frac{\mathcal{E}, \chi \vdash \kappa}{\mathcal{E} \vdash \forall\chi.\,\kappa}$$

$\boxed{\text{Type environment formation} \quad \mathcal{E} \vdash \Delta}$

$$\frac{}{\mathcal{E} \vdash \varepsilon} \qquad \frac{\mathcal{E} \vdash \Delta \quad \mathcal{E} \vdash \kappa}{\mathcal{E} \vdash \Delta, \alpha{:}\kappa}$$

$\boxed{\text{Type formation} \quad \mathcal{E}; \Delta \vdash \tau : \kappa}$

$$\frac{\mathcal{E} \vdash \Delta \quad \alpha{:}\kappa \text{ in } \Delta}{\mathcal{E}; \Delta \vdash \alpha : \kappa} \qquad \frac{\mathcal{E} \vdash \Delta}{\begin{array}{l} \mathcal{E}; \Delta \vdash \mathsf{int} : \forall\chi.\,\natural\chi \\ \mathcal{E}; \Delta \vdash (\stackrel{\circ}{\twoheadrightarrow}) : \forall\chi.\,\natural\chi \to \natural\chi \to \natural\chi \\ \mathcal{E}; \Delta \vdash \mathring{\forall} : \forall\chi.\,\forall\chi'.\,(\chi' \to \natural\chi) \to \natural\chi \\ \mathcal{E}; \Delta \vdash \mathring{\forall}^{+} : \forall\chi.\,(\forall\chi'.\,\natural\chi) \to \natural\chi \\ \mathcal{E}; \Delta \vdash \mathring{\mu} : \forall\chi.\,(\natural\chi \to \natural\chi) \to \natural\chi \\ \mathcal{E}; \Delta \vdash \mathsf{Place} : \forall\chi.\,\chi \to \natural\chi \end{array}}$$

$$\frac{\mathcal{E}; \Delta, \alpha{:}\kappa \vdash \tau : \kappa'}{\mathcal{E}; \Delta \vdash \lambda\alpha{:}\kappa.\,\tau : \kappa \to \kappa'} \qquad \frac{\mathcal{E}; \Delta \vdash \tau : \kappa' \to \kappa \quad \mathcal{E}; \Delta \vdash \tau' : \kappa'}{\mathcal{E}; \Delta \vdash \tau\,\tau' : \kappa}$$

$$\frac{\mathcal{E}, \chi; \Delta \vdash \tau : \kappa}{\mathcal{E}; \Delta \vdash \Lambda\chi.\,\tau : \forall\chi.\,\kappa} \qquad \frac{\mathcal{E}; \Delta \vdash \tau : \forall\chi.\,\kappa \quad \mathcal{E} \vdash \kappa'}{\mathcal{E}; \Delta \vdash \tau\,[\kappa'] : \kappa\{\kappa'/\chi\}}$$

$$\frac{\begin{array}{l} \mathcal{E}; \Delta \vdash \tau : \natural\kappa \quad \mathcal{E}; \Delta \vdash \tau_{\mathsf{int}} : \kappa \\ \mathcal{E}; \Delta \vdash \tau_{\to} : \natural\kappa \to \natural\kappa \to \kappa \to \kappa \to \kappa \\ \mathcal{E}; \Delta \vdash \tau_{\forall} : \forall\chi.\,(\chi \to \natural\kappa) \to (\chi \to \kappa) \to \kappa \\ \mathcal{E}; \Delta \vdash \tau_{\forall^+} : (\forall\chi.\,\natural\kappa) \to (\forall\chi.\,\kappa) \to \kappa \\ \mathcal{E}; \Delta \vdash \tau_{\mu} : (\natural\kappa \to \natural\kappa) \to (\kappa \to \kappa) \to \kappa \end{array}}{\mathcal{E}; \Delta \vdash \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu}) : \kappa}$$

Figure 10: λ_i^Q type formation rules

We define the λ_i^Q calculus in Figures 8 and 9. Figures 10, 11, and 12 show the static semantics. Figure 13 shows the dynamic semantics.

Types which had kind Ω in λ_i^P could be analyzed by a Typerec with an arbitrary result kind κ'. In our new language λ_i^Q, a type that can be analyzed by an arbitrary Typerec construct must have the kind $\natural\kappa$ for all possible κ. Thus the kind Ω of λ_i^P is represented by the kind $\forall\chi.\,\natural\chi$ in λ_i^Q.

To be able to analyze function and polymorphic types, we now have to modify their kinds as well; to avoid confusion with the constructors based on Ω, we denote the new constructors by $\stackrel{\circ}{\twoheadrightarrow}$, $\mathring{\forall}$, and $\mathring{\forall}^{+}$ (Figure 8). The kind rules for these constructors are shown in Figure 10. We can define equivalents of the λ_i^P types $(\twoheadrightarrow)$, $\forall$, and $\forall^{+}$ starting from $\stackrel{\circ}{\twoheadrightarrow}$, $\mathring{\forall}$, and $\mathring{\forall}^{+}$ respectively. The key intuition in the definition (Figure 9) is that we thread the same kind through all components of kind Ω. For example, expanding the definition of $\tau \to \tau'$ we obtain its equivalent, $\Lambda\chi.\,\stackrel{\circ}{\twoheadrightarrow}[\chi]\,(\tau\,[\chi])(\tau'\,[\chi])$. Expressed in terms of these derived types, the typing rules for most λ_i^Q terms (Figure 11) are identical to those of λ_i^P. Compared with λ_i^P, the term language of λ_i^Q has two new constructs – $\mathsf{fold}\ e\ \mathsf{as}\ \tau$ and $\mathsf{unfold}\ e\ \mathsf{as}\ \tau$ – to implement the isomorphism between a recursive type and its unfolding.

Each of these constructors must first be applied to a kind κ before being analyzed; moreover, the result of the analysis is constrained to be of kind κ. The rest of the analysis is similar to the type analysis in λ_i^P. We iterate over the components of the type and then pass the results of the iteration, and the original components to the corresponding branch of the iterator. For example, consider the analysis of the int and $\mathring{\forall}$ constructors (Figure 12) :

$$\begin{array}{l} \mathsf{Typerec}[\kappa]\ (\mathsf{int}\,[\kappa])\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu}) \leadsto \tau_{\mathsf{int}} \\ \mathsf{Typerec}[\kappa]\ (\mathring{\forall}\,[\kappa]\,[\kappa']\,\tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu}) \leadsto \\ \tau_{\forall}\,[\kappa']\,\tau\,(\lambda\alpha{:}\kappa'.\,\mathsf{Typerec}[\kappa]\ (\tau\,\alpha)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})) \end{array}$$

The reduction rule for the term level typecase is the same as in λ_i^P with the recursive type handled in an obvious way (Figure 13). However, there is one subtlety in the typecase reduction rules. Since typecase does not iterate over the structure of a type, its reductions do not introduce the Place contructor; thus the type analyzed by $\mathsf{Typerec}[\kappa]$ must be of kind $\natural\kappa$, but a typecase can only analyze types of kind $\forall\chi.\,\natural\chi$. It is easy to see that there are no closed types of this kind constructed using Place. Thus there are no reduction rules for typecase analyzing the Place constructor. We prove this (in Section C.1) in proving the soundness of λ_i^Q.

The language λ_i^Q obeys the following properties. For detailed proofs, see Appendix C.

Theorem 4.1 (Strong Normalisation) *Reduction of well formed types is strongly normalizing.*

We prove strong normalization using Girard's candidates method [6]. We impose the same properties on a candidate. The set of neutral types now includes a Typerec. Given a candidate $\mathcal{C}_\kappa$ of kind κ, we define a new set $\natural_R\mathcal{C}_\kappa$ as follows – it is the set of types τ of kind $\natural\kappa$ such that $\mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})$ belongs to a candidate for kind κ; given that the branches belong to candidates of the corresponding kind. We then prove that this set is a candidate. We then define an operation $\mathcal{S}_\kappa$ that, given a kind κ, forms a set of types of kind κ. For the kind $\natural\kappa'$, this operation returns $\natural_R\mathcal{S}_{\kappa'}$. For function, polymorphic, and variable kinds the operation acts as in [6]. We prove that $\mathcal{S}_\kappa$ always returns a candidate. We then prove that a type of kind κ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, where the kind variable χ_i maps to the candidate $\mathcal{C}_i$.

Theorem 4.2 (Confluence) *Reduction of well formed types is confluent.*

Confluence of type reduction now follows from local confluence which we prove by case analysis of the type reduction relation ($\leadsto$). We consider types with two holes and show that the reduction is locally confluent in each case.

Theorem 4.3 (Type Safety) *If* $\vdash e : \tau$ *and* $e \leadsto^{*} e'$ *then* e' *is not stuck.*

We say that a term is stuck if it is not a value and no rule of the operational semantics applies. We prove soundness of the system using a contextual semantics in Wright/Felleisen style [26]. It uses the usual progress, subject reduction, and substitution lemmas and also that our type system is strongly normalising and confluent.

5 Type-erasure semantics

We give a type erasure semantics for our calculi following Crary et al. [2]. This embedding simplifies certain stages of the compiler, most notably typed closure conversion. The basic idea is to construct term level representations of types and pass these representations at runtime. The term level type analysis operator is modified to analyse these representations.

$\boxed{\text{Type reduction} \qquad \mathcal{E}; \Delta \vdash \tau_1 \rightsquigarrow \tau_2 \,:\, \kappa}$

$$\frac{\mathcal{E}; \Delta \vdash \tau \,:\, \kappa}{\mathcal{E}; \Delta \vdash \tau \rightsquigarrow \tau \,:\, \kappa} \qquad \frac{\mathcal{E}; \Delta \vdash \tau_1 \rightsquigarrow \tau_2 \,:\, \kappa}{\mathcal{E}; \Delta \vdash \tau_2 \rightsquigarrow \tau_1 \,:\, \kappa}$$

$$\frac{\mathcal{E}; \Delta \vdash \tau_1 \rightsquigarrow \tau_2 \,:\, \kappa \qquad \mathcal{E}; \Delta \vdash \tau_2 \rightsquigarrow \tau_3 \,:\, \kappa}{\mathcal{E}; \Delta \vdash \tau_1 \rightsquigarrow \tau_3 \,:\, \kappa}$$

$$\frac{\mathcal{E}, \chi; \Delta \vdash \tau \rightsquigarrow \tau' \,:\, \kappa}{\mathcal{E}; \Delta \vdash \Lambda\chi.\, \tau \rightsquigarrow \Lambda\chi.\, \tau' \,:\, \forall\chi.\, \kappa}$$

$$\frac{\mathcal{E}; \Delta \vdash \tau_1 \rightsquigarrow \tau_2 \,:\, \forall\chi.\, \kappa \qquad \mathcal{E} \vdash \kappa'}{\mathcal{E}; \Delta \vdash \tau_1\, [\kappa'] \rightsquigarrow \tau_2\, [\kappa'] \,:\, \kappa\{\kappa'/\chi\}}$$

$$\frac{\mathcal{E}; \Delta, \alpha\!:\!\kappa' \vdash \tau \,:\, \kappa \qquad \mathcal{E}; \Delta \vdash \tau' \,:\, \kappa'}{\mathcal{E}; \Delta \vdash (\lambda\alpha\!:\!\kappa'.\, \tau)\, \tau' \rightsquigarrow \tau\{\tau'/\alpha\} \,:\, \kappa}$$

$$\frac{\mathcal{E}, \chi; \Delta \vdash \tau \,:\, \forall\chi.\, \kappa \qquad \mathcal{E} \vdash \kappa'}{\mathcal{E}; \Delta \vdash (\Lambda\chi.\, \tau)\, [\kappa'] \rightsquigarrow \tau\{\kappa'/\chi\} \,:\, \kappa\{\kappa'/\chi\}}$$

$$\frac{\mathcal{E}; \Delta \vdash \tau \,:\, \kappa \to \kappa' \qquad \alpha \notin \mathit{ftv}(\tau)}{\mathcal{E}; \Delta \vdash \lambda\alpha\!:\!\kappa.\, \tau\, \alpha \rightsquigarrow \tau \,:\, \kappa \to \kappa'}$$

$$\frac{\mathcal{E}; \Delta \vdash \tau \,:\, \forall\chi'.\, \kappa \qquad \chi \notin \mathit{fkv}(\tau)}{\mathcal{E}; \Delta \vdash \Lambda\chi.\, \tau\, [\chi] \rightsquigarrow \tau \,:\, \forall\chi'.\, \kappa}$$

$$\frac{\mathcal{E}; \Delta, \alpha\!:\!\kappa \vdash \tau_1 \rightsquigarrow \tau_2 \,:\, \kappa'}{\mathcal{E}; \Delta \vdash \lambda\alpha\!:\!\kappa.\, \tau_1 \rightsquigarrow \lambda\alpha\!:\!\kappa.\, \tau_2 \,:\, \kappa \to \kappa'}$$

$$\frac{\mathcal{E}; \Delta \vdash \tau_1 \rightsquigarrow \tau_2 \,:\, \kappa' \to \kappa \qquad \mathcal{E}; \Delta \vdash \tau_1' \rightsquigarrow \tau_2' \,:\, \kappa'}{\mathcal{E}; \Delta \vdash \tau_1\, \tau_1' \rightsquigarrow \tau_2\, \tau_2' \,:\, \kappa}$$

$$\frac{\mathcal{E}; \Delta \vdash \mathsf{Typerec}[\kappa]\ (\mathsf{int}\,[\kappa])\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu) \,:\, \kappa}{\mathcal{E}; \Delta \vdash \mathsf{Typerec}[\kappa]\ (\mathsf{int}\,[\kappa])\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu) \rightsquigarrow \tau_{\mathsf{int}} \,:\, \kappa}$$

$$\frac{\begin{array}{c}\mathcal{E}; \Delta \vdash \mathsf{Typerec}[\kappa]\ \tau_1\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu) \rightsquigarrow \tau_1' \,:\, \kappa \\ \mathcal{E}; \Delta \vdash \mathsf{Typerec}[\kappa]\ \tau_2\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu) \rightsquigarrow \tau_2' \,:\, \kappa\end{array}}{\mathcal{E}; \Delta \vdash \mathsf{Typerec}[\kappa]\ ((\overset{\circ}{\to})\, [\kappa]\, \tau_1\, \tau_2)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu) \rightsquigarrow \tau_{\to}\, \tau_1\, \tau_2\, \tau_1'\, \tau_2' \,:\, \kappa}$$

$$\frac{\mathcal{E}; \Delta, \alpha\!:\!\kappa' \vdash \mathsf{Typerec}[\kappa]\ (\tau\, \alpha)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu) \rightsquigarrow \tau' \,:\, \kappa}{\mathcal{E}; \Delta \vdash \mathsf{Typerec}[\kappa]\ (\mathring{\forall}\, [\kappa]\, [\kappa']\, \tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu) \rightsquigarrow \tau_{\forall}\, [\kappa']\, \tau\, (\lambda\alpha\!:\!\kappa'.\, \tau') \,:\, \kappa}$$

$$\frac{\mathcal{E}, \chi; \Delta \vdash \mathsf{Typerec}[\kappa]\ (\tau\, [\chi])\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu) \rightsquigarrow \tau' \,:\, \kappa}{\mathcal{E}; \Delta \vdash \mathsf{Typerec}[\kappa]\ (\mathring{\forall}^+ [\kappa]\, \tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu) \rightsquigarrow \tau_{\forall^+}\, \tau\, (\Lambda\chi.\, \tau') \,:\, \kappa}$$

$$\frac{\mathcal{E}; \Delta, \alpha\!:\!\kappa \vdash \mathsf{Typerec}[\kappa]\ (\tau\, (\mathsf{Place}\, [\kappa]\, \alpha))\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu) \rightsquigarrow \tau' \,:\, \kappa}{\mathcal{E}; \Delta \vdash \mathsf{Typerec}[\kappa]\ (\mathring{\mu}\, [\kappa]\, \tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu) \rightsquigarrow \tau_\mu\, \tau\, (\lambda\alpha\!:\!\kappa.\, \tau') \,:\, \kappa}$$

$$\frac{\mathcal{E}; \Delta \vdash \mathsf{Typerec}[\kappa]\ (\mathsf{Place}\, [\kappa]\, \tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu) \,:\, \kappa}{\mathcal{E}; \Delta \vdash \mathsf{Typerec}[\kappa]\ (\mathsf{Place}\, [\kappa]\, \tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu) \rightsquigarrow \tau \,:\, \kappa}$$

Figure 12: Selected λ_i^Q type reduction rules

5.1 Type-erasure for λ_i^P

Since only types of kind Ω are analysed, we provide representation constants for types of kind Ω; the representations for other kinds will be constructed inductively. Thus the λ_R^P language (Figure 15) has the constant $\mathsf{R_{int}}$ corresponding to the type int, and representation constants like $\mathsf{R}_\rightarrow$ corresponding to each λ_i^P type constructor.

Consider the problem of typing these representations. We introduce the type constructor R to type the representation for types of kind Ω. Types of higher kind are translated as functions from representations to representations. However, the kind polymorphism in λ_i^P complicates this. For example, consider the type $\lambda\alpha : \chi.\, \alpha$. To get the type of the runtime representation of α, we must know the kind χ. Therefore, we use a dictionary passing style at the type level. For every kind argument κ at a kind application, we supply the type function R_κ (bound by the variable α_χ) mapping types of kind κ to the types of their representation terms. We show the mapping in Figure 16.

In λ_i^P, the $\forall$ and the $\forall^+$ constructor bind a kind. But the language λ_R^P requires that every construct binding a kind should also bind the corresponding type dictionary. We therefore introduce tags at the type level corresponding to every type constructor in λ_i^P and a corresponding kind T. The type level analysis operator (Tagrec) now operates on tags. Therefore, we get the following translation of λ_i^P kinds to λ_R^P kinds.

$$\begin{array}{ll} |\Omega| = \mathsf{T} & |\kappa \to \kappa'| = |\kappa| \to |\kappa'| \\ |\chi| = \chi & |\forall\chi.\,\kappa| = \forall\chi.\,(\chi \to \Omega) \to |\kappa| \end{array}$$

The formation rule for the tags (Figure 17) follows directly from the kind translation and the λ_i^P kind of the corresponding type constructor. The mapping of λ_i^P types to λ_R^P tags (Figure 20) is also straightforward. The only interesting case is that of a kind function $\Lambda\chi.\,\tau$; the λ_R^P translation also binds a type dictionary α_χ. Since we do not have the R constructor in λ_i^P, we only need to fill in a type of the appropriate kind for the T_R branch of the Tagrec.

The Tagrec construct provides primitive recursion at the type level. Its reduction rule (Figure 18) is similar to that of the $\mathsf{Typerec}$ in λ_i^P. Consider the reduction for the $(T_\forall\,[\kappa']\,\tau_1\,\tau_2)$ constructor. Here τ_1 is the type dictionary for the kind κ' and τ_2 corresponds to the body of the $\forall$ constructor of λ_i^P. The Tagrec applies the $\tau_\forall$ branch to the kind κ', the dictionary τ_1, the body τ_2, and the result of the iteration over the body.

$$\frac{\begin{array}{c}\mathcal{E};\Delta,\alpha:\kappa' \vdash \mathsf{Tagrec}[\kappa]\;(\tau_2\,\alpha)\;\mathsf{of}\;(\tau_{\mathsf{int}};\,\tau_\to;\,\tau_\forall;\,\tau_{\forall^+};\,\tau_R) \\ \mapsto \tau' \;:\; \kappa\end{array}}{\begin{array}{c}\mathcal{E};\Delta \vdash \mathsf{Tagrec}[\kappa]\;(T_\forall\,[\kappa']\,\tau_1\,\tau_2)\;\mathsf{of}\;(\tau_{\mathsf{int}};\,\tau_\to;\,\tau_\forall;\,\tau_{\forall^+};\,\tau_R) \\ \mapsto \tau_\forall\,[\kappa']\,\tau_1\,\tau_2\,(\lambda\alpha:\kappa'.\,\tau') \;:\; \kappa\end{array}}$$

The term level in λ_R^P contains term level tags corresponding to the type constructors of λ_i^P. We introduce the constructor R at the type level to type the term level tags. Figure 17 shows the formation rules for the term level tags. Given a λ_i^P type τ of kind Ω, its term tag has the type $R\,(|\tau|)$ where $|\tau|$ is the type tag of τ. Intuitively, it makes sense since the $\mathsf{repcase}$ analyses the term tag and the Tagrec analyses $|\tau|$. The $\mathsf{repcase}$ has the obvious reduction rule (Figure 19); every branch is applied to the components of the corresponding term tag.

In Figure 21, we show the representation of λ_i^P types as λ_R^P terms. The key point is to maintain the invariant that every kind abstraction introduces the corresponding type tag and every type abstraction introduces the corresponding term tag. Therefore, the kind and type abstractions are translated as:

$$\Re(\Lambda\chi.\,\tau) = \Lambda^+\chi.\,\Lambda\alpha_\chi : \chi \to \Omega.\,\Re(\tau)$$

$$\Re(\lambda\alpha:\kappa.\,\tau) = \Lambda\alpha : |\kappa|.\,\lambda x_\alpha : R_\kappa\,\alpha.\,\Re(\tau)$$

The kind application and the type application must supply the corresponding tags. The type tag is R_κ (Figure 16) and the term tag is the translation itself. Therefore, the kind and type applications are translated as:

$$\Re(\tau\,[\kappa]) = \Re(\tau)\,[|\kappa|]^+\,[R_\kappa]$$

$$\Re(\tau\,\tau') = \Re(\tau)\,[|\tau'|]\,(\Re(\tau'))$$

The translation of type constructors follows from their kind. Consider the translation of the $\forall$ constructor. This constructor binds a kind κ and a type τ. Therefore, the translation introduces a kind and the corresponding type tag (χ and α_χ) and a type and the corresponding term tag (α and x_α). The $\mathsf{R}_\forall$ denotes that this is the term tag for the $\forall$ constructor.

$$\begin{array}{l}\Re(\forall) = \Lambda^+\chi.\,\Lambda\alpha_\chi : \chi \to \Omega.\,\Lambda\alpha : \chi \to \mathsf{T}.\,\lambda x_\alpha : R_{\chi\to\Omega}\,(\alpha). \\ \qquad \mathsf{R}_\forall\,(\chi, R_\chi, \alpha, x_\alpha)\end{array}$$

The $\mathsf{Typerec}$ translation uses a $\mathsf{repcase}$, and a fixpoint to simulate the recursion.

We show the translation of λ_i^P terms to λ_R^P terms in Figure 22. The interesting part of the translation is the use of the Tagrec construct to define the type of the translated term. This is possible only because our system is fully reflexive, but this is crucial for the term translation. In particular, to prove that the translation of a type application and a kind application are of the correct type, the type reduction relation must commute with respect to type and kind substitution which is enforced by the definition of our type analysis operators.

In Appendix B, we give the detailed semantics of λ_R^P and the translation from λ_i^P to λ_R^P.

We can prove the following propositions about the translation of λ_i^P to λ_R^P. The propositions always extend the original λ_i^P type environment Δ with a type environment $\Delta(\mathcal{E})$ which binds a type variable α_χ of kind $\chi \to \Omega$ for each $\chi \in \mathcal{E}$. Similarly the term-level translations extend the term environment Γ with $\Gamma(\Delta)$, binding a variable x_α of type $R_\kappa\alpha$ for each type variable α bound in Δ with kind κ.

Proposition 5.1 *If $\mathcal{E};\Delta \vdash \tau : \kappa$ holds in λ_i^P, then $|\mathcal{E}|;|\Delta|,\,\Delta(\mathcal{E}) \vdash |\tau| : |\kappa|$ holds in λ_R^P.*

The runtime representation $\Re(\tau)$ of a λ_i^P type τ in λ_R^P is computed as shown in Figure 21.

Proposition 5.2 *If $\mathcal{E};\Delta \vdash \tau : \kappa$ and $\mathcal{E};\Delta \vdash \Gamma$ hold in λ_i^P, then $|\mathcal{E}|;|\Delta|,\,\Delta(\mathcal{E});|\Gamma|,\,\Gamma(\Delta) \vdash \Re(\tau) : R_\kappa\,|\tau|$ holds in λ_R^P.*

Figure 22 gives the translation $|e|$ of λ_i^P terms to λ_R^P terms. The operational semantics of λ_R^P is summarized in Figure 19.

Proposition 5.3 *If $\mathcal{E};\Delta;\Gamma \vdash e : \tau$ holds in λ_i^P, then $|\mathcal{E}|;|\Delta|,\,\Delta(\mathcal{E});|\Gamma|,\,\Gamma(\Delta) \vdash |e| : \mathsf{Type}\,|\tau|$ holds in λ_R^P.*

6 Conclusions

We presented a type-theoretic framework for fully reflexive intensional analysis of types which includes analysis of polymorphic, existential, and recursive types. We can analyze arbitrary types both at the type level and at the term level. Moreover, we are not restricted to analyzing only parametrically polymorphic functions; we can also handle polymorphic functions that analyze the quantified type variable. We proved the calculus sound and showed that type checking still remains decidable. We gave an encoding of our calculus into a type erasure semantics. Since we can analyze arbitrary types, we can now use these constructs to write type-dependent language primitives that operate on all values. We showed how to use reflexive type analysis to support type-safe marshalling. We plan to implement this framework in the FLINT system [20].

References

[1] K. Crary and S. Weirich. Flexible type analysis. In *Proc. 1999 ACM SIGPLAN International Conference on Functional Programming*, pages 233–248. ACM Press, Sept. 1999.

[2] K. Crary, S. Weirich, and G. Morrisett. Intensional polymorphism in type-erasure semantics. In *Proc. 1998 ACM SIGPLAN International Conference on Functional Programming*, pages 301–312. ACM Press, Sept. 1998.

[3] D. Duggan. A type-based semantics for user-defined marshalling in polymorphic languages. In X. Leroy and A. Ohori, editors, *Proc. 1998 International Workshop on Types in Compilation*, volume 1473 of *LNCS*, pages 273–298, Kyoto, Japan, Mar. 1998. Springer-Verlag.

[4] L. Fegaras and T. Sheard. Revisiting catamorphism over datatypes with embedded functions. In *23rd Annual ACM Symp. on Principles of Programming Languages*, pages 284–294. ACM Press, Jan. 1996.

[5] J. Y. Girard. *Interpretation Fonctionnelle et Elimination des Coupures dans l'Arithmetique d'Ordre Superieur*. PhD thesis, University of Paris VII, 1972.

[6] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1989.

[7] R. Harper and G. Morrisett. Compiling polymorphism using intensional type analysis. In *Proc. 22nd Annual ACM Symp. on Principles of Programming Languages*, pages 130–141. ACM Press, Jan. 1995.

[8] M. P. Jones. *Qualified Types: Theory and Practice*. PhD thesis, Oxford University Computing Laboratory, Oxford, July 1992. Technical Monograph PRG-106.

[9] C. League, Z. Shao, and V. Trifonov. Representing Java classes in a typed intermediate language. In *Proc. 1999 ACM SIGPLAN International Conference on Functional Programming (ICFP'99)*, pages 183–196. ACM Press, September 1999.

[10] E. Meijer and G. Hutton. Bananas in space: Extending fold and unfold to exponential types. In *Functional Programming and Computer Architecture*, 1995.

[11] Y. Minamide, G. Morrisett, and R. Harper. Typed closure conversion. In *Proc. 23rd Annual ACM Symp. on Principles of Programming Languages*, pages 271–283. ACM Press, 1996.

[12] G. Morrisett, D. Walker, K. Crary, and N. Glew. From System F to typed assembly language. In *Proc. 25th Annual ACM Symp. on Principles of Programming Languages*, pages 85–97. ACM Press, Jan. 1998.

[13] G. C. Necula. *Compiling with Proofs*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, Sept. 1998.

[14] A. Ohori and K. Kato. Semantics for communication primitives in a polymorphic language. In *Proc. 20th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, pages 99–112. ACM Press, 1993.

[15] J. Peterson and M. Jones. Implementing type classes. In *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation*, pages 227–236. ACM Press, June 1993.

[16] F. Pfenning and C. Paulin-Mohring. Inductively defined types in the calculus of constructions. In *Proc. Fifth Conference on the Mathematical Foundations of Programming Semantics*, pages 209–228, New Orleans, Louisiana, Mar. 1989. Springer-Verlag.

[17] B. Pierce, S. Dietzen, and S. Michaylov. Programming in higher-order typed lambda-calculi. Technical Report CMU-CS-89-111, Carnegie Mellon University, 1989.

[18] J. C. Reynolds. Towards a theory of type structure. In *Proceedings, Colloque sur la Programmation, Lecture Notes in Computer Science, volume 19*, pages 408–425. Springer-Verlag, Berlin, 1974.

[19] Z. Shao. Flexible representation analysis. In *Proc. 1997 ACM SIGPLAN International Conference on Functional Programming*, pages 85–98. ACM Press, June 1997.

[20] Z. Shao. An overview of the FLINT/ML compiler. In *Proc. 1997 ACM SIGPLAN Workshop on Types in Compilation*, June 1997.

[21] Z. Shao. Typed cross-module compilation. In *Proc. 1998 ACM SIGPLAN International Conference on Functional Programming*. ACM Press, 1998.

[22] Z. Shao. Transparent modules with fully syntactic signatures. In *Proc. 1999 ACM SIGPLAN International Conference on Functional Programming (ICFP'99)*, pages 220–232. ACM Press, September 1999.

[23] Z. Shao and A. W. Appel. A type-based compiler for Standard ML. In *Proc. ACM SIGPLAN '95 Conf. on Prog. Lang. Design and Implementation*, pages 116–129, New York, 1995. ACM Press.

[24] D. Tarditi. *Design and Implementation of Code Optimizations for a Type-Directed Compiler for Standard ML*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, Dec. 1996. Tech Report CMU-CS-97-108.

[25] D. Tarditi, G. Morrisett, P. Cheng, C. Stone, R. Harper, and P. Lee. TIL: A type-directed optimizing compiler for ML. In *Proc. ACM SIGPLAN '96 Conf. on Prog. Lang. Design and Implementation*, pages 181–192. ACM Press, 1996.

[26] A. Wright and M. Felleisen. A syntactic approach to type soundness. Technical report, Dept. of Computer Science, Rice University, June 1992.

[27] Z. Yang. Encoding types in ML-like languages. In *Proc. 1998 ACM SIGPLAN International Conference on Functional Programming*, pages 289–300. ACM Press, 1998.

A Semantics of λ_R^P and Translation from λ_i^P

$\boxed{\text{Term environment formation} \quad \mathcal{E};\Delta \vdash \Gamma}$

$$\frac{\mathcal{E} \vdash \Delta}{\mathcal{E};\Delta \vdash \varepsilon} \qquad \frac{\mathcal{E};\Delta \vdash \Gamma \quad \mathcal{E};\Delta \vdash \tau : \Omega}{\mathcal{E};\Delta \vdash \Gamma, x{:}\tau}$$

$\boxed{\text{Term formation} \quad \mathcal{E};\Delta;\Gamma \vdash e : \tau}$

$$\frac{\mathcal{E};\Delta \vdash \Gamma \quad x{:}\tau \text{ in } \Gamma}{\mathcal{E};\Delta;\Gamma \vdash x : \tau} \qquad \frac{\mathcal{E};\Delta \vdash \Gamma}{\mathcal{E};\Delta;\Gamma \vdash i : \mathsf{int}} \qquad \frac{\mathcal{E};\Delta;\Gamma \vdash e : \tau \quad \mathcal{E};\Delta \vdash \tau \leadsto \tau' :}{\mathcal{E};\Delta;\Gamma \vdash e : \tau'}$$

$$\frac{\mathcal{E};\Delta \vdash \tau : \forall\chi.\natural\chi \to \natural\chi \quad \mathcal{E};\Delta;\Gamma \vdash e : \mu\tau}{\mathcal{E};\Delta;\Gamma \vdash \mathsf{unfold}\ e\ \mathsf{as}\ \tau : \tau\$(\mu\tau)}$$

$$\frac{\mathcal{E};\Delta \vdash \tau : \forall\chi.\natural\chi \to \natural\chi \quad \mathcal{E};\Delta;\Gamma \vdash e : \tau\$(\mu\tau)}{\mathcal{E};\Delta;\Gamma \vdash \mathsf{fold}\ e\ \mathsf{as}\ \tau : \mu\tau}$$

$$\frac{\mathcal{E},\chi;\Delta;\Gamma \vdash v : \tau}{\mathcal{E};\Delta;\Gamma \vdash \overset{+}{\Lambda}\chi.\,v : \overset{+}{\forall}\chi.\,\tau} \qquad \frac{\mathcal{E};\Delta;\Gamma \vdash e : \overset{+}{\mathbb{\forall}}\tau \quad \mathcal{E} \vdash \kappa}{\mathcal{E};\Delta;\Gamma \vdash e\,[\kappa]^{+} : \tau\,[\kappa]}$$

$$\frac{\mathcal{E};\Delta,\alpha{:}\kappa;\Gamma \vdash e : \tau}{\mathcal{E};\Delta;\Gamma \vdash \Lambda\alpha{:}\kappa.\,e : \forall\alpha{:}\kappa.\,\tau} \qquad \frac{\mathcal{E};\Delta;\Gamma,x{:}\tau \vdash e : \tau'}{\mathcal{E};\Delta;\Gamma \vdash \lambda x{:}\tau.\,e : \tau \to \tau'}$$

$$\frac{\mathcal{E};\Delta;\Gamma \vdash e : \mathbb{\forall}\,[\kappa]\,\tau \quad \mathcal{E};\Delta \vdash \tau' : \kappa}{\mathcal{E};\Delta;\Gamma \vdash e\,[\tau'] : \tau\,\tau'}$$

$$\frac{\mathcal{E};\Delta;\Gamma \vdash e_1 : \tau_2 \to \tau_1 \quad \mathcal{E};\Delta;\Gamma \vdash e_2 : \tau_2}{\mathcal{E};\Delta;\Gamma \vdash e_1\,e_2 : \tau_1}$$

$$\frac{\begin{array}{c}\mathcal{E};\Delta;\Gamma,x{:}\tau \vdash v : \tau \\ \tau = \overset{+}{\forall}\chi_1 \ldots \chi_n.\,\forall\alpha_1{:}\kappa_1 \ldots \alpha_m{:}\kappa_m{:}\tau_1 \to \tau_2. \\ n \geq 0, m \geq 0\end{array}}{\mathcal{E};\Delta;\Gamma \vdash \mathsf{fix}\,x{:}\tau.\,v : \tau}$$

$$\frac{\begin{array}{c}\mathcal{E};\Delta \vdash \tau : \Omega \to \Omega \\ \mathcal{E};\Delta \vdash \tau' : \Omega \\ \mathcal{E};\Delta;\Gamma \vdash e_{\mathsf{int}} : \tau\,\mathsf{int} \\ \mathcal{E};\Delta,\alpha_1{:}\Omega,\alpha_2{:}\Omega;\Gamma \vdash e_{\to} : \tau\,(\alpha_1 \to \alpha_2) \\ \mathcal{E},\chi;\Delta,\alpha{:}\chi \to \Omega;\Gamma \vdash e_{\forall} : \tau\,(\mathbb{\forall}\,[\chi]\,\alpha) \\ \mathcal{E};\Delta,\alpha{:}\forall\chi.\,\Omega;\Gamma \vdash e_{\forall^+} : \tau\,(\overset{+}{\mathbb{\forall}}\alpha) \\ \mathcal{E};\Delta,\alpha{:}\forall\chi.\,\natural\chi \to \natural\chi;\Gamma \vdash e_{\mu} : \tau\,(\mu\,\alpha)\end{array}}{\mathcal{E};\Delta;\Gamma \vdash \mathsf{typecase}[\tau]\,\tau'\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\to};\ e_{\forall};\ e_{\forall^+};\ e_{\mu}) : \tau\,\tau'}$$

Figure 11: λ_i^Q term formation rules

$$(\lambda x{:}\tau.\,e)\,v \leadsto e\{v/x\} \qquad (\mathsf{fix}\,x{:}\tau.\,v)\,v' \leadsto (v\{\mathsf{fix}\,x{:}\tau.\,v/x\})\,v'$$

$$(\Lambda\alpha{:}\kappa.\,v)\,[\tau] \leadsto v\{\tau/\alpha\} \qquad (\mathsf{fix}\,x{:}\tau.\,v)\,[\tau] \leadsto (v\{\mathsf{fix}\,x{:}\tau.\,v/x\})\,[\tau]$$

$$(\Lambda^{+}\chi.\,v)\,[\kappa]^{+} \leadsto v\{\kappa/\chi\} \qquad (\mathsf{fix}\,x{:}\tau.\,v)\,[\kappa]^{+} \leadsto (v\{\mathsf{fix}\,x{:}\tau.\,v/x\})\,[\kappa]^{+}$$

$$\mathsf{unfold}\ (\mathsf{fold}\ v\ \mathsf{as}\ \tau)\ \mathsf{as}\ \tau \leadsto v$$

$$\mathsf{typecase}[\tau]\ \mathsf{int}\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\to};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \leadsto e_{\mathsf{int}}$$

$$\mathsf{typecase}[\tau]\ (\tau_1 \to \tau_2)\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\to};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \leadsto e_{\to}\,[\tau_1]\,[\tau_2]$$

$$\mathsf{typecase}[\tau]\ (\forall\,[\kappa]\,\tau')\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\to};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \leadsto e_{\forall}\,[\kappa]^{+}\,[\tau']$$

$$\mathsf{typecase}[\tau]\ (\forall^{+}\tau')\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\to};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \leadsto e_{\forall^{+}}\,[\tau']$$

$$\mathsf{typecase}[\tau]\ (\mu\tau')\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\to};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \leadsto e_{\mu}\,[\tau']$$

$$\frac{e \leadsto e_1}{e\,e' \leadsto e_1\,e'} \qquad \frac{e \leadsto e_1}{v\,e \leadsto v\,e_1} \qquad \frac{e \leadsto e_1}{e\,[\tau] \leadsto e_1\,[\tau]} \qquad \frac{e \leadsto e_1}{e\,[\kappa]^{+} \leadsto e_1\,[\kappa]^{+}}$$

$$\frac{e \leadsto e_1}{\mathsf{fold}\ e\ \mathsf{as}\ \tau \leadsto \mathsf{fold}\ e_1\ \mathsf{as}\ \tau} \qquad \frac{e \leadsto e_1}{\mathsf{unfold}\ e\ \mathsf{as}\ \tau \leadsto \mathsf{unfold}\ e_1\ \mathsf{as}\ \tau}$$

$$\frac{\varepsilon;\varepsilon \vdash \tau' \leadsto^{*} \nu' : \Omega \qquad \nu'\ \text{is normal form}}{\begin{array}{l}\mathsf{typecase}[\tau]\ \tau'\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\to};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \leadsto \\ \qquad \mathsf{typecase}[\tau]\ \nu'\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\to};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu})\end{array}}$$

Figure 13: Operational semantics of λ_i^Q

$$\begin{array}{llcl}
(\textit{value}) & v & ::= & i \mid \lambda x{:}\tau.\,e \mid \mathsf{fold}\ v\ \mathsf{as}\ \tau \mid \mathsf{unfold}\ v\ \mathsf{as}\ \tau \\
 & & \mid & \Lambda\alpha{:}\kappa.\,v \mid \Lambda^{+}\chi.\,v \mid \mathsf{fix}\,x{:}\tau.\,v \\[1ex]
(\textit{context}) & E & ::= & [\,] \mid E\,e \mid v\,E \mid E\,[\tau] \mid E\,[\kappa]^{+} \\
 & & \mid & \mathsf{fold}\ E\ \mathsf{as}\ \tau \mid \mathsf{unfold}\ E\ \mathsf{as}\ \tau \\[1ex]
(\textit{redex}) & r & ::= & (\lambda x{:}\tau.\,e)\,v \mid (\Lambda\alpha{:}\kappa.\,v)\,[\tau] \mid (\Lambda^{+}\chi.\,v)\,[\kappa]^{+} \\
 & & \mid & (\mathsf{fix}\,x{:}\tau.\,v)\,v' \mid (\mathsf{fix}\,x{:}\tau.\,v)\,[\tau'] \\
 & & \mid & (\mathsf{fix}\,x{:}\tau.\,v)\,[\kappa]^{+} \\
 & & \mid & \mathsf{unfold}\ (\mathsf{fold}\ v\ \mathsf{as}\ \tau)\ \mathsf{as}\ \tau \\
 & & \mid & \mathsf{typecase}[\tau]\ \tau'\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\to};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \\
 & & \mid & \mathsf{typecase}[\tau]\ \mathsf{int}\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\to};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \\
 & & \mid & \mathsf{typecase}[\tau]\ (\tau' \to \tau'')\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\to};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \\
 & & \mid & \mathsf{typecase}[\tau]\ (\forall\,[\kappa]\,\tau')\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\to};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \\
 & & \mid & \mathsf{typecase}[\tau]\ (\forall^{+}\tau')\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\to};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \\
 & & \mid & \mathsf{typecase}[\tau]\ (\mu\tau')\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\to};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu})
\end{array}$$

Figure 14: Term contexts

$$\begin{array}{llcl}
(\textit{kinds}) & \kappa & ::= & \Omega \mid \mathsf{T} \mid \kappa \to \kappa' \mid \chi \mid \forall\chi.\,\kappa \\[1ex]
(\textit{types}) & \tau & ::= & \mathsf{int} \mid \twoheadrightarrow \mid \forall \mid \forall^{+} \mid R \\
 & & \mid & T_{\mathsf{int}} \mid T_{\to} \mid T_{\forall} \mid T_{\forall^{+}} \mid T_R \\
 & & \mid & \alpha \mid \Lambda\chi.\,\tau \mid \tau\,[\kappa] \mid \lambda\alpha{:}\kappa.\,\tau \mid \tau\,\tau' \\
 & & \mid & \mathsf{Tagrec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\, \tau_{\to};\, \tau_{\forall};\, \tau_{\forall^{+}};\, \tau_R) \\[1ex]
(\textit{fixtype}) & \sigma & ::= & \twoheadrightarrow\tau\,\tau' \mid \forall\,[\kappa]\,(\lambda\alpha{:}\kappa.\,\sigma) \mid \forall^{+}(\Lambda\chi.\,\sigma) \\[1ex]
(\textit{values}) & v & ::= & i \mid \Lambda^{+}\chi.\,v \mid \Lambda\alpha{:}\kappa.\,v \mid \lambda x{:}\tau.\,e \mid \mathsf{fix}\,x{:}\sigma.\,v \\
 & & \mid & v\,[\tau] \mid v\,[\kappa]^{+} \\
 & & \mid & \mathsf{R}_{\mathsf{int}} \mid \mathsf{R}_{\to}\,(\tau,\tau',v,v') \mid \mathsf{R}_{\forall}\,(\kappa,\tau,\tau',v') \\
 & & \mid & \mathsf{R}_{\forall^{+}}\,(\tau,v) \mid \mathsf{R}_{R}\,(\tau,v) \\[1ex]
(\textit{terms}) & e & ::= & v \mid x \mid e\,[\kappa]^{+} \mid e\,[\tau] \mid e\,e' \\
 & & \mid & \mathsf{repcase}[\tau]\ e\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\to};\, e_{\forall};\, e_{\forall^{+}};\, e_R) \\
 & & \mid & \mathsf{R}_{\mathsf{int}} \mid \mathsf{R}_{\to}\,(\tau,\tau',e,e') \mid \mathsf{R}_{\forall}\,(\kappa,\tau,\tau',e') \\
 & & \mid & \mathsf{R}_{\forall^{+}}\,(\tau,e) \mid \mathsf{R}_{R}\,(\tau,e)
\end{array}$$

Figure 15: Syntax of the λ_R^P language

$$\frac{\mathcal{E} \vdash \Delta}{\mathcal{E};\Delta \vdash R_{\Omega} \equiv R \,:\, \mathsf{T} \to \Omega} \qquad \frac{\mathcal{E};\Delta \vdash \alpha_{\chi} \,:\, \chi \to \Omega}{\mathcal{E};\Delta \vdash R_{\chi} \equiv \alpha_{\chi} \,:\, \chi \to \Omega}$$

$$\frac{\mathcal{E};\Delta \vdash R_{\kappa} \equiv \tau \,:\, |\kappa| \to \Omega \qquad \mathcal{E};\Delta \vdash R_{\kappa'} \equiv \tau' \,:\, |\kappa'| \to \Omega}{\begin{array}{l}\mathcal{E};\Delta \vdash R_{\kappa\to\kappa'} \equiv \lambda\alpha{:}|\kappa\to\kappa'|.\,\forall\beta{:}|\kappa|.\,\tau\,\beta \to \tau'\,(\alpha\,\beta) \\ \qquad : |\kappa\to\kappa'| \to \Omega\end{array}}$$

$$\frac{\mathcal{E},\chi;\Delta,\alpha_{\chi}{:}\chi\to\Omega \vdash R_{\kappa} \equiv \tau \,:\, |\kappa| \to \Omega}{\begin{array}{l}\mathcal{E};\Delta \vdash R_{\forall\chi.\,\kappa} \equiv \lambda\alpha{:}|\forall\chi.\,\kappa|.\,\forall^{+}\chi.\,\forall\alpha_{\chi}{:}\chi\to\Omega.\,\tau\,(\alpha\,[\chi]\,R_{\chi}) \\ \qquad : |\forall\chi.\,\kappa| \to \Omega\end{array}}$$

Figure 16: Types of representations at higher kinds

$\boxed{\text{Kind formation} \qquad \mathcal{E} \vdash \kappa}$

$$\mathcal{E} \vdash \mathsf{T}$$

$\boxed{\text{Type formation} \qquad \mathcal{E};\Delta \vdash \tau : \kappa}$

$$\frac{\mathcal{E} \vdash \Delta}{\begin{array}{l}\mathcal{E};\Delta \vdash R \ \ : \mathsf{T} \to \Omega \\ \mathcal{E};\Delta \vdash T_{\mathsf{int}} : \mathsf{T} \\ \mathcal{E};\Delta \vdash T_{\to} : \mathsf{T} \to \mathsf{T} \to \mathsf{T} \\ \mathcal{E};\Delta \vdash T_{\forall} : \forall\chi.\,(\chi \to \Omega) \to (\chi \to \mathsf{T}) \to \mathsf{T} \\ \mathcal{E};\Delta \vdash T_{\forall^+} : (\forall\chi.\,(\chi \to \Omega) \to \mathsf{T}) \to \mathsf{T} \\ \mathcal{E};\Delta \vdash T_{R} : \mathsf{T} \to \mathsf{T}\end{array}}$$

$$\frac{\begin{array}{l}\mathcal{E};\Delta \vdash \tau \ \ : \mathsf{T} \\ \mathcal{E};\Delta \vdash \tau_{\mathsf{int}} : \kappa \\ \mathcal{E};\Delta \vdash \tau_{\to} : \mathsf{T} \to \mathsf{T} \to \kappa \to \kappa \to \kappa \\ \mathcal{E};\Delta \vdash \tau_{\forall} : \forall\chi.\,(\chi \to \Omega) \to (\chi \to \mathsf{T}) \to (\chi \to \kappa) \to \kappa \\ \mathcal{E};\Delta \vdash \tau_{\forall^+} : (\forall\chi.\,(\chi \to \Omega) \to \mathsf{T}) \to (\forall\chi.\,(\chi \to \Omega) \to \kappa) \to \kappa \\ \mathcal{E};\Delta \vdash \tau_{R} : \mathsf{T} \to \kappa \to \kappa\end{array}}{\mathcal{E};\Delta \vdash \mathsf{Tagrec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{R}) : \kappa}$$

$\boxed{\text{Term formation} \qquad \mathcal{E};\Delta;\Gamma \vdash e : \tau}$

$$\frac{\mathcal{E};\Delta \vdash \Gamma}{\mathcal{E};\Delta;\Gamma \vdash \mathsf{R}_{\mathsf{int}} : R\,T_{\mathsf{int}}}$$

$$\frac{\mathcal{E};\Delta;\Gamma \vdash e : R\,\tau \qquad \mathcal{E};\Delta;\Gamma \vdash e' : R\,\tau'}{\mathcal{E};\Delta;\Gamma \vdash \mathsf{R}_{\to}\,(\tau,\tau',e,e') : R\,(T_{\to}\,\tau\,\tau')}$$

$$\frac{\mathcal{E};\Delta \vdash \tau : |\kappa| \to \Omega \qquad \mathcal{E};\Delta;\Gamma \vdash e' : R_{\kappa\to\Omega}\,(\tau')}{\mathcal{E};\Delta;\Gamma \vdash \mathsf{R}_{\forall}\,(|\kappa|,\tau,\tau',e') : R\,(T_{\forall}\,[|\kappa|]\,\tau\,\tau')}$$

$$\frac{\mathcal{E};\Delta;\Gamma \vdash e : R_{\forall\chi.\,\Omega}\,(\tau)}{\mathcal{E};\Delta;\Gamma \vdash \mathsf{R}_{\forall^+}\,(\tau,e) : R\,(\forall^+\tau)}$$

$$\frac{\mathcal{E};\Delta;\Gamma \vdash e : R\,\tau}{\mathcal{E};\Delta;\Gamma \vdash \mathsf{R}_{R}\,(\tau,e) : R\,(R\,\tau)}$$

$$\frac{\begin{array}{l}\mathcal{E};\Delta \vdash \tau : \mathsf{T} \to \Omega \\ \mathcal{E};\Delta;\Gamma \vdash e \ \ : R\,\tau' \\ \mathcal{E};\Delta;\Gamma \vdash e_{\mathsf{int}} : \tau\,T_{\mathsf{int}} \\ \mathcal{E};\Delta;\Gamma \vdash e_{\to} : \forall\alpha_1 \!:\! \mathsf{T}.\,R\,\alpha_1 \to \forall\alpha_2 \!:\! \mathsf{T}.\,R\,\alpha_2 \to \tau\,(T_{\to}\,\alpha_1\,\alpha_2) \\ \mathcal{E};\Delta;\Gamma \vdash e_{\forall} : \forall^+\chi.\,\forall\alpha_{\chi} \!:\! \chi \to \Omega. \\ \qquad\qquad\qquad \forall\alpha \!:\! \chi \to \mathsf{T}.\,R_{\chi\to\Omega}\,(\alpha) \to \tau\,(T_{\forall}\,[\chi]\,\alpha_{\chi}\,\alpha) \\ \mathcal{E};\Delta;\Gamma \vdash e_{\forall^+} : \forall\alpha \!:\! \forall\chi.\,(\chi \to \Omega) \to \mathsf{T}.\,R_{\forall\chi.\,\Omega}\,(\alpha) \to \tau\,(T_{\forall^+}\,\alpha) \\ \mathcal{E};\Delta;\Gamma \vdash e_{R} : \forall\alpha \!:\! \mathsf{T}.\,R\,\alpha \to \tau\,(T_{R}\,\alpha)\end{array}}{\mathcal{E};\Delta;\Gamma \vdash \mathsf{repcase}[\tau]\ e\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\to};\ e_{\forall};\ e_{\forall^+};\ e_{R}) : \tau\,\tau'}$$

Figure 17: Formation rules for the new constructs in λ_R^P

$\boxed{\text{Type reduction} \qquad \mathcal{E};\Delta \vdash \tau \mapsto \tau' : \kappa}$

$$\frac{\mathcal{E};\Delta \vdash \mathsf{Tagrec}[\kappa]\ T_{\mathsf{int}}\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{R}) : \kappa}{\mathcal{E};\Delta \vdash \mathsf{Tagrec}[\kappa]\ T_{\mathsf{int}}\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{R}) \mapsto \tau_{\mathsf{int}} : \kappa}$$

$$\frac{\begin{array}{l}\mathcal{E};\Delta \vdash \mathsf{Tagrec}[\kappa]\ \tau_1\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{R}) \mapsto \tau_1' : \kappa \\ \mathcal{E};\Delta \vdash \mathsf{Tagrec}[\kappa]\ \tau_2\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{R}) \mapsto \tau_2' : \kappa\end{array}}{\begin{array}{l}\mathcal{E};\Delta \vdash \mathsf{Tagrec}[\kappa]\ (T_{\to}\,\tau_1\,\tau_2)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{R}) \\ \qquad \mapsto \tau_{\to}\,\tau_1\,\tau_2\,\tau_1'\,\tau_2' : \kappa\end{array}}$$

$$\frac{\begin{array}{l}\mathcal{E};\Delta,\alpha \!:\! \kappa' \vdash \mathsf{Tagrec}[\kappa]\ (\tau_2\,\alpha)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{R}) \\ \qquad \mapsto \tau' : \kappa\end{array}}{\begin{array}{l}\mathcal{E};\Delta \vdash \mathsf{Tagrec}[\kappa]\ (T_{\forall}\,[\kappa']\,\tau_1\,\tau_2)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{R}) \\ \qquad \mapsto \tau_{\forall}\,[\kappa']\,\tau_1\,\tau_2\,(\lambda\alpha \!:\! \kappa'.\,\tau') : \kappa\end{array}}$$

$$\frac{\begin{array}{l}\mathcal{E},\chi;\Delta,\alpha_{\chi} \!:\! \chi \to \Omega \vdash \\ \mathsf{Tagrec}[\kappa]\ (\tau\,[\chi]\,\alpha_{\chi})\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{R}) \mapsto \tau' : \kappa\end{array}}{\begin{array}{l}\mathcal{E};\Delta \vdash \mathsf{Tagrec}[\kappa]\ (T_{\forall^+}\,\tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{R}) \\ \qquad \mapsto \tau_{\forall^+}\,\tau\,(\Lambda\chi.\,\lambda\alpha_{\chi} \!:\! \chi \to \Omega.\,\tau') : \kappa\end{array}}$$

$$\frac{\mathcal{E};\Delta \vdash \mathsf{Tagrec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{R}) \mapsto \tau' : \kappa}{\mathcal{E};\Delta \vdash \mathsf{Tagrec}[\kappa]\ (T_{R}\,\tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{R}) \mapsto \tau_{R}\,\tau\,\tau' : \kappa}$$

Figure 18: Non-standard reduction rules for λ_R^P types

$\mathsf{repcase}[\tau]\ \mathsf{R}_{\mathsf{int}}\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\to};\ e_{\forall};\ e_{\forall^+};\ e_R) \rightsquigarrow e_{\mathsf{int}}$

$\mathsf{repcase}[\tau]\ \mathsf{R}_{\to}\,(\tau,\tau',e,e')\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\to};\ e_{\forall};\ e_{\forall^+};\ e_R) \rightsquigarrow$
$\qquad e_{\to}\,[\tau]\,[\tau']\,e\,e'$

$\mathsf{repcase}[\tau]\ \mathsf{R}_{\forall}\,(\kappa,\tau,\tau',e')\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\to};\ e_{\forall};\ e_{\forall^+};\ e_R) \rightsquigarrow$
$\qquad e_{\forall}\,[\kappa]^{+}\,[\tau]\,[\tau']\,e'$

$\mathsf{repcase}[\tau]\ \mathsf{R}_{\forall^+}\,(\tau,e)\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\to};\ e_{\forall};\ e_{\forall^+};\ e_R) \rightsquigarrow e_{\forall^+}\,[\tau]\,e$

$\mathsf{repcase}[\tau]\ \mathsf{R}_{R}\,(\tau,e)\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\to};\ e_{\forall};\ e_{\forall^+};\ e_R) \rightsquigarrow e_{R}\,[\tau]\,e$

$$\frac{e \rightsquigarrow e'}{\begin{array}{l}\mathsf{repcase}[\tau]\ e\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\to};\ e_{\forall};\ e_{\forall^+};\ e_R) \rightsquigarrow \\ \qquad \mathsf{repcase}[\tau]\ e'\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\to};\ e_{\forall};\ e_{\forall^+};\ e_R)\end{array}}$$

Figure 19: New term reduction rules of λ_R^P

$$\begin{array}{ll}|\alpha| = \alpha & \\ |\mathsf{int}| = T_{\mathsf{int}} & |\Lambda\chi.\,\tau| = \Lambda\chi.\,\lambda\alpha_{\chi} \!:\! \chi \to \Omega.\,|\tau| \\ |\to| = T_{\to} & |\tau\,[\kappa]| = |\tau|\,[|\kappa|]\,R_{\kappa} \\ |\forall| = T_{\forall} & |\lambda\alpha \!:\! \kappa.\,\tau| = \lambda\alpha \!:\! |\kappa|.\,|\tau| \\ |\forall^+| = T_{\forall^+} & |\tau\,\tau'| = |\tau|\,|\tau'|\end{array}$$

$$\begin{array}{l}|\mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+})| \\ \quad = \mathsf{Tagrec}[|\kappa|]\ |\tau|\ \mathsf{of} \\ \qquad (|\tau_{\mathsf{int}}|;\ |\tau_{\to}|;\ |\tau_{\forall}|;\ |\tau_{\forall^+}|;\ \lambda\_ \!:\! \mathsf{T}.\,\lambda\_ \!:\! |\kappa|.\,|\tau_{\mathsf{int}}|)\end{array}$$

Figure 20: Mapping of λ_i^P types to λ_R^P tags

$$
\begin{aligned}
\Re(\mathsf{int}) &= \mathsf{R_{int}} \\
\Re(\rightarrow) &= \Lambda\alpha\!:\!\mathsf{T}.\,\lambda x_\alpha\!:\!R\,\alpha.\,\Lambda\beta\!:\!\mathsf{T}.\,\lambda x_\beta\!:\!R\,\beta. \\
&\qquad \mathsf{R}_\rightarrow\,(\alpha,\beta,x_\alpha,x_\beta) \\
\Re(\forall) &= \Lambda^{+}\chi.\,\Lambda\alpha_\chi\!:\!\chi\rightarrow\Omega.\,\Lambda\alpha\!:\!\chi\rightarrow\mathsf{T}.\,\lambda x_\alpha\!:\!R_{\chi\rightarrow\Omega}\,(\alpha). \\
&\qquad \mathsf{R}_\forall\,(\chi,R_\chi,\alpha,x_\alpha) \\
\Re(\forall^{+}) &= \Lambda\alpha\!:\!(\forall\chi.\,(\chi\rightarrow\Omega)\rightarrow\mathsf{T}).\,\lambda x_\alpha\!:\!R_{\forall\chi.\,\Omega}\,\alpha. \\
&\qquad \mathsf{R}_{\forall^+}\,(\alpha,x_\alpha) \\
\Re(\alpha) &= x_\alpha \\
\Re(\Lambda\chi.\,\tau) &= \Lambda^{+}\chi.\,\Lambda\alpha_\chi\!:\!\chi\rightarrow\Omega.\,\Re(\tau) \\
\Re(\tau\,[\kappa]) &= \Re(\tau)\,[|\kappa|]^{+}\,[R_\kappa] \\
\Re(\lambda\alpha\!:\!\kappa.\,\tau) &= \Lambda\alpha\!:\!|\kappa|.\,\lambda x_\alpha\!:\!R_\kappa\,\alpha.\,\Re(\tau) \\
\Re(\tau\,\tau') &= \Re(\tau)\,[|\tau'|]\,(\Re(\tau'))
\end{aligned}
$$

$$
\begin{aligned}
&\Re(\mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_\rightarrow;\ \tau_\forall;\ \tau_{\forall^+})) \\
&= (\mathsf{fix}\,\mathsf{f}\!:\!\forall\alpha\!:\!\mathsf{T}.\,R\,\alpha\rightarrow R_\kappa\,(\tau^*\,\alpha). \\
&\quad \Lambda\alpha\!:\!\mathsf{T}.\,\lambda x_\alpha\!:\!R\,\alpha. \\
&\quad\quad \mathsf{repcase}[\lambda\alpha\!:\!\mathsf{T}.\,R_\kappa\,(\tau^*\,\alpha)]\ x_\alpha\ \mathsf{of} \\
&\quad\quad\quad \mathsf{R_{int}} \Rightarrow \Re(\tau_{\mathsf{int}}) \\
&\quad\quad\quad \mathsf{R}_\rightarrow \Rightarrow \Lambda\alpha\!:\!\mathsf{T}.\,\lambda x_\alpha\!:\!R\,\alpha.\,\Lambda\beta\!:\!\mathsf{T}.\,\lambda x_\beta\!:\!R\,\beta. \\
&\quad\quad\quad\qquad \Re(\tau_\rightarrow)\,[\alpha]\,x_\alpha\,[\beta]\,x_\beta \\
&\quad\quad\quad\qquad [\tau^*\,\alpha]\,(\mathsf{f}\,[\alpha]\,x_\alpha)\,[\tau^*\,\beta]\,(\mathsf{f}\,[\beta]\,x_\beta) \\
&\quad\quad\quad \mathsf{R}_\forall \Rightarrow \Lambda^{+}\chi.\,\Lambda\alpha_\chi\!:\!\chi\rightarrow\Omega.\,\Lambda\alpha\!:\!\chi\rightarrow\mathsf{T}.\,\lambda x_\alpha\!:\!R_{\chi\rightarrow\Omega}\,(\alpha). \\
&\quad\quad\quad\qquad \Re(\tau_\forall)\,[\chi]^{+}\,[R_\chi]\,[\alpha]\,x_\alpha\,[\lambda\beta\!:\!\chi.\,\tau^*\,(\alpha\,\beta)] \\
&\quad\quad\quad\qquad\quad (\Lambda\beta\!:\!\chi.\,\lambda x_\beta\!:\!R_\chi\,\beta.\,\mathsf{f}\,[\alpha\,\beta]\,(x_\alpha\,[\beta]\,x_\beta)) \\
&\quad\quad\quad \mathsf{R}_{\forall^+} \Rightarrow \Lambda\alpha\!:\!(\forall\chi.\,(\chi\rightarrow\Omega)\rightarrow\mathsf{T}).\,\lambda x_\alpha\!:\!R_{\forall\chi.\,\Omega}\,\alpha. \\
&\quad\quad\quad\qquad \Re(\tau_{\forall^+})\,[\alpha]\,x_\alpha \\
&\quad\quad\quad\qquad [\Lambda\chi.\,\lambda\alpha_\chi\!:\!\chi\rightarrow\Omega.\,\tau^*\,(\alpha\,[\chi]\,R_\chi)] \\
&\quad\quad\quad\qquad (\Lambda^{+}\chi.\,\Lambda\alpha_\chi\!:\!\chi\rightarrow\Omega.\,\mathsf{f}\,[\alpha\,[\chi]\,R_\chi]\,(x_\alpha\,[\chi]^{+}\,[R_\chi])) \\
&\quad\quad\quad \mathsf{R}_R \Rightarrow \Lambda\alpha\!:\!\mathsf{T}.\,\lambda x\!:\!R\,\alpha.\,\Re(\tau_{\mathsf{int}})) \\
&\quad [|\tau|] \\
&\quad \Re(\tau) \\
&\quad \text{where} \\
&\quad\quad \tau^* = |\lambda\alpha\!:\!\Omega.\,\mathsf{Typerec}[\kappa]\ \alpha\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_\rightarrow;\ \tau_\forall;\ \tau_{\forall^+})|
\end{aligned}
$$

Figure 21: Representation of λ_i^P types as λ_R^P terms

$$
\begin{aligned}
|i| &= i \\
|x| &= x \\
|\Lambda^{+}\chi.\,v| &= \Lambda^{+}\chi.\,\Lambda\alpha_\chi\!:\!\chi\rightarrow\Omega.\,|v| \\
|e\,[\kappa]^{+}| &= |e|\,[|\kappa|]^{+}\,[R_\kappa] \\
|\Lambda\alpha\!:\!\kappa.\,v| &= \Lambda\alpha\!:\!|\kappa|.\,\lambda x_\alpha\!:\!R_\kappa\,\alpha.\,|v| \\
|e\,[\tau]| &= |e|\,[|\tau|]\,\Re(\tau) \\
|\lambda x\!:\!\tau.\,e| &= \lambda x\!:\!\mathsf{Type}\,|\tau|.\,|e| \\
|e\,e'| &= |e|\,|e'|
\end{aligned}
$$

$$
\begin{aligned}
&|\mathsf{typecase}[\tau]\ \tau'\ \mathsf{of}\ (e_{\mathsf{int}};\ e_\rightarrow;\ e_\forall;\ e_{\forall^+})| \\
&\quad = \mathsf{repcase}[\lambda\alpha\!:\!\mathsf{T}.\,\mathsf{Type}\,(|\tau|\,\alpha)]\ \Re(\tau')\ \mathsf{of} \\
&\qquad\quad \mathsf{R_{int}} \Rightarrow |e_{\mathsf{int}}| \\
&\qquad\quad \mathsf{R}_\rightarrow \Rightarrow |e_\rightarrow| \\
&\qquad\quad \mathsf{R}_\forall \Rightarrow |e_\forall| \\
&\qquad\quad \mathsf{R}_{\forall^+} \Rightarrow |e_{\forall^+}| \\
&\qquad\quad \mathsf{R}_R \Rightarrow \Lambda\alpha\!:\!\mathsf{T}.\,\lambda x\!:\!R\,\alpha.\,|e_{\mathsf{int}}| \\
&\text{where} \\
&\ \mathsf{Type}\ =\ \lambda\alpha\!:\!\mathsf{T}.\,\mathsf{Tagrec}[\Omega]\ \alpha\ \mathsf{of} \\
&\qquad\qquad T_{\mathsf{int}} \Rightarrow \mathsf{int} \\
&\qquad\qquad T_\rightarrow \Rightarrow \lambda\_\!:\!\mathsf{T}.\,\lambda\_\!:\!\mathsf{T}.\,\lambda\alpha_1\!:\!\Omega.\,\lambda\alpha_2\!:\!\Omega. \\
&\qquad\qquad\qquad \alpha_1\rightarrow\alpha_2 \\
&\qquad\qquad T_\forall \Rightarrow \Lambda\chi.\,\lambda\alpha_\chi\!:\!\chi\rightarrow\Omega.\,\lambda\_\!:\!\chi\rightarrow\mathsf{T}. \\
&\qquad\qquad\qquad \lambda\alpha'\!:\!\chi\rightarrow\Omega.\,\forall\alpha\!:\!\chi.\,R_\chi\,\alpha\rightarrow\alpha'\,\alpha \\
&\qquad\qquad T_{\forall^+} \Rightarrow \lambda\_\!:\!(\forall\chi.\,(\chi\rightarrow\Omega)\rightarrow\mathsf{T}). \\
&\qquad\qquad\qquad \lambda\alpha\!:\!(\forall\chi.\,(\chi\rightarrow\Omega)\rightarrow\Omega). \\
&\qquad\qquad\qquad\quad \forall^{+}\chi.\,\forall\alpha_\chi\!:\!\chi\rightarrow\Omega.\,\alpha\,[\chi]\,R_\chi \\
&\qquad\qquad T_R \Rightarrow \mathsf{int}
\end{aligned}
$$

Figure 22: Translation of λ_i^P terms to λ_R^P

$$
\begin{array}{llcl}
(\mathit{value}) & v & ::= & i \mid \lambda x\!:\!\tau.\, e \mid \mathsf{fix}\, x\!:\!\tau.\, v \mid \Lambda\alpha\!:\!\kappa.\, v \mid \Lambda^{+}\chi.\, v \\
\\
(\mathit{context}) & E & ::= & [\,] \mid E\, e \mid v\, E \mid E\,[\tau] \mid E\,[\kappa]^{+} \\
\\
(\mathit{redex}) & r & ::= & (\lambda x\!:\!\tau.\, e)\, v \mid (\Lambda\alpha\!:\!\kappa.\, v)\,[\tau] \mid (\Lambda^{+}\chi.\, e)\,[\kappa]^{+} \\
 & & \mid & (\mathsf{fix}\, x\!:\!\tau.\, v)\, v' \mid (\mathsf{fix}\, x\!:\!\tau.\, v)\,[\tau'] \\
 & & \mid & (\mathsf{fix}\, x\!:\!\tau.\, v)\,[\kappa]^{+} \\
 & & \mid & \mathsf{typecase}[\tau]\ \tau'\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\rightarrow};\ e_{\forall};\ e_{\forall^{+}}) \\
 & & \mid & \mathsf{typecase}[\tau]\ \mathsf{int}\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\rightarrow};\ e_{\forall};\ e_{\forall^{+}}) \\
 & & \mid & \mathsf{typecase}[\tau]\ \tau \rightarrow \tau'\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\rightarrow};\ e_{\forall};\ e_{\forall^{+}}) \\
 & & \mid & \mathsf{typecase}[\tau]\ \forall\,[\kappa]\,\tau\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\rightarrow};\ e_{\forall};\ e_{\forall^{+}}) \\
 & & \mid & \mathsf{typecase}[\tau]\ \forall^{+}\tau\ \mathsf{of}\ (e_{\mathsf{int}};\ e_{\rightarrow};\ e_{\forall};\ e_{\forall^{+}})
\end{array}
$$

Figure 23: Term contexts

$$
\begin{array}{lcl}
\nu^0 & ::= & \alpha \mid \nu^0\,\nu \mid \nu^0\,[\kappa] \\
 & \mid & \mathsf{Typerec}[\kappa]\ \nu^0\ \mathsf{of}\ (\nu_{\mathsf{int}};\ \nu_{\rightarrow};\ \nu_{\forall};\ \nu_{\forall^{+}}) \\
\nu & ::= & \nu^0 \mid \mathsf{int} \mid \twoheadrightarrow \mid (\twoheadrightarrow)\,\nu \mid (\twoheadrightarrow)\,\nu\,\nu' \\
 & \mid & \forall \mid \forall\,[\kappa] \mid \forall\,[\kappa]\,\nu \mid \forall^{+} \mid \forall^{+}\nu \\
 & \mid & \lambda\alpha\!:\!\kappa.\,\nu,\ \text{where } \forall\nu^0.\ \nu \neq \nu^0\,\alpha \text{ or } \alpha \in \mathit{ftv}(\nu^0) \\
 & \mid & \Lambda\chi.\,\nu,\ \text{where } \forall\nu^0.\ \nu \neq \nu^0\,[\chi] \text{ or } \chi \in \mathit{fkv}(\nu^0)
\end{array}
$$

Figure 24: Normal forms in the λ_i^P type language

B Properties of λ_i^P

B.1 Soundness of λ_i^P

The operational semantics for λ_i^P are in Figure 6. The reduction rules are standard except for the typecase construct. The typecase chooses a branch depending on the head constructor of the type being analyzed and passes the corresponding subtypes as arguments. For example, while analyzing the polymorphic type $\forall\,[\kappa]\,\tau$, it chooses the $e_\forall$ branch and applies it to the kind κ and the type function τ. If the type being analyzed is not in normal form, the typecase reduces the type to its unique normal form.

We prove soundness of the system by using contextual semantics in Wright/Felleisen style [26]. The evaluation contexts E are shown in Figure 23. The reduction rules for the redexes r are shown in Figure 6. We assume unique variable names and our environments are sets of variables. The notation $\vdash e : \tau$ is used a shorthand for $\varepsilon; \varepsilon; \varepsilon \vdash e : \tau$.

Lemma B.1 *If $\varepsilon; \varepsilon \vdash \nu : \Omega$, then ν is one of* int*, $\nu_1 \rightarrow \nu_2$, $\forall\,[\kappa]\,\nu_1$, or $\forall^{+}\nu_1$.*

Proof Since ν is well-formed in an empty environment, it does not contain any free type or kind variables. Therefore ν can not be a ν^0 since the head of a ν^0 is a type variable. The lemma now follows by inspecting the remaining possibilities for ν. ☐

Lemma B.2 (Decomposition of terms) *If $\vdash e : \tau$, then either e is a value or it can be decomposed into unique E and r such that $e = E\,[r]$.*

This is proved by induction over the derivation of $\vdash e : \tau$, using Lemma B.1 in the case of the typecase construct.

Corollary B.3 (Progress) *If $\vdash e : \tau$, then either e is a value or there exists an e' such that $e \mapsto e'$.*

Proof By Lemma B.2, we know that if $\vdash e : \tau$ and e is not a value, then there exist some E and redex e_1 such that $e = E\,[e_1]$. Since e_1 is a redex, there exists a contraction e_2 such that $e_1 \leadsto e_2$. Therefore $e \mapsto e'$ for $e' = E\,[e_2]$. ☐

Lemma B.4 *If $\vdash E\,[e] : \tau$, then there exists a τ' such that $\vdash e : \tau'$, and for all e' such that $\vdash e' : \tau'$ we have $\vdash E\,[e'] : \tau$.*

Proof The proof is by induction on the derivation of $\vdash E\,[e] : \tau$. The different forms of E are handled similarly; we will show only one case here.

- **case** $E = E_1\, e_1$: We have that $\vdash (E_1\,[e])\, e_1 : \tau$. By the typing rules, this implies that $\vdash E_1\,[e] : \tau_1 \rightarrow \tau$, for some τ_1. By induction, there exists a τ' such that $\vdash e : \tau'$ and for all e' such that $\vdash e' : \tau'$, we have that $\vdash E_1\,[e'] : \tau_1 \rightarrow \tau$. Therefore $\vdash (E_1\,[e'])\, e_1 : \tau$. ☐

As usual, the proof of soundness depends on several substitution lemmas; these are shown below. The proofs are fairly straightforward and proceed by induction on the derivation of the judgments. The notion of substitution is extended to environments in the usual way.

Lemma B.5 *If $\mathcal{E}, \chi \vdash \kappa$ and $\mathcal{E} \vdash \kappa'$, then $\mathcal{E} \vdash \kappa\{\kappa'/\chi\}$.*

Lemma B.6 *If $\mathcal{E}, \chi; \Delta \vdash \tau : \kappa$ and $\mathcal{E} \vdash \kappa'$, then $\mathcal{E}; \Delta\{\kappa'/\chi\} \vdash \tau\{\kappa'/\chi\} : \kappa\{\kappa'/\chi\}$.*

Lemma B.7 *If $\mathcal{E}, \chi; \Delta; \Gamma \vdash e : \tau$ and $\mathcal{E} \vdash \kappa$, then $\mathcal{E}; \Delta\{\kappa/\chi\}; \Gamma\{\kappa/\chi\} \vdash e\{\kappa/\chi\} : \tau\{\kappa/\chi\}$.*

Lemma B.8 *If $\mathcal{E}; \Delta, \alpha : \kappa' \vdash \tau : \kappa$ and $\mathcal{E}; \Delta \vdash \tau' : \kappa'$, then $\mathcal{E}; \Delta \vdash \tau\{\tau'/\alpha\} : \kappa$.*

Lemma B.9 *If $\mathcal{E}; \Delta, \alpha : \kappa; \Gamma \vdash e : \tau$ and $\mathcal{E}; \Delta \vdash \tau' : \kappa$, then $\mathcal{E}; \Delta; \Gamma\{\tau'/\alpha\} \vdash e\{\tau'/\alpha\} : \tau\{\tau'/\alpha\}$.*

Proof We prove this by induction on the structure of e. We demonstrate the proof here only for a few cases; the rest follow analogously.

- **case** $e = e_1\,[\tau_1]$: We have that $\mathcal{E}; \Delta \vdash \tau' : \kappa$. and also that $\mathcal{E}; \Delta, \alpha : \kappa; \Gamma \vdash e_1\,[\tau_1] : \tau$. By the typing rule for a type application we get that

  $\mathcal{E}; \Delta, \alpha\!:\!\kappa; \Gamma \vdash e_1 : \forall\beta\!:\!\kappa_1.\, \tau_2$ and
  $\mathcal{E}; \Delta, \alpha\!:\!\kappa \vdash \tau_1 : \kappa_1$ and
  $\tau = \tau_2\{\tau_1/\beta\}$

  By induction on e_1,

  $\mathcal{E}; \Delta; \Gamma\{\tau'/\alpha\} \vdash e_1\{\tau'/\alpha\} : \forall\beta\!:\!\kappa_1.\, \tau_2\{\tau'/\alpha\}$

  By Lemma B.8, $\mathcal{E}; \Delta \vdash \tau_1\{\tau'/\alpha\} : \kappa_1$. Therefore

  $$\mathcal{E}; \Delta; \Gamma\{\tau'/\alpha\} \vdash (e_1\{\tau'/\alpha\})\,[\tau_1\{\tau'/\alpha\}] : (\tau_2\{\tau'/\alpha\})\{\tau_1\{\tau'/\alpha\}/\beta\}$$

  But this is equivalent to

  $$\mathcal{E}; \Delta; \Gamma\{\tau'/\alpha\} \vdash (e_1\{\tau'/\alpha\})\,[\tau_1\{\tau'/\alpha\}] : (\tau_2\{\tau_1/\beta\})\{\tau'/\alpha\}$$

- **case** $e = e_1\,[\kappa_1]^{+}$: We have that $\mathcal{E}; \Delta, \alpha\!:\!\kappa; \Gamma \vdash e_1\,[\kappa_1]^{+} : \tau$ and $\mathcal{E}; \Delta \vdash \tau' : \kappa$. By the typing rule for kind application,

  $\mathcal{E}; \Delta, \alpha\!:\!\kappa; \Gamma \vdash e_1 : \forall\chi.\, \tau_1$ and
  $\tau = \tau_1\{\kappa_1/\chi\}$ and
  $\mathcal{E} \vdash \kappa_1$

By induction on e_1,

$\mathcal{E};\Delta;\Gamma \vdash e_1\{\tau'/\alpha\} \;:\; \forall\chi.\,\tau_1\{\tau'/\alpha\}$

Therefore

$\mathcal{E};\Delta;\Gamma \vdash (e_1\{\tau'/\alpha\})\,[\kappa_1]^{+} \;:\; (\tau_1\{\tau'/\alpha\})\{\kappa_1/\chi\}$

Since χ does not occur free in τ',

$(\tau_1\{\tau'/\alpha\})\{\kappa_1/\chi\} = (\tau_1\{\kappa_1/\chi\})\{\tau'/\alpha\}$

- **case** $e = \mathsf{typecase}[\tau_0]\;\tau_1\;\mathsf{of}\;(e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}})$: We have that $\mathcal{E};\Delta \vdash \tau' : \kappa$ and $\mathcal{E};\Delta,\alpha : \kappa;\Gamma \vdash \mathsf{typecase}[\tau_0]\;\tau_1\;\mathsf{of}\;(e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}}) \;:\; \tau_0\,\tau_1$. Using Lemma B.8 on the kind derivation of τ_0 and τ_1, and the inductive assumption on the typing rules for the subterms we get,

  $\mathcal{E};\Delta \vdash \tau_0\{\tau'/\alpha\} \;:\; \Omega \rightarrow \Omega$ and
  $\mathcal{E};\Delta \vdash \tau_1\{\tau'/\alpha\} \;:\; \Omega$ and
  $\mathcal{E};\Delta;\Gamma\{\tau'/\alpha\} \vdash e_{\mathsf{int}}\{\tau'/\alpha\} \;:\; (\tau_0\;\mathsf{int})\{\tau'/\alpha\}$ and
  $\mathcal{E};\Delta;\Gamma\{\tau'/\alpha\} \vdash e_{\rightarrow}\{\tau'/\alpha\} \;:$
  $\quad(\forall\alpha_1{:}\Omega.\,\forall\alpha_2{:}\Omega.\,\tau_0\,(\alpha_1 \rightarrow \alpha_2))\{\tau'/\alpha\}$ and
  $\mathcal{E};\Delta;\Gamma\{\tau'/\alpha\} \vdash e_{\forall}\{\tau'/\alpha\} \;:$
  $\quad(\forall^{+}\chi.\,\forall\alpha{:}\chi \rightarrow \Omega.\,\tau_0\,(\forall\,[\chi]\,\alpha))\{\tau'/\alpha\}$ and
  $\mathcal{E};\Delta;\Gamma\{\tau'/\alpha\} \vdash e_{\forall^{+}}\{\tau'/\alpha\} \;:$
  $\quad(\forall\alpha{:}\forall\chi.\,\Omega.\,\tau_0\,(\forall^{+}\alpha))\{\tau'/\alpha\}$

  The above typing judgments are equivalent to

  $\mathcal{E};\Delta \vdash \tau_0\{\tau'/\alpha\} \;:\; \Omega \rightarrow \Omega$ and
  $\mathcal{E};\Delta \vdash \tau_1\{\tau'/\alpha\} \;:\; \Omega$ and
  $\mathcal{E};\Delta;\Gamma\{\tau'/\alpha\} \vdash e_{\mathsf{int}}\{\tau'/\alpha\} \;:\; (\tau_0\{\tau'/\alpha\})\;\mathsf{int}$ and
  $\mathcal{E};\Delta;\Gamma\{\tau'/\alpha\} \vdash e_{\rightarrow}\{\tau'/\alpha\} \;:$
  $\quad\forall\alpha_1{:}\Omega.\,\forall\alpha_2{:}\Omega.\,(\tau_0\{\tau'/\alpha\})\,(\alpha_1 \rightarrow \alpha_2)$ and
  $\mathcal{E};\Delta;\Gamma\{\tau'/\alpha\} \vdash e_{\forall}\{\tau'/\alpha\} \;:$
  $\quad\forall^{+}\chi.\,\forall\alpha{:}\chi \rightarrow \Omega.\,(\tau_0\{\tau'/\alpha\})\,(\forall\,[\chi]\,\alpha)$ and
  $\mathcal{E};\Delta;\Gamma\{\tau'/\alpha\} \vdash e_{\forall^{+}}\{\tau'/\alpha\} \;:$
  $\quad\forall\alpha{:}\forall\chi.\,\Omega.\,(\tau_0\{\tau'/\alpha\})\,(\forall^{+}\alpha)$

  from which the statement of the lemma follows directly. $\square$

Lemma B.10 *If $\mathcal{E};\Delta;\Gamma, x : \tau' \vdash e \;:\; \tau$ and $\mathcal{E};\Delta;\Gamma \vdash e' \;:\; \tau'$, then $\mathcal{E};\Delta;\Gamma \vdash e\{e'/x\} \;:\; \tau$.*

Proof Proved by induction over the structure of e. The different cases are proved similarly. We will show only two cases here.

- **case** $e = \Lambda\alpha{:}\kappa.\,v$: We have that

  $\mathcal{E};\Delta;\Gamma, x{:}\tau' \vdash \Lambda\alpha{:}\kappa.\,v \;:\; \forall\alpha{:}\kappa.\,\tau$ and
  $\mathcal{E};\Delta;\Gamma \vdash e' \;:\; \tau'$

  Since e can always be alpha-converted, we assume that α is not previously defined in Δ. This implies $\mathcal{E};\Delta,\alpha : \kappa;\Gamma, x : \tau' \vdash v \;:\; \tau$. Since α is not free in e', we have $\mathcal{E};\Delta,\alpha{:}\kappa;\Gamma \vdash e' \;:\; \tau'$. By induction, $\mathcal{E};\Delta,\alpha{:}\kappa;\Gamma \vdash v\{e'/x\} \;:\; \tau$. Hence $\mathcal{E};\Delta;\Gamma \vdash \Lambda\alpha{:}\kappa.\,v\{e'/x\} \;:\; \forall\alpha{:}\kappa.\,\tau$.

- **case** $e = \mathsf{typecase}[\tau_0]\;\tau_1\;\mathsf{of}\;(e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}})$: We have that

  $\mathcal{E};\Delta;\Gamma \vdash e' \;:\; \tau'$ and
  $\mathcal{E};\Delta;\Gamma, x{:}\tau' \vdash \mathsf{typecase}[\tau_0]\;\tau_1\;\mathsf{of}\;(e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}}) \;:$
  $\quad\tau_0\,\tau_1$

  By the typecase typing rule we get

  $\mathcal{E};\Delta \vdash \tau_0 \;:\; \Omega \rightarrow \Omega$ and
  $\mathcal{E};\Delta \vdash \tau_1 \;:\; \Omega$ and
  $\mathcal{E};\Delta;\Gamma, x{:}\tau' \vdash e_{\mathsf{int}} \;:\; \tau_0\;\mathsf{int}$ and
  $\mathcal{E};\Delta;\Gamma, x{:}\tau' \vdash e_{\rightarrow} \;:\; \forall\alpha_1{:}\Omega.\,\forall\alpha_2{:}\Omega.\,\tau_0\,(\alpha_1 \rightarrow \alpha_2)$ and
  $\mathcal{E};\Delta;\Gamma, x{:}\tau' \vdash e_{\forall} \;:\; \forall^{+}\chi.\,\forall\alpha{:}\chi \rightarrow \Omega.\,\tau_0\,(\forall\,[\chi]\,\alpha)$ and
  $\mathcal{E};\Delta;\Gamma, x{:}\tau' \vdash e_{\forall^{+}} \;:\; \forall\alpha{:}\forall\chi.\,\Omega.\,\tau_0\,(\forall^{+}\alpha)$

$$
\begin{array}{lll}
(\textit{kinds}) & \kappa ::= & \Omega \mid \kappa \rightarrow \kappa' \mid \chi \mid \forall\chi.\,\kappa \\[1ex]
(\textit{types}) & \tau ::= & \mathsf{int} \mid \twoheadrightarrow \mid \forall \mid \forall^{+} \\
 & & \mid\; \alpha \mid \Lambda\chi.\,\tau \mid \lambda\alpha{:}\kappa.\,\tau \mid \tau\,[\kappa] \mid \tau\,\tau' \\
 & & \mid\; \mathsf{Typerec}[\kappa]\;\tau\;\mathsf{of}\;(\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^{+}})
\end{array}
$$

Figure 25: The λ_i^P type language

Applying the inductive hypothesis to each of the subterms $e_{\mathsf{int}}, e_{\rightarrow}, e_{\forall}, e_{\forall^{+}}$ yields directly the claim. $\square$

Definition B.11 *e evaluates to e' (written $e \mapsto e'$) if there exist E, e_1, and e_2 such that $e = E\,[e_1]$ and $e' = E\,[e_2]$ and $e_1 \rightsquigarrow e_2$.*

Theorem B.12 (Subject reduction) *If $\vdash e : \tau$ and $e \mapsto e'$, then $\vdash e' {:} \tau$.*

Proof By Lemma B.2, e can be decomposed into unique E and unique redex e_1 such that $e = E\,[e_1]$. By definition, $e' = E\,[e_2]$ and $e_1 \rightsquigarrow e_2$. By Lemma B.4, there exists a τ' such that $\vdash e_1 {:} \tau'$. By the same lemma, all we need to prove is that $\vdash e_2 : \tau'$ holds. This is proved by considering each possible redex in turn. We will show only two cases, the rest follow similarly.

- **case** $e_1 = (\mathsf{fix}\,x{:}\tau_1.\,v)\;v'$: Then $e_2 = (v\{\mathsf{fix}\,x{:}\tau_1.\,v/x\})\;v'$. We have that $\vdash (\mathsf{fix}\,x : \tau_1.\,v)\;v' : \tau'$. By the typing rules for term application we get that for some τ_2,

  $\vdash \mathsf{fix}\,x{:}\tau_1.\,v{:}\tau_2 \rightarrow \tau'$ and
  $\vdash v'{:}\tau_2$

  By the typing rule for fix we get that,

  $\vdash \tau_1 = \tau_2 \rightarrow \tau'$ and
  $\varepsilon;\varepsilon;\varepsilon, x{:}\tau_2 \rightarrow \tau' \vdash v \;:\; \tau_2 \rightarrow \tau'$

  Using Lemma B.10 and the typing rule for application, we obtain the desired judgment

  $\vdash (v\{\mathsf{fix}\,x{:}\tau_1.\,v/x\})\;v'{:}\tau'$

- **case** $e_1 = \mathsf{typecase}[\tau_0]\;\tau_1\;\mathsf{of}\;(e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}})$: If τ_1 is not in normal form, the reduction is to $e_2 = \mathsf{typecase}[\tau_0]\;\nu_1\;\mathsf{of}\;(e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}})$, where $\varepsilon;\varepsilon \vdash \tau_1 \mapsto^{*} \nu_1 : \Omega$. The latter implies $\varepsilon;\varepsilon \vdash \tau_0\,\tau_1 = \tau_0\,\nu_1 : \Omega$, hence $\vdash e_2 {:} \tau'$ follows directly from $\vdash e_1 {:} \tau'$.

  If τ_1 is in normal form ν_1, by the second premise of the typing rule for typecase and Lemma B.1 we have four cases for ν_1. In each case the contraction has the desired type $\tau_0\,\nu_1$, according to the corresponding premises of the typecase typing rule and the rules for type and kind applications. $\square$

B.2 Strong normalisation

The type language is shown in Figure 25. The single step reduction relation ($\tau \rightsquigarrow \tau'$) is shown in Figure 27.

Lemma B.13 *If $\mathcal{E};\Delta \vdash \tau \;:\; \kappa$ and $\tau \rightsquigarrow \tau'$, then $\mathcal{E};\Delta \vdash \tau' \;:\; \kappa$.*

Proof (Sketch) The proof follows from a case analysis of the reduction relation ($\rightsquigarrow$). $\square$

Lemma B.14 *If $\tau_1 \rightsquigarrow \tau_2$, then $\tau_1\{\tau/\alpha\} \rightsquigarrow \tau_2\{\tau/\alpha\}$.*

Proof The proof is by enumerating each possible reduction from τ_1 to τ_2.

case β_1: In this case, $\tau_1 = (\lambda\beta{:}\kappa.\,\tau')\,\tau''$ and $\tau_2 = \tau'\{\tau''/\beta\}$. This implies that

$$\tau_1\{\tau/\alpha\} = (\lambda\beta{:}\kappa.\,\tau'\{\tau/\alpha\})\,\tau''\{\tau/\alpha\}$$

This beta reduces to

$$(\tau'\{\tau/\alpha\})\{\tau''\{\tau/\alpha\}/\beta\}$$

Since β does not occur free in τ, this is equivalent to

$$(\tau'\{\tau''/\beta\})\{\tau/\alpha\}$$

case β_2: In this case, $\tau_1 = (\Lambda\chi.\,\tau')\,[\kappa]$ and $\tau_2 = \tau'\{\kappa/\chi\}$. We get that

$$\tau_1\{\tau/\alpha\} = (\Lambda\chi.\,\tau'\{\tau/\alpha\})\,[\kappa]$$

This beta reduces to

$$\tau'\{\tau/\alpha\}\{\kappa/\chi\}$$

Since χ is not free in τ, this is equivalent to

$$(\tau'\{\kappa/\chi\})\{\tau/\alpha\}$$

case η_1: In this case, $\tau_1 = \lambda\beta{:}\kappa.\,\tau'\,\beta$ and $\tau_2 = \tau'$ and β does not occur free in τ'. We get that

$$\tau_1\{\tau/\alpha\} = \lambda\beta{:}\kappa.\,(\tau'\{\tau/\alpha\})\,\beta$$

Since this is a capture avoiding substitution, β still does not occur free in $\tau'\{\tau/\alpha\}$. Therefore this eta reduces to $\tau'\{\tau/\alpha\}$.

case η_2: In this case, $\tau_1 = \Lambda\chi.\,\tau'\,[\chi]$ and $\tau_2 = \tau'$ and χ does not occur free in τ'. We get that

$$\tau_1\{\tau/\alpha\} = \Lambda\chi.\,(\tau'\{\tau/\alpha\})\,[\chi]$$

Since this is a capture avoiding substitution, χ still does not occur free in $\tau'\{\tau/\alpha\}$. Therefore, this eta reduces to $\tau'\{\tau/\alpha\}$.

case t_1: $\tau_1 = \mathsf{Typerec}[\kappa]\ \mathsf{int}\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ and $\tau_2 = \tau_{\mathsf{int}}$. We get that

$$\begin{array}{l}\tau_1\{\tau/\alpha\} = \\ \mathsf{Typerec}[\kappa]\ \mathsf{int}\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\rightarrow}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\})\end{array}$$

But this reduces by the t_1 reduction to $\tau_{\mathsf{int}}\{\tau/\alpha\}$.

case t_2: $\tau_1 = \mathsf{Typerec}[\kappa]\ (\tau' \rightarrow \tau'')\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ and

$$\begin{array}{l}\tau_2 = \tau_{\rightarrow}\,\tau'\,\tau''\,(\mathsf{Typerec}[\kappa]\ \tau'\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})) \\ \qquad\qquad (\mathsf{Typerec}[\kappa]\ \tau''\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+}))\end{array}$$

We get that

$$\begin{array}{l}\tau_1\{\tau/\alpha\} = \\ \mathsf{Typerec}[\kappa]\ (\tau'\{\tau/\alpha\} \rightarrow \tau''\{\tau/\alpha\})\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\rightarrow}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\})\end{array}$$

This reduces by t_2 to

$$\begin{array}{l}\tau_{\rightarrow}\{\tau/\alpha\}\,(\tau'\{\tau/\alpha\})\,(\tau''\{\tau/\alpha\}) \\ \ (\mathsf{Typerec}[\kappa]\ (\tau'\{\tau/\alpha\})\ \mathsf{of} \\ \qquad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\rightarrow}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\})) \\ \ (\mathsf{Typerec}[\kappa]\ (\tau''\{\tau/\alpha\})\ \mathsf{of} \\ \qquad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\rightarrow}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\}))\end{array}$$

But this is syntactically equal to $\tau_2\{\tau/\alpha\}$.

case t_3: $\tau_1 = \mathsf{Typerec}[\kappa]\ (\forall\,[\kappa']\,\tau')\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ and

$$\tau_2 = \tau_{\forall}\,[\kappa']\,\tau'\,(\lambda\beta{:}\kappa'.\,\mathsf{Typerec}[\kappa]\ (\tau'\,\beta)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+}))$$

We get that

$$\begin{array}{l}\tau_1\{\tau/\alpha\} = \\ \mathsf{Typerec}[\kappa]\ (\forall\,[\kappa']\,\tau'\{\tau/\alpha\})\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\rightarrow}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\})\end{array}$$

This reduces by t_3 to

$$\begin{array}{l}\tau_{\forall}\{\tau/\alpha\}\,[\kappa']\,(\tau'\{\tau/\alpha\}) \\ \ (\lambda\beta{:}\kappa'.\,\mathsf{Typerec}[\kappa]\ ((\tau'\{\tau/\alpha\})\,\beta)\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\rightarrow}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\}))\end{array}$$

But this is syntactically equivalent to $\tau_2\{\tau/\alpha\}$.

case t_4: $\tau_1 = \mathsf{Typerec}[\kappa]\ (\forall^+\tau')\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ and

$$\tau_2 = \tau_{\forall^+}\,\tau'\,(\Lambda\chi.\,\mathsf{Typerec}[\kappa]\ (\tau'\,[\chi])\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+}))$$

We get that

$$\begin{array}{l}\tau_1\{\tau/\alpha\} = \\ \mathsf{Typerec}[\kappa]\ (\forall^+\tau'\{\tau/\alpha\})\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\rightarrow}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\})\end{array}$$

This reduces by t_4 to

$$\begin{array}{l}\tau_{\forall^+}\{\tau/\alpha\}\,(\tau'\{\tau/\alpha\}) \\ \ (\Lambda\chi.\,\mathsf{Typerec}[\kappa]\ ((\tau'\{\tau/\alpha\})\,[\chi])\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\rightarrow}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\}))\end{array}$$

But this is syntactically equal to $\tau_2\{\tau/\alpha\}$. □

Lemma B.15 *If* $\tau_1 \leadsto \tau_2$*, then* $\tau_1\{\kappa'/\chi'\} \leadsto \tau_2\{\kappa'/\chi'\}$.

Proof This is proved by case analysis of the type reduction relation.

case β_1: In this case, $\tau_1 = (\lambda\beta{:}\kappa.\,\tau')\,\tau''$ and $\tau_2 = \tau'\{\tau''/\beta\}$. This implies that

$$\tau_1\{\kappa'/\chi'\} = (\lambda\beta{:}\kappa\{\kappa'/\chi'\}.\,\tau'\{\kappa'/\chi'\})\,\tau''\{\kappa'/\chi'\}$$

This beta reduces to

$$(\tau'\{\kappa'/\chi'\})\{\tau''\{\kappa'/\chi'\}/\beta\}$$

But this is equivalent to

$$(\tau'\{\tau''/\beta\})\{\kappa'/\chi'\}$$

case β_2: In this case, $\tau_1 = (\Lambda\chi.\,\tau')\,[\kappa]$ and $\tau_2 = \tau'\{\kappa/\chi\}$. We get that

$$\tau_1\{\kappa'/\chi'\} = (\Lambda\chi.\,\tau'\{\kappa'/\chi'\})\,[\kappa\{\kappa'/\chi'\}]$$

This beta reduces to

$$\tau'\{\kappa'/\chi'\}\{\kappa\{\kappa'/\chi'\}/\chi\}$$

Since χ is not free in κ', this is equivalent to

$$(\tau'\{\kappa/\chi\})\{\kappa'/\chi'\}$$

case η_1: In this case, $\tau_1 = \lambda\beta{:}\kappa.\,\tau'\,\beta$ and $\tau_2 = \tau'$ and β does not occur free in τ'. We get that

$$\tau_1\{\kappa'/\chi'\} = \lambda\beta{:}\kappa\{\kappa'/\chi'\}.\,(\tau'\{\kappa'/\chi'\})\,\beta$$

Again β does not occur free in $\tau'\{\kappa'/\chi'\}$. Therefore this eta reduces to $\tau'\{\kappa'/\chi'\}$.

case η_2: In this case, $\tau_1 = \Lambda\chi.\, \tau'\,[\chi]$ and $\tau_2 = \tau'$ and χ does not occur free in τ'. We get that

$$\tau_1\{\kappa'/\chi'\} = \Lambda\chi.\, (\tau'\{\kappa'/\chi'\})\,[\chi]$$

Since this is a capture avoiding substitution, χ still does not occur free in $\tau'\{\kappa'/\chi'\}$. Therefore, this eta reduces to $\tau'\{\kappa'/\chi'\}$.

case t_1: $\tau_1 = \mathsf{Typerec}[\kappa]\ \mathsf{int}\ \mathsf{of}\ (\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+})$ and $\tau_2 = \tau_{\mathsf{int}}$. We get that

$$\begin{aligned}&\tau_1\{\kappa'/\chi'\} = \\ &\mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ \mathsf{int}\ \mathsf{of} \\ &\quad (\tau_{\mathsf{int}}\{\kappa'/\chi'\};\, \tau_{\rightarrow}\{\kappa'/\chi'\};\, \tau_{\forall}\{\kappa'/\chi'\};\, \tau_{\forall^+}\{\kappa'/\chi'\})\end{aligned}$$

But this reduces by the t_1 reduction to $\tau_{\mathsf{int}}\{\kappa'/\chi'\}$.

case t_2: $\tau_1 = \mathsf{Typerec}[\kappa]\ (\tau' \rightarrow \tau'')\ \mathsf{of}\ (\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+})$ and

$$\begin{aligned}\tau_2 = \tau_{\rightarrow}\, \tau'\, \tau''\, &(\mathsf{Typerec}[\kappa]\ \tau'\ \mathsf{of}\ (\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+})) \\ &(\mathsf{Typerec}[\kappa]\ \tau''\ \mathsf{of}\ (\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+}))\end{aligned}$$

We get that

$$\begin{aligned}\tau_1\{\kappa'/\chi'\} = &\ \mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ (\tau'\{\kappa'/\chi'\} \rightarrow \tau''\{\kappa'/\chi'\})\ \mathsf{of} \\ &(\tau_{\mathsf{int}}\{\kappa'/\chi'\};\, \tau_{\rightarrow}\{\kappa'/\chi'\};\, \tau_{\forall}\{\kappa'/\chi'\};\, \tau_{\forall^+}\{\kappa'/\chi'\})\end{aligned}$$

This reduces by t_2 to

$$\begin{aligned}&\tau_{\rightarrow}\{\kappa'/\chi'\}\, (\tau'\{\kappa'/\chi'\})\, (\tau''\{\kappa'/\chi'\}) \\ &\ (\mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ (\tau'\{\kappa'/\chi'\})\ \mathsf{of} \\ &\quad\ (\tau_{\mathsf{int}}\{\kappa'/\chi'\};\, \tau_{\rightarrow}\{\kappa'/\chi'\};\, \tau_{\forall}\{\kappa'/\chi'\};\, \tau_{\forall^+}\{\kappa'/\chi'\})) \\ &\ (\mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ (\tau''\{\kappa'/\chi'\})\ \mathsf{of} \\ &\quad\ (\tau_{\mathsf{int}}\{\kappa'/\chi'\};\, \tau_{\rightarrow}\{\kappa'/\chi'\};\, \tau_{\forall}\{\kappa'/\chi'\};\, \tau_{\forall^+}\{\kappa'/\chi'\}))\end{aligned}$$

But this is syntactically equal to $\tau_2\{\kappa'/\chi'\}$.

case t_3: $\tau_1 = \mathsf{Typerec}[\kappa]\ (\forall\,[\kappa_1]\,\tau')\ \mathsf{of}\ (\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+})$ and

$$\tau_2 = \tau_{\forall}\,[\kappa_1]\,\tau'\,(\lambda\beta\!:\!\kappa_1.\, \mathsf{Typerec}[\kappa]\ (\tau'\,\beta)\ \mathsf{of}\ (\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+}))$$

We get that

$$\begin{aligned}&\tau_1\{\kappa'/\chi'\} = \\ &\mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ (\forall\,[\kappa_1\{\kappa'/\chi'\}]\,\tau'\{\kappa'/\chi'\})\ \mathsf{of} \\ &\quad (\tau_{\mathsf{int}}\{\kappa'/\chi'\};\, \tau_{\rightarrow}\{\kappa'/\chi'\};\, \tau_{\forall}\{\kappa'/\chi'\};\, \tau_{\forall^+}\{\kappa'/\chi'\})\end{aligned}$$

This reduces by t_3 to

$$\begin{aligned}&\tau_{\forall}\{\kappa'/\chi'\}\,[\kappa_1\{\kappa'/\chi'\}]\,(\tau'\{\kappa'/\chi'\}) \\ &\ (\lambda\beta\!:\!\kappa_1\{\kappa'/\chi'\}.\, \mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ ((\tau'\{\kappa'/\chi'\})\,\beta)\ \mathsf{of} \\ &\ (\tau_{\mathsf{int}}\{\kappa'/\chi'\};\, \tau_{\rightarrow}\{\kappa'/\chi'\};\, \tau_{\forall}\{\kappa'/\chi'\};\, \tau_{\forall^+}\{\kappa'/\chi'\}))\end{aligned}$$

But this is syntactically equivalent to $\tau_2\{\kappa'/\chi'\}$.

case t_4: $\tau_1 = \mathsf{Typerec}[\kappa]\ (\forall^+\tau')\ \mathsf{of}\ (\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+})$ and

$$\tau_2 = \tau_{\forall^+}\,\tau'\,(\Lambda\chi.\, \mathsf{Typerec}[\kappa]\ (\tau'\,[\chi])\ \mathsf{of}\ (\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+}))$$

We get that

$$\begin{aligned}&\tau_1\{\kappa'/\chi'\} = \\ &\mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ (\forall^+\tau'\{\kappa'/\chi'\})\ \mathsf{of} \\ &\quad (\tau_{\mathsf{int}}\{\kappa'/\chi'\};\, \tau_{\rightarrow}\{\kappa'/\chi'\};\, \tau_{\forall}\{\kappa'/\chi'\};\, \tau_{\forall^+}\{\kappa'/\chi'\})\end{aligned}$$

This reduces by t_4 to

$$\begin{aligned}&\tau_{\forall^+}\{\kappa'/\chi'\}\,(\tau'\{\kappa'/\chi'\}) \\ &\ (\Lambda\chi.\, \mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ ((\tau'\{\kappa'/\chi'\})\,[\chi])\ \mathsf{of} \\ &\quad (\tau_{\mathsf{int}}\{\kappa'/\chi'\};\, \tau_{\rightarrow}\{\kappa'/\chi'\};\, \tau_{\forall}\{\kappa'/\chi'\};\, \tau_{\forall^+}\{\kappa'/\chi'\}))\end{aligned}$$

But this is syntactically equal to $\tau_2\{\kappa'/\chi'\}$. □

Definition B.16 *A type τ is strongly normalisable if every reduction sequence from τ terminates into a normal form (with no redexes). We use $\nu(\tau)$ to denote the length of the largest reduction sequence from τ to a normal form.*

Definition B.17 *We define neutral types, n, as*

$$\begin{aligned}n_0 &::= \Lambda\chi.\, \tau \mid \lambda\alpha\!:\!\kappa.\, \tau \\ n &::= \alpha \mid n_0\, \tau \mid n\, \tau \mid n_0\, [\kappa] \mid n\, [\kappa] \\ &\quad\mid \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+})\end{aligned}$$

Definition B.18 *A reducibility candidate (also referred to as a candidate) of kind κ is a set $\mathcal{C}$ of types of kind κ such that*

1. *if $\tau \in \mathcal{C}$, then τ is strongly normalisable.*
2. *if $\tau \in \mathcal{C}$ and $\tau \rightsquigarrow \tau'$, then $\tau' \in \mathcal{C}$.*
3. *if τ is neutral and if for all τ' such that $\tau \rightsquigarrow \tau'$, we have that $\tau' \in \mathcal{C}$, then $\tau \in \mathcal{C}$.*

This implies that the candidates are never empty since if α has kind κ, then α belongs to candidates of kind κ.

Definition B.19 *Let κ be an arbitrary kind. Let $\mathcal{C}_\kappa$ be a candidate of kind κ. Let $\mathcal{C}_{\Omega\rightarrow\Omega\rightarrow\kappa\rightarrow\kappa\rightarrow\kappa}$ be a candidate of kind $\Omega \rightarrow \Omega \rightarrow \kappa \rightarrow \kappa \rightarrow \kappa$. Let $\mathcal{C}_{\forall\chi.\,(\chi\rightarrow\Omega)\rightarrow(\chi\rightarrow\kappa)\rightarrow\kappa}$ be a candidate of kind $\forall\chi.\,(\chi \rightarrow \Omega) \rightarrow (\chi \rightarrow \kappa) \rightarrow \kappa$. Let $\mathcal{C}_{(\forall\chi.\,\Omega)\rightarrow(\forall\chi.\,\kappa)\rightarrow\kappa}$ be a candidate of kind $(\forall\chi.\,\Omega) \rightarrow (\forall\chi.\,\kappa) \rightarrow \kappa$. We then define the set R_Ω of types of kind Ω as*

$$\begin{aligned}&\tau \in R_\Omega \quad \textit{iff} \\ &\quad \forall\tau_{\mathsf{int}} \in \mathcal{C}_\kappa \\ &\quad \forall\tau_{\rightarrow} \in \mathcal{C}_{\Omega\rightarrow\Omega\rightarrow\kappa\rightarrow\kappa\rightarrow\kappa}, \\ &\quad \forall\tau_{\forall} \in \mathcal{C}_{\forall\chi.\,(\chi\rightarrow\Omega)\rightarrow(\chi\rightarrow\kappa)\rightarrow\kappa}, \\ &\quad \forall\tau_{\forall^+} \in \mathcal{C}_{(\forall\chi.\,\Omega)\rightarrow(\forall\chi.\,\kappa)\rightarrow\kappa} \\ &\qquad \Rightarrow \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+}) \in \mathcal{C}_\kappa\end{aligned}$$

Lemma B.20 *R_Ω is a candidate of kind Ω.*

Proof Suppose $\tau \in R_\Omega$. Suppose τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, and $\tau_{\forall^+}$ belong to $\mathcal{C}_\kappa$, $\mathcal{C}_{\Omega\rightarrow\Omega\rightarrow\kappa\rightarrow\kappa\rightarrow\kappa}$, $\mathcal{C}_{\forall\chi.\,(\chi\rightarrow\Omega)\rightarrow(\chi\rightarrow\kappa)\rightarrow\kappa}$, $\mathcal{C}_{(\forall\chi.\,\Omega)\rightarrow(\forall\chi.\,\kappa)\rightarrow\kappa}$ respectively, where the candidates are of the appropriate kinds (see definition B.19).

Consider $\tau' = \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+})$. By definition this belongs to $\mathcal{C}_\kappa$. By property 1 of definition B.18, τ' is strongly normalisable and therefore τ must be strongly normalisable.

Consider $\tau' = \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+})$. Suppose $\tau \rightsquigarrow \tau_1$. Then $\tau' \rightsquigarrow \mathsf{Typerec}[\kappa]\ \tau_1\ \mathsf{of}\ (\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+})$. Since $\tau' \in \mathcal{C}_\kappa$, $\mathsf{Typerec}[\kappa]\ \tau_1\ \mathsf{of}\ (\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+})$ belongs to $\mathcal{C}_\kappa$ by property 2 of definition B.18. Therefore, by definition, τ_1 belongs to R_Ω.

Suppose τ is neutral and for all τ_1 such that $\tau \rightsquigarrow \tau_1$, $\tau_1 \in R_\Omega$. Consider $\tau' = \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+})$. Since we know that τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, and $\tau_{\forall^+}$ are strongly normalisable, we can induct over $len = \nu(\tau_{\mathsf{int}}) + \nu(\tau_{\rightarrow}) + \nu(\tau_{\forall}) + \nu(\tau_{\forall^+})$. We will prove that for all values of len, $\mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+})$ always reduces to a type that belongs to $\mathcal{C}_\kappa$; given that τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, and $\tau_{\forall^+}$ belong to $\mathcal{C}_\kappa$, $\mathcal{C}_{\Omega\rightarrow\Omega\rightarrow\kappa\rightarrow\kappa\rightarrow\kappa}$, $\mathcal{C}_{\forall\chi.\,(\chi\rightarrow\Omega)\rightarrow(\chi\rightarrow\kappa)\rightarrow\kappa}$, and $\mathcal{C}_{(\forall\chi.\,\Omega)\rightarrow(\forall\chi.\,\kappa)\rightarrow\kappa}$ respectively (see definition B.19).

- $len = 0$ Then $\tau' \rightsquigarrow \mathsf{Typerec}[\kappa]\ \tau_1\ \mathsf{of}\ (\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+})$ is the only possible reduction since τ is neutral. By the assumption on τ_1, this belongs to $\mathcal{C}_\kappa$.

- $len = k+1$ For the inductive case, assume that the hypothesis is true for $len = k$. That is, for $len = k$, $\mathsf{Typerec}[\kappa]\ \tau$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ always reduces to a type that belongs to $\mathcal{C}_\kappa$; given that τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, and $\tau_{\forall^+}$ belong to $\mathcal{C}_\kappa$, $\mathcal{C}_{\Omega\rightarrow\Omega\rightarrow\kappa\rightarrow\kappa\rightarrow\kappa}$, $\mathcal{C}_{\forall\chi.\,(\chi\rightarrow\Omega)\rightarrow(\chi\rightarrow\kappa)\rightarrow\kappa}$, and $\mathcal{C}_{(\forall\chi.\,\Omega)\rightarrow(\forall\chi.\,\kappa)\rightarrow\kappa}$ respectively. This implies that for $len = k$, $\mathsf{Typerec}[\kappa]\ \tau$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ belongs to $\mathcal{C}_\kappa$ (by property 3 of definition B.18). For $len = k+1$, consider $\tau' = \mathsf{Typerec}[\kappa]\ \tau$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$. This can reduce to $\mathsf{Typerec}[\kappa]\ \tau_1$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ which belongs to $\mathcal{C}_\kappa$. The other possible reductions are $\mathsf{Typerec}[\kappa]\ \tau$ of $(\tau'_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ where $\tau_{\mathsf{int}} \rightsquigarrow \tau'_{\mathsf{int}}$, or $\mathsf{Typerec}[\kappa]\ \tau$ of $(\tau_{\mathsf{int}};\ \tau'_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ where $\tau_{\rightarrow} \rightsquigarrow \tau'_{\rightarrow}$, or $\mathsf{Typerec}[\kappa]\ \tau$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau'_{\forall};\ \tau_{\forall^+})$ where $\tau_{\forall} \rightsquigarrow \tau'_{\forall}$, or $\mathsf{Typerec}[\kappa]\ \tau$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau'_{\forall^+})$ where $\tau_{\forall^+} \rightsquigarrow \tau'_{\forall^+}$. By property 2 of definition B.18, each of τ'_{int}, $\tau'_{\rightarrow}$, $\tau'_{\forall}$, and $\tau'_{\forall^+}$ belongs to the required candidate and $len = k$ for each of the reducts. Therefore, by the inductive hypothesis, each of the reducts belongs to $\mathcal{C}_\kappa$.

Therefore $\mathsf{Typerec}[\kappa]\ \tau$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ always reduces to a type that belongs to $\mathcal{C}_\kappa$. By property 3 of definition B.18, $\mathsf{Typerec}[\kappa]\ \tau$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ also belongs to $\mathcal{C}_\kappa$. Therefore, $\tau \in R_\Omega$ □

Definition B.21 *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two candidates of kinds κ_1 and κ_2. We then define the set $\mathcal{C}_1 \rightarrow \mathcal{C}_2$, of types of kind $\kappa_1 \rightarrow \kappa_2$, as*

$$\tau \in \mathcal{C}_1 \rightarrow \mathcal{C}_2 \quad \textit{iff} \quad \forall \tau' (\tau' \in \mathcal{C}_1 \Rightarrow \tau\,\tau' \in \mathcal{C}_2)$$

Lemma B.22 *If $\mathcal{C}_1$ and $\mathcal{C}_2$ are candidates of kinds κ_1 and κ_2, then $\mathcal{C}_1 \rightarrow \mathcal{C}_2$ is a candidate of kind $\kappa_1 \rightarrow \kappa_2$.*

Proof Suppose τ of kind $\kappa_1 \rightarrow \kappa_2$ belongs to $\mathcal{C}_1 \rightarrow \mathcal{C}_2$. By definition, if $\tau' \in \mathcal{C}_1$, then $\tau\,\tau' \in \mathcal{C}_2$. Since $\mathcal{C}_2$ is a candidate, $\tau\,\tau'$ is strongly normalisable. Therefore, τ must be strongly normalisable since for every sequence of reductions $\tau \rightsquigarrow \tau_1 \ldots \tau_k \ldots$, there is a corresponding sequence of reductions $\tau\,\tau' \rightsquigarrow \tau_1\,\tau' \ldots \tau_k\,\tau' \ldots$.

Suppose τ of kind $\kappa_1 \rightarrow \kappa_2$ belongs to $\mathcal{C}_1 \rightarrow \mathcal{C}_2$ and $\tau \rightsquigarrow \tau'$. Suppose $\tau_1 \in \mathcal{C}_1$. By definition, $\tau\,\tau_1 \in \mathcal{C}_2$. But $\tau\,\tau_1 \rightsquigarrow \tau'\,\tau_1$. By using property 2 of definition B.18 on $\mathcal{C}_2$, $\tau'\,\tau_1 \in \mathcal{C}_2$; therefore, $\tau' \in \mathcal{C}_1 \rightarrow \mathcal{C}_2$.

Consider a neutral τ of kind $\kappa_1 \rightarrow \kappa_2$. Suppose that for all τ' such that $\tau \rightsquigarrow \tau'$, $\tau' \in \mathcal{C}_1 \rightarrow \mathcal{C}_2$. Consider $\tau\,\tau_1$ where $\tau_1 \in \mathcal{C}_1$. Since τ_1 is strongly normalisable, we can induct over $\nu(\tau_1)$. If $\nu(\tau_1) = 0$, then $\tau\,\tau_1 \rightsquigarrow \tau'\,\tau_1$. But $\tau'\,\tau_1 \in \mathcal{C}_2$ (by assumption on τ'), and since τ is neutral, no other reduction is possible. If $\nu(\tau_1) \neq 0$, then $\tau_1 \rightsquigarrow \tau'_1$. In this case, $\tau\,\tau_1$ may reduce to either $\tau'\,\tau_1$ or to $\tau\,\tau'_1$. We saw that the first reduct belongs to $\mathcal{C}_2$. By property 2 of definition B.18, $\tau'_1 \in \mathcal{C}_1$ and $\nu(\tau'_1) < \nu(\tau_1)$. By the inductive assumption over $\nu(\tau_1)$, we get that $\tau\,\tau'_1$ belongs to $\mathcal{C}_2$. By property 3 of definition B.18, $\tau\,\tau_1 \in \mathcal{C}_2$. This implies that $\tau \in \mathcal{C}_1 \rightarrow \mathcal{C}_2$. □

Definition B.23 *We use $\overline{\chi}$ to denote the set $\chi_1, \ldots, \chi_n$ of χ. We use a similar syntax to denote a set of other constructs.*

Definition B.24 *Let $\kappa[\overline{\chi}]$ be a kind where $\overline{\chi}$ contains all the free kind variables of κ. Let $\overline{\kappa}$ be a sequence of closed kinds of the same length and $\overline{\mathcal{C}}$ be a sequence of candidates of the corresponding kind. We now define the set $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$ of types of kind $\kappa\{\overline{\kappa}/\overline{\chi}\}$ as*

1. *if $\kappa = \Omega$, then $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}] = R_\Omega$.*
2. *if $\kappa = \chi_i$, then $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}] = \mathcal{C}_i$.*
3. *if $\kappa = \kappa_1 \rightarrow \kappa_2$, then $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}] = \mathcal{S}_{\kappa_1}[\overline{\mathcal{C}}/\overline{\chi}] \rightarrow \mathcal{S}_{\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$.*
4. *if $\kappa = \forall\chi.\,\kappa'$, then $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$ = the set of types τ of kind $\kappa\{\overline{\kappa}/\overline{\chi}\}$ such that for every kind κ'' and reducibility candidate $\mathcal{C}''$ of this kind, $\tau\,[\kappa''] \in \mathcal{S}_{\kappa'}[\overline{\mathcal{C}}, \mathcal{C}''/\overline{\chi}, \chi]$.*

Lemma B.25 *$\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$ is a reducibility candidate of kind $\kappa\{\overline{\kappa}/\overline{\chi}\}$.*

Proof For $\kappa = \Omega$, the lemma follows from lemma B.20. For $\kappa = \chi$, the lemma follows by definition. If $\kappa = \kappa_1 \rightarrow \kappa_2$, then the lemma follows from the inductive hypothesis on κ_1 and κ_2 and lemma B.22. We only need to prove the case for $\kappa = \forall\chi'.\,\kappa'$. We will induct over the size of κ with the $\overline{\chi}$ containing all the free kind variables of κ.

Consider a $\tau \in \mathcal{S}_{\forall\chi'.\,\kappa'}[\overline{\mathcal{C}}/\overline{\chi}]$. By definition, for any kind κ_1 and corresponding candidate $\mathcal{C}'$, $\tau\,[\kappa_1] \in \mathcal{S}_{\kappa'}[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$. Applying the inductive hypothesis on κ', we get that $\mathcal{S}_{\kappa'}[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$ is a candidate. Therefore, $\tau\,[\kappa_1]$ is strongly normalisable which implies that τ is strongly normalisable.

Consider a $\tau \in \mathcal{S}_{\forall\chi'.\,\kappa'}[\overline{\mathcal{C}}/\overline{\chi}]$ and $\tau \rightsquigarrow \tau_1$. For any kind κ_1 and corresponding candidate $\mathcal{C}'$, by definition, $\tau\,[\kappa_1] \in \mathcal{S}_{\kappa'}[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$. But $\tau\,[\kappa_1] \rightsquigarrow \tau_1\,[\kappa_1]$. By the inductive hypothesis on κ', we get that $\mathcal{S}_{\kappa'}[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$ is a candidate. By property 2 of definition B.18, $\tau_1\,[\kappa_1] \in \mathcal{S}_{\kappa'}[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$. Therefore, $\tau_1 \in \mathcal{S}_{\forall\chi'.\,\kappa'}[\overline{\mathcal{C}}/\overline{\chi}]$.

Consider a neutral τ so that for all τ_1, such that $\tau \rightsquigarrow \tau_1$, $\tau_1 \in \mathcal{S}_{\forall\chi'.\,\kappa'}[\overline{\mathcal{C}}/\overline{\chi}]$. Consider $\tau\,[\kappa_1]$ for an arbitrary kind κ_1 and corresponding candidate $\mathcal{C}'$. We have that $\tau\,[\kappa_1] \rightsquigarrow \tau_1\,[\kappa_1]$. This is the only possible reduction since τ is neutral. By the assumption on τ_1 $\tau_1\,[\kappa_1] \in \mathcal{S}_{\kappa'}[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$. By the inductive hypothesis on κ', we get that $\mathcal{S}_{\kappa'}[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$ is a candidate. By property 3 of definition B.18, $\tau\,[\kappa_1] \in \mathcal{S}_{\kappa'}[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$. Therefore $\tau \in \mathcal{S}_{\forall\chi'.\,\kappa'}[\overline{\mathcal{C}}/\overline{\chi}]$. □

Lemma B.26 $\mathcal{S}_{\kappa\{\kappa'/\chi'\}}[\overline{\mathcal{C}}/\overline{\chi}] = \mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{S}_{\kappa'}[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi']$

Proof The proof is by induction over the structure of κ. We will show only the case for polymorphic kinds, the others follow directly by induction. Suppose $\kappa = \forall\chi''.\,\kappa''$. Then the LHS is the set of types τ of kind $(\forall\chi''.\,\kappa''\{\kappa'/\chi'\})\{\overline{\kappa}/\overline{\chi}\}$ such that for every kind κ''' and corresponding candidate $\mathcal{C}'''$, $\tau\,[\kappa''']$ belongs to $\mathcal{S}_{\kappa''\{\kappa'/\chi'\}}[\overline{\mathcal{C}}, \mathcal{C}'''/\overline{\chi}, \chi'']$. Applying the inductive hypothesis to κ'', this is equal to $\mathcal{S}_{\kappa''}[\overline{\mathcal{C}}, \mathcal{C}''', \mathcal{S}_{\kappa'}[\overline{\mathcal{C}}, \mathcal{C}'''/\overline{\chi}, \chi'']/\overline{\chi}, \chi'', \chi']$. But χ'' does not occur free in κ' (variables in κ' can always be renamed). Therefore, $\tau\,[\kappa''']$ belongs to $\mathcal{S}_{\kappa''}[\overline{\mathcal{C}}, \mathcal{C}''', \mathcal{S}_{\kappa'}[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi'', \chi']$. The RHS consists of types τ' of kind $(\forall\chi''.\,\kappa'')\{\overline{\kappa}, \kappa'\{\overline{\kappa}/\overline{\chi}\}/\overline{\chi}, \chi'\}$ such that for every kind κ''' and corresponding candidate $\mathcal{C}'''$, $\tau'\,[\kappa''']$ belongs to $\mathcal{S}_{\kappa''}[\overline{\mathcal{C}}, \mathcal{S}_{\kappa'}[\overline{\mathcal{C}}/\overline{\chi}], \mathcal{C}'''/\overline{\chi}, \chi', \chi'']$. Also, the kind of τ' is equivalent to $(\forall\chi''.\,\kappa''\{\kappa'/\chi'\})\{\overline{\kappa}/\overline{\chi}\}$. □

Proposition B.27 *From lemma B.25, we know that $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$ is a candidate of kind $\kappa\{\overline{\kappa}/\overline{\chi}\}$, that $\mathcal{S}_{\Omega\rightarrow\Omega\rightarrow\kappa\rightarrow\kappa\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$ is a candidate of kind $(\Omega \rightarrow \Omega \rightarrow \kappa \rightarrow \kappa \rightarrow \kappa)\{\overline{\kappa}/\overline{\chi}\}$, that $\mathcal{S}_{\forall\chi.\,(\chi\rightarrow\Omega)\rightarrow(\chi\rightarrow\kappa)\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$ is a candidate of kind $(\forall\chi.\,(\chi \rightarrow \Omega) \rightarrow (\chi \rightarrow \kappa) \rightarrow \kappa)\{\overline{\kappa}/\overline{\chi}\}$, and $\mathcal{S}_{(\forall\chi.\,\Omega)\rightarrow(\forall\chi.\,\kappa)\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$ is a candidate of kind $((\forall\chi.\,\Omega) \rightarrow (\forall\chi.\,\kappa) \rightarrow \kappa)\{\overline{\kappa}/\overline{\chi}\}$. In the rest of the section, we will assume that the types τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, and $\tau_{\forall^+}$ belong to the above candidates respectively.*

Lemma B.28 $\mathsf{int} \in R_\Omega = \mathcal{S}_\Omega[\overline{\mathcal{C}}/\overline{\chi}]$

Proof Consider $\tau = \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \mathsf{int}\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$. The lemma holds if $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \mathsf{int}\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\kappa}]$ is true; given that $\tau_{\mathsf{int}} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\rightarrow} \in \mathcal{S}_{\Omega\rightarrow\Omega\rightarrow\kappa\rightarrow\kappa\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall} \in \mathcal{S}_{\forall\chi.\,(\chi\rightarrow\Omega)\rightarrow(\chi\rightarrow\kappa)\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall^+} \in \mathcal{S}_{(\forall\chi.\,\Omega)\rightarrow(\forall\chi.\,\kappa)\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$.

Since τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, and $\tau_{\forall^+}$ are strongly normalisable, we will induct over $len = \nu(\tau_{\mathsf{int}}) + \nu(\tau_{\rightarrow}) + \nu(\tau_{\forall}) + \nu(\tau_{\forall^+})$. We will prove that for all values of len, $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \mathsf{int}\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ always reduces to a type that belongs to $\mathcal{C}_\kappa$; given that the branches belong to the candidates as in proposition B.27.

- $len = 0$ Then $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \mathsf{int}\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ can reduce only to τ_{int} which by assumption belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\kappa}]$.
- $len = k + 1$ For the inductive case, assume that the hypothesis holds true for $len = k$. That is, for $len = k$, $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \mathsf{int}\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$; given that τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, and $\tau_{\forall^+}$ belong to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, $\mathcal{S}_{\Omega\rightarrow\Omega\rightarrow\kappa\rightarrow\kappa\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$, $\mathcal{S}_{\forall\chi.\,(\chi\rightarrow\Omega)\rightarrow(\chi\rightarrow\kappa)\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$, and to $\mathcal{S}_{(\forall\chi.\,\Omega)\rightarrow(\forall\chi.\,\kappa)\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$. This implies that for $len = k$, the type $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \mathsf{int}\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$ (by property 3 of definition B.18). For $len = k+1$, τ can reduce to τ_{int} which belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. The other possible reductions are to $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \mathsf{int}\ \mathsf{of}\ (\tau'_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ where $\tau_{\mathsf{int}} \leadsto \tau'_{\mathsf{int}}$, or to $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \mathsf{int}\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau'_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ where $\tau_{\rightarrow} \leadsto \tau'_{\rightarrow}$, or to $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \mathsf{int}\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau'_{\forall};\ \tau_{\forall^+})$ where $\tau_{\forall} \leadsto \tau'_{\forall}$, or to $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \mathsf{int}\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau'_{\forall^+})$ where $\tau_{\forall^+} \leadsto \tau'_{\forall^+}$. By property 2 of definition B.18, each of τ'_{int}, $\tau'_{\rightarrow}$, $\tau'_{\forall}$, $\tau'_{\forall^+}$ belongs to the same candidate. Moreover, $len = k$ for each of the reducts. Therefore, by the inductive hypothesis, each of the reducts belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.

Therefore, $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \mathsf{int}\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. By property 3 of definition B.18, $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \mathsf{int}\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ also belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. Therefore, $\mathsf{int} \in R_\Omega$. □

Lemma B.29 $\twoheadrightarrow \in R_\Omega \rightarrow R_\Omega \rightarrow R_\Omega = \mathcal{S}_{\Omega\rightarrow\Omega\rightarrow\Omega}[\overline{\mathcal{C}}/\overline{\chi}]$.

Proof $\twoheadrightarrow \in R_\Omega \rightarrow R_\Omega \rightarrow R_\Omega$ if for all $\tau_1 \in R_\Omega$, we get that $(\twoheadrightarrow)\tau_1 \in R_\Omega \rightarrow R_\Omega$. This is true if for all $\tau_2 \in R_\Omega$, we get that $(\twoheadrightarrow)\tau_1\,\tau_2 \in R_\Omega$. This is true if $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\twoheadrightarrow)\tau_1\,\tau_2\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$ is true with the conditions in proposition B.27. Since τ_1, τ_2, τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, and $\tau_{\forall^+}$ are strongly normalisable, we will induct over $len = \nu(\tau_1) + \nu(\tau_2) + \nu(\tau_{\mathsf{int}}) + \nu(\tau_{\rightarrow}) + \nu(\tau_{\forall}) + \nu(\tau_{\forall^+})$. We will prove that for all values of len, the type $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ ((\twoheadrightarrow)\tau_1\tau_2)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$; given that $\tau_1 \in R_\Omega$, and $\tau_2 \in R_\Omega$, and $\tau_{\mathsf{int}} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\rightarrow} \in \mathcal{S}_{\Omega\rightarrow\Omega\rightarrow\kappa\rightarrow\kappa\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall} \in \mathcal{S}_{\forall\chi.\,(\chi\rightarrow\Omega)\rightarrow(\chi\rightarrow\kappa)\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall^+} \in \mathcal{S}_{(\forall\chi.\,\Omega)\rightarrow(\forall\chi.\,\kappa)\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$. Consider $\tau = \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ ((\twoheadrightarrow)\tau_1\,\tau_2)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$.

- $len = 0$ The only reduction of τ is
$$\begin{aligned}\tau' = \tau_{\rightarrow}\,\tau_1\,\tau_2\,&(\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \tau_1\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+}))\\ &(\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \tau_2\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+}))\end{aligned}$$
Since both τ_1 and τ_2 belong to R_Ω, $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \tau_1\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ and $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \tau_2\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ belong to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. This implies that τ' also belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.
- $len = k + 1$ The other possible reductions come from the reduction of one of the individual types τ_1, τ_2, τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, and $\tau_{\forall^+}$. The proof in this case is similar to the proof of the corresponding case in lemma B.28.

Since τ is neutral, by property 3 of definition B.18, τ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. □

Lemma B.30 *If for all $\tau_1 \in \mathcal{S}_{\kappa_1}[\overline{\mathcal{C}}/\overline{\chi}]$, $\tau\{\tau_1/\alpha\} \in \mathcal{S}_{\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$, then $\lambda\alpha\!:\!\kappa_1\{\overline{\kappa}/\overline{\chi}\}.\,\tau \in \mathcal{S}_{\kappa_1\rightarrow\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$.*

Proof Consider the neutral type $\tau' = (\lambda\alpha : \kappa_1\{\overline{\kappa}/\overline{\chi}\}.\,\tau)\,\tau_1$. We have that τ_1 is strongly normalisable and $\tau\{\alpha'/\alpha\}$ is strongly normalisable. Therefore, τ is also strongly normalisable. We will induct over $len = \nu(\tau) + \nu(\tau_1)$. We will prove that for all values of len, the type $(\lambda\alpha\!:\!\kappa_1\{\overline{\kappa}/\overline{\chi}\}.\,\tau)\,\tau_1$ always reduces to a type that belongs to $\mathcal{S}_{\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$; given that $\tau_1 \in \mathcal{S}_{\kappa_1}[\overline{\mathcal{C}}/\overline{\chi}]$ and $\tau\{\tau_1/\alpha\} \in \mathcal{S}_{\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$.

- $len = 0$ There are two possible reductions. A beta reduction yields $\tau\{\tau_1/\alpha\}$ which by assumption belongs to $\mathcal{S}_{\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$. If $\tau = \tau_0\,\alpha$ and α does not occur free in τ_0, then we have an eta reduction to $\tau_0\,\tau_1$. But in this case $\tau\{\tau_1/\alpha\} = \tau_0\,\tau_1$.
- $len = k + 1$ For the inductive case, assume that the hypothesis is true for $len = k$. There are two additional reductions. The type τ' can reduce to $(\lambda\alpha\!:\!\kappa_1\{\overline{\kappa}/\overline{\chi}\}.\,\tau)\,\tau''_1$ where $\tau_1 \leadsto \tau''_1$. By property 2 of definition B.18, τ''_1 belongs to $\mathcal{S}_{\kappa_1}[\overline{\mathcal{C}}/\overline{\chi}]$. Therefore, $\tau\{\tau''_1/\alpha\}$ belongs to $\mathcal{S}_{\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$. Moreover, $len = k$. By the inductive hypothesis, $(\lambda\alpha\!:\!\kappa_1.\,\tau)\,\tau''_1$ always reduces to a type that belongs to $\mathcal{S}_{\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$. By property 3 of definition B.18, $(\lambda\alpha\!:\!\kappa_1.\,\tau)\,\tau''_1$ belongs to $\mathcal{S}_{\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$.
The other reduction of τ' is to $(\lambda\alpha\!:\!\kappa_1\{\overline{\kappa}/\overline{\chi}\}.\,\tau'')\,\tau_1$ where $\tau \leadsto \tau''$. By lemma B.14, $\tau\{\tau_1/\alpha\} \leadsto \tau''\{\tau_1/\alpha\}$. By property 2 of definition B.18, $\tau''\{\tau_1/\alpha\} \in \mathcal{S}_{\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$. Moreover, $len = k$ for the type τ'. Therefore, by the inductive hypothesis, $(\lambda\alpha\!:\!\kappa_1\{\overline{\kappa}/\overline{\chi}\}.\,\tau'')\,\tau_1$ always reduces to a type that belongs to $\mathcal{S}_{\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$. By property 3 of definition B.18, $(\lambda\alpha\!:\!\kappa_1\{\overline{\kappa}/\overline{\chi}\}.\,\tau'')\,\tau_1$ belongs to $\mathcal{S}_{\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$.

Therefore, the neutral type τ' always reduces to a type that belongs to $\mathcal{S}_{\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$. By property 3 of definition B.18, $\tau' \in \mathcal{S}_{\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$. Therefore, $\lambda\alpha : \kappa_1\{\overline{\kappa}/\overline{\chi}\}.\,\tau$ belongs to $\mathcal{S}_{\kappa_1}[\overline{\mathcal{C}}/\overline{\chi}] \rightarrow \mathcal{S}_{\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$. This implies that $\lambda\alpha\!:\!\kappa_1\{\overline{\kappa}/\overline{\chi}\}.\,\tau$ belongs to $\mathcal{S}_{\kappa_1\rightarrow\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$. □

Lemma B.31 $\forall \in \mathcal{S}_{\forall\chi.\,(\chi\rightarrow\Omega)\rightarrow\Omega}[\overline{\mathcal{C}}/\overline{\chi}]$.

Proof This is true if for any kind $\kappa_1\{\overline{\kappa}/\overline{\chi}\}$, $\forall\,[\kappa_1\{\overline{\kappa}/\overline{\chi}\}] \in \mathcal{S}_{(\chi\rightarrow\Omega)\rightarrow\Omega}[\overline{\mathcal{C}}, \mathcal{C}_{\kappa_1}/\overline{\chi}, \chi]$. This implies that

$$\forall\,[\kappa_1\{\overline{\kappa}/\overline{\chi}\}] \in \mathcal{S}_{\chi\rightarrow\Omega}[\overline{\mathcal{C}}, \mathcal{C}_{\kappa_1}/\overline{\chi}, \chi] \rightarrow \mathcal{S}_\Omega[\overline{\mathcal{C}}, \mathcal{C}_{\kappa_1}/\overline{\chi}, \chi]$$

This is true if for all $\tau \in \mathcal{S}_{\chi\rightarrow\Omega}[\overline{\mathcal{C}}, \mathcal{C}_{\kappa_1}/\overline{\chi}, \chi]$, it is true that $\forall\,[\kappa_1\{\overline{\kappa}/\overline{\chi}\}]\,\tau \in \mathcal{S}_\Omega[\overline{\mathcal{C}}, \mathcal{C}_{\kappa_1}/\overline{\chi}, \chi]$. This implies that $\forall\,[\kappa_1\{\overline{\kappa}/\overline{\chi}\}]\,\tau \in R_\Omega$. This is true if $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\forall\,[\kappa_1\{\overline{\kappa}/\overline{\chi}\}]\,\tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$ is true with the conditions in proposition B.27. Since each of the types τ, τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, and $\tau_{\forall^+}$ belongs to a candidate, they are strongly normalisable. We will induct over $len = \nu(\tau) + \nu(\tau_{\mathsf{int}}) + \nu(\tau_{\rightarrow}) + \nu(\tau_{\forall}) + \nu(\tau_{\forall^+})$. We will prove that for all values of len, the type $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\forall\,[\kappa_1\{\overline{\kappa}/\overline{\chi}\}]\,\tau)$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$; given that $\tau \in \mathcal{S}_{\chi\rightarrow\Omega}[\overline{\mathcal{C}}, \mathcal{C}_{\kappa_1}/\overline{\chi}, \chi]$, and $\tau_{\mathsf{int}} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\rightarrow} \in \mathcal{S}_{\Omega\rightarrow\Omega\rightarrow\kappa\rightarrow\kappa\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall} \in \mathcal{S}_{\forall\chi.\,(\chi\rightarrow\Omega)\rightarrow(\chi\rightarrow\kappa)\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall^+} \in \mathcal{S}_{(\forall\chi.\,\Omega)\rightarrow(\forall\chi.\,\kappa)\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$. Consider $\tau' = \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\forall\,[\kappa_1\{\overline{\kappa}/\overline{\chi}\}]\,\tau)$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$

- $len = 0$ Then the only possible reduction of τ' is

  $$\tau_1' = \tau_{\forall}\,[\kappa_1\{\overline{\kappa}/\overline{\chi}\}]\,\tau \\ (\lambda\alpha\!:\!\kappa_1\{\overline{\kappa}/\overline{\chi}\}.\ \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \tau\,\alpha \text{ of } (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+}))$$

  Consider $\tau'' = \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \tau\,\alpha$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$. For all $\tau_1 \in \mathcal{C}_{\kappa_1}$, the type $\tau''\{\tau_1/\alpha\}$ reduces to $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \tau\,\tau_1$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$. By assumption, τ belongs to $\mathcal{S}_\chi[\overline{\mathcal{C}}, \mathcal{C}_{\kappa_1}/\overline{\chi}, \chi] \rightarrow \mathcal{S}_\Omega[\overline{\mathcal{C}}, \mathcal{C}_{\kappa_1}/\overline{\chi}, \chi]$. Therefore, τ belongs to $\mathcal{C}_{\kappa_1} \rightarrow R_\Omega$ which implies that $\tau\,\tau_1 \in R_\Omega$. Therefore $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \tau\,\tau_1$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. Therefore, by lemma B.30, (replacing $\mathcal{S}_{\kappa_1}[\overline{\mathcal{C}}/\overline{\chi}]$ with $\mathcal{C}_{\kappa_1}$ in the lemma), $\lambda\alpha\!:\!\kappa_1\{\overline{\kappa}/\overline{\chi}\}.\ \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \tau\,\alpha$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ belongs to $\mathcal{C}_{\kappa_1} \rightarrow \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.

  By assumption, $\tau_{\forall}$ belongs to $\mathcal{S}_{\forall\chi.\,(\chi\rightarrow\Omega)\rightarrow(\chi\rightarrow\kappa)\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$. Therefore, $\tau_{\forall}\,[\kappa_1\{\overline{\kappa}/\overline{\chi}\}]$ belongs to $\mathcal{S}_{(\chi\rightarrow\Omega)\rightarrow(\chi\rightarrow\kappa)\rightarrow\kappa}[\overline{\mathcal{C}}, \mathcal{C}_{\kappa_1}/\overline{\chi}, \chi]$. This implies that $\tau_{\forall}\,[\kappa_1\{\overline{\kappa}/\overline{\chi}\}]\,\tau$ belongs to $\mathcal{S}_{(\chi\rightarrow\kappa)\rightarrow\kappa}[\overline{\mathcal{C}}, \mathcal{C}_{\kappa_1}/\overline{\chi}, \chi]$.

  Consider $\mathcal{C} = \mathcal{S}_{(\chi\rightarrow\kappa)\rightarrow\kappa}[\overline{\mathcal{C}}, \mathcal{C}_{\kappa_1}/\overline{\chi}, \chi]$. Then $\mathcal{C}$ is equal to $\mathcal{S}_{\chi\rightarrow\kappa}[\overline{\mathcal{C}}, \mathcal{C}_{\kappa_1}/\overline{\chi}, \chi] \rightarrow \mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{C}_{\kappa_1}/\overline{\chi}, \chi]$. This is equivalent to $(\mathcal{C}_{\kappa_1} \rightarrow \mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{C}_{\kappa_1}/\overline{\chi}, \chi]) \rightarrow \mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{C}_{\kappa_1}/\overline{\chi}, \chi]$. But χ does not occur free in κ. So the above can be written as $(\mathcal{C}_{\kappa_1} \rightarrow \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]) \rightarrow \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. This implies that τ_1' belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.

- $len = k + 1$ The other possible reductions come from the reduction of one of the individual types τ, τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, and $\tau_{\forall^+}$. The proof in this case is similar to the proof of the corresponding case in lemma B.28.

Since τ' is neutral, by property 3 of definition B.18, τ' belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. □

Lemma B.32 *If for every kind κ' and reducibility candidate $\mathcal{C}'$ of this kind, $\tau\{\kappa'/\chi'\} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$, then $\Lambda\chi'.\,\tau \in \mathcal{S}_{\forall\chi'.\,\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$.*

Proof Consider the neutral type $\tau' = (\Lambda\chi'.\,\tau)\,[\kappa']$ for an arbitrary kind κ'. Since $\tau\{\chi''/\chi'\}$ is strongly normalisable, τ is strongly normalisable. We will induct over $len = \nu(\tau)$. We will prove that for all values of len, the neutral type $(\Lambda\chi'.\,\tau)\,[\kappa']$ always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$; given that $\tau\{\kappa'/\chi'\} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$.

- $len = 0$ There are two possible reductions. A beta reduction yields $\tau\{\kappa'/\chi'\}$ which by assumption belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$. If $\tau = \tau_0\,[\chi']$ and χ' does not occur free in τ_0, then we have an eta reduction to $\tau_0\,[\kappa']$. But in this case $\tau\{\kappa'/\chi'\} = \tau_0\,[\kappa']$.

- $len = k + 1$ For the inductive case, assume that the hypothesis is true for $len = k$. There is one additional reduction, $(\Lambda\chi'.\,\tau)\,[\kappa'] \leadsto (\Lambda\chi'.\,\tau_1)\,[\kappa']$ where $\tau \leadsto \tau_1$. By lemma B.15, we know that $\tau\{\kappa'/\chi'\} \leadsto \tau_1\{\kappa'/\chi'\}$. By property 2 of definition B.18, $\tau_1\{\kappa'/\chi'\} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$. Moreover, $len = k$ for this reduct. Therefore, by the inductive hypothesis, $(\Lambda\chi'.\,\tau_1)\,[\kappa']$ always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$. By property 3 of definition B.18, $(\Lambda\chi'.\,\tau_1)\,[\kappa']$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$.

Therefore, the neutral type τ' always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$. By property 3 of definition B.18, $\tau' \in \mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$. Therefore, $\Lambda\chi'.\,\tau$ belongs to $\mathcal{S}_{\forall\chi'.\,\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$. □

Lemma B.33 *If $\tau \in \mathcal{S}_{\forall\chi.\,\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$, then for every kind $\kappa'\{\overline{\kappa}/\overline{\chi}\}$ $\tau\,[\kappa'\{\overline{\kappa}/\overline{\chi}\}] \in \mathcal{S}_{\kappa\{\kappa'/\chi\}}[\overline{\mathcal{C}}/\overline{\chi}]$.*

Proof By definition, $\tau\,[\kappa'\{\overline{\kappa}/\overline{\chi}\}]$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi]$, for every kind $\kappa'\{\overline{\kappa}/\overline{\chi}\}$ and reducibility candidate $\mathcal{C}'$ of this kind. Set $\mathcal{C}' = \mathcal{S}_{\kappa'}[\overline{\mathcal{C}}/\overline{\chi}]$. Applying lemma B.26 leads to the result. □

Lemma B.34 $\forall^+ \in \mathcal{S}_{(\forall\chi.\,\Omega)\rightarrow\Omega}[\overline{\mathcal{C}}/\overline{\chi}]$.

Proof This is true if for all $\tau \in \mathcal{S}_{\forall\chi.\,\Omega}[\overline{\mathcal{C}}/\overline{\chi}]$, we have $\forall^+\tau \in R_\Omega$. This is true if $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\forall^+\tau)$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$ with the conditions in proposition B.27. Since all the types are strongly normalisable, we will induct over $len = \nu(\tau) + \nu(\tau_{\mathsf{int}}) + \nu(\tau_{\rightarrow}) + \nu(\tau_{\forall}) + \nu(\tau_{\forall^+})$. We will prove that for all values of len, the type $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\forall^+\tau)$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$; given that $\tau \in \mathcal{S}_{\forall\chi.\,\Omega}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\mathsf{int}} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\rightarrow} \in \mathcal{S}_{\Omega\rightarrow\Omega\rightarrow\kappa\rightarrow\kappa\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall} \in \mathcal{S}_{\forall\chi.\,(\chi\rightarrow\Omega)\rightarrow(\chi\rightarrow\kappa)\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall^+} \in \mathcal{S}_{(\forall\chi.\,\Omega)\rightarrow(\forall\chi.\,\kappa)\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$. Consider $\tau' = \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\forall^+\tau)$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$

- $len = 0$ Then the only possible reduction of τ' is

  $$\tau_{\forall^+}\,\tau\,(\Lambda\chi.\ \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\tau\,[\chi]) \text{ of } (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+}))$$

  Consider $\tau'' = \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\tau\,[\chi])$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$. For an arbitrary kind κ', $\tau''\{\kappa'/\chi\}$ is equal to $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \tau\,[\kappa']$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$. By the assumption on τ, we get that $\tau\,[\kappa'] \in R_\Omega$. Therefore, by definition, $\tau''\{\kappa'/\chi\} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. Since χ does not occur free in κ, we can write this as $\tau''\{\kappa'/\chi\} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi]$ for a candidate $\mathcal{C}'$ of kind κ'. By lemma B.32 $\Lambda\chi.\ \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\tau\,[\chi])$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ belongs to $\mathcal{S}_{\forall\chi.\,\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$. By the assumptions on $\tau_{\forall^+}$ and τ, $\tau_{\forall^+}\,\tau\,(\Lambda\chi.\ \mathsf{Typerec}[\kappa]\ (\tau\,[\chi])$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+}))$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.

- $len = k + 1$ The other possible reductions come from the reduction of one of the individual types τ, τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, and $\tau_{\forall^+}$. The proof in this case is similar to the proof of the corresponding case in lemma B.28.

Since τ' is neutral, by property 3 of definition B.18, τ' belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. □

We now come to the main result of this section.

Theorem B.35 (Candidacy) *Let τ be a type of kind κ. Suppose all the free type variables of τ are in $\alpha_1 \dots \alpha_n$ of kinds $\kappa_1 \dots \kappa_n$ and all the free kind variables of κ, $\kappa_1 \dots \kappa_n$ are among $\chi_1 \dots \chi_m$. If $\mathcal{C}_1 \dots \mathcal{C}_m$ are candidates of kinds $\kappa'_1 \dots \kappa'_m$ and $\tau_1 \dots \tau_n$ are types of kind $\kappa_1\{\overline{\kappa'}/\overline{\chi}\} \dots \kappa_n\{\overline{\kappa'}/\overline{\chi}\}$ which are in $\mathcal{S}_{\kappa_1}[\overline{\mathcal{C}}/\overline{\chi}] \dots \mathcal{S}_{\kappa_n}[\overline{\mathcal{C}}/\overline{\chi}]$, then $\tau\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.*

Proof The proof is by induction over the structure of τ.

The cases of int, $\twoheadrightarrow$, $\forall$, $\overset{+}{\forall}$ are covered by lemmas B.28 B.29 B.31 B.34.

Suppose $\tau = \alpha_i$ and $\kappa = \kappa_i$. Then $\tau\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\} = \tau_i$. By assumption, this belongs to $\mathcal{S}_{\kappa_i}[\overline{\mathcal{C}}/\overline{\chi}]$.

Suppose $\tau = \tau'_1\,\tau'_2$. Then $\tau'_1 : \kappa' \to \kappa$ for some kind κ' and $\tau'_2 : \kappa'$. By the inductive hypothesis, $\tau'_1\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_{\kappa' \to \kappa}[\overline{\mathcal{C}}/\overline{\chi}]$ and $\tau'_2\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_{\kappa'}[\overline{\mathcal{C}}/\overline{\chi}]$. Therefore, $(\tau'_1\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\})\,(\tau'_2\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\})$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.

Suppose $\tau = \tau'\,[\kappa']$. Then $\tau' : \forall \chi_1.\, \kappa_1$ and $\kappa = \kappa_1\{\kappa'/\chi_1\}$. By the inductive hypothesis, $\tau'\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_{\forall\chi_1.\,\kappa_1}[\overline{\mathcal{C}}/\overline{\chi}]$. By lemma B.33 $\tau'\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}\ [\kappa'\{\overline{\kappa'}/\overline{\chi}\}]$ belongs to $\mathcal{S}_{\kappa_1\{\kappa'/\chi_1\}}[\overline{\mathcal{C}}/\overline{\chi}]$ which is equivalent to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.

Suppose $\tau = \mathsf{Typerec}[\kappa]\ \tau'$ of $(\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\overset{+}{\forall}})$. Then $\tau' : \Omega$, and $\tau_{\mathsf{int}} : \kappa$, and $\tau_\to : \Omega \to \Omega \to \kappa \to \kappa \to \kappa$, and $\tau_\forall : \forall\chi.\,(\chi \to \Omega) \to (\chi \to \kappa) \to \kappa$, and $\tau_{\overset{+}{\forall}} : (\forall\chi.\,\Omega) \to (\forall\chi.\,\kappa) \to \kappa$. By the inductive hypothesis $\tau'\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to R_Ω, and $\tau_{\mathsf{int}}\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\to\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_{\Omega\to\Omega\to\kappa\to\kappa\to\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\forall\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_{\forall\chi.\,(\chi\to\Omega)\to(\chi\to\kappa)\to\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\overset{+}{\forall}}\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_{(\forall\chi.\,\Omega)\to(\forall\chi.\,\kappa)\to\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$. By definition of R_Ω,

$$\begin{array}{l}\mathsf{Typerec}[\kappa\{\overline{\kappa'}/\overline{\chi}\}]\ \tau'\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\};\tau_\to\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}; \\ \quad\ \tau_\forall\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\};\tau_{\overset{+}{\forall}}\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\})\end{array}$$

belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.

Suppose $\tau = \lambda\alpha' : \kappa'.\,\tau_1$. Then $\tau_1 : \kappa''$ where the free type variables of τ_1 are in $\alpha_1, \dots, \alpha_n, \alpha'$ and $\kappa = \kappa' \to \kappa''$. By the inductive hypothesis, $\tau_1\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}, \tau'/\overline{\alpha}, \alpha'\}$ belongs to $\mathcal{S}_{\kappa''}[\overline{\mathcal{C}}/\overline{\chi}]$ where τ' is of kind $\kappa'\{\overline{\kappa'}/\overline{\chi}\}$ and belongs to $\mathcal{S}_{\kappa'}[\overline{\mathcal{C}}/\overline{\chi}]$. This implies that $(\tau_1\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\})\{\tau'/\alpha'\}$ (since α' occurs free only in τ_1) belongs to $\mathcal{S}_{\kappa''}[\overline{\mathcal{C}}/\overline{\chi}]$. By lemma B.30, $\lambda\alpha' : \kappa'\{\overline{\kappa'}/\overline{\chi}\}.\,(\tau_1\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\})$ belongs to $\mathcal{S}_{\kappa'\to\kappa''}[\overline{\mathcal{C}}/\overline{\chi}]$.

Suppose $\tau = \Lambda\chi'.\,\tau'$. Then $\tau' : \kappa''$ and $\kappa = \forall\chi'.\,\kappa''$. By the inductive hypothesis, $\tau'\{\overline{\kappa'}, \kappa'/\overline{\chi}, \chi'\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_{\kappa''}[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$ for an arbitrary kind κ' and candidate $\mathcal{C}'$ of kind κ'. Since χ' occurs free only in τ', we get that $(\tau'\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\})\{\kappa'/\chi'\}$ belongs to $\mathcal{S}_{\kappa''}[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$. By lemma B.32, $\Lambda\chi'.\,(\tau'\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\})$ belongs to $\mathcal{S}_{\forall\chi'.\,\kappa''}[\overline{\mathcal{C}}/\overline{\chi}]$. □

Suppose SN_i is the set of strongly normalisable types of kind κ_i.

Corollary B.36 *All types are strongly normalisable.*

Proof Follows from theorem B.35 by putting $\mathcal{C}_i = SN_i$ and $\tau_i = \alpha_i$. □

B.3 Confluence

The type contexts C are shown in Figure 26. The reduction rules are shown in Figure 27.

$$\begin{array}{lcl} (\mathit{context})\ \ C & ::= & [\,]\ \mid\ \twoheadrightarrow C\ \mid\ \twoheadrightarrow(C, \tau)\ \mid\ \twoheadrightarrow(\tau, C) \\ & \mid & \forall\,[\kappa]\,C\ \mid\ \overset{+}{\forall} C\ \mid\ \Lambda\chi.\,C\ \mid\ C\,[\kappa] \\ & \mid & \lambda\alpha{:}\kappa.\,C\ \mid\ C\,\tau\ \mid\ \tau\,C \\ & \mid & \mathsf{Typerec}[\kappa]\ C\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\overset{+}{\forall}}) \\ & \mid & \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (C;\ \tau_\to;\ \tau_\forall;\ \tau_{\overset{+}{\forall}}) \\ & \mid & \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ C;\ \tau_\forall;\ \tau_{\overset{+}{\forall}}) \\ & \mid & \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_\to;\ C;\ \tau_{\overset{+}{\forall}}) \\ & \mid & \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ C) \end{array}$$

Figure 26: Type contexts

$$\begin{array}{lcl} (\beta_1) & ::= & (\lambda\alpha{:}\kappa.\,\tau)\,\tau' \rightsquigarrow \tau\{\tau'/\alpha\} \\ (\beta_2) & ::= & (\Lambda\chi.\,\tau)\,[\kappa] \rightsquigarrow \tau\{\kappa/\chi\} \\ (\eta_1) & ::= & \lambda\alpha{:}\kappa.\,\tau\,\alpha \rightsquigarrow \tau \qquad \alpha \notin \mathit{ftv}(\tau) \\ (\eta_2) & ::= & \Lambda\chi.\,\tau\,[\chi] \rightsquigarrow \tau \qquad \chi \notin \mathit{fkv}(\tau) \\ (t_1) & ::= & \mathsf{Typerec}[\kappa]\ \mathsf{int}\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\overset{+}{\forall}}) \rightsquigarrow \tau_{\mathsf{int}} \\ (t_2) & ::= & \mathsf{Typerec}[\kappa]\ \tau_1 \to \tau_2\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\overset{+}{\forall}}) \rightsquigarrow \\ & & \quad \tau_\to\ \tau_1\ \tau_2 \\ & & \quad\ (\mathsf{Typerec}[\kappa]\ \tau_1\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\overset{+}{\forall}})) \\ & & \quad\ (\mathsf{Typerec}[\kappa]\ \tau_2\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\overset{+}{\forall}})) \\ (t_3) & ::= & \mathsf{Typerec}[\kappa]\ \forall\,[\kappa']\,\tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\overset{+}{\forall}}) \rightsquigarrow \\ & & \quad \tau_\forall\,[\kappa']\,\tau \\ & & \quad\ (\lambda\alpha{:}\kappa'.\,\mathsf{Typerec}[\kappa]\ (\tau\,\alpha)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\overset{+}{\forall}})) \\ (t_4) & ::= & \mathsf{Typerec}[\kappa]\ \overset{+}{\forall}\tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\overset{+}{\forall}}) \rightsquigarrow \\ & & \quad \tau_{\overset{+}{\forall}}\,\tau \\ & & \quad\ (\Lambda\chi.\,\mathsf{Typerec}[\kappa]\ \tau\,[\chi]\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\overset{+}{\forall}})) \end{array}$$

Figure 27: Type reductions

Definition B.37 $\tau_1 \mapsto \tau_2$ *iff there exists a τ'_1 and τ'_2 and C such that $\tau_1 = C\,[\tau'_1]$ and $\tau_2 = C\,[\tau'_2]$ and $\tau'_1 \rightsquigarrow \tau'_2$.*

Lemma B.38 *If $\tau_1 \mapsto \tau_2$, then $\tau_1\{\tau/\alpha\} \mapsto \tau_2\{\tau/\alpha\}$.*

Proof This requires us to prove that if $\tau' \rightsquigarrow \tau''$, then $\tau'\{\tau/\alpha\} \rightsquigarrow \tau''\{\tau/\alpha\}$. This follows from lemma B.14. □

Lemma B.39 *If $\tau_1 \mapsto \tau_2$, then $\tau_1\{\kappa/\chi\} \mapsto \tau_2\{\kappa/\chi\}$.*

Proof This requires us to prove that if $\tau' \rightsquigarrow \tau''$, then $\tau'\{\kappa/\chi\} \rightsquigarrow \tau''\{\kappa/\chi\}$. This follows from lemma B.15. □

Lemma B.40 *If $\tau_1 \mapsto \tau_2$, then $\tau\{\tau_1/\alpha\} \mapsto \tau\{\tau_2/\alpha\}$.*

Proof This is proved by induction over the structure of τ and then defining an appropriate type context C.

Suppose $\tau = \Lambda\chi.\,\tau'$. Then $\tau\{\tau_1/\alpha\} = \Lambda\chi.\,\tau'\{\tau_1/\alpha\}$. By induction assume that $\tau'\{\tau_1/\alpha\} \mapsto \tau'\{\tau_2/\alpha\}$. This implies that for some context C, $\tau'\{\tau_1/\alpha\} = C\,[\tau'_1]$ and $\tau'\{\tau_2/\alpha\} = C\,[\tau'_2]$ and $\tau'_1 \rightsquigarrow \tau'_2$. Consider the context $C_0 = \Lambda\chi.\,C$. Then we get that $\Lambda\chi.\,\tau'\{\tau_1/\alpha\} = C_0\,[\tau'_1]$ and $\Lambda\chi.\,\tau'\{\tau_2/\alpha\} = C_0\,[\tau'_2]$.

Suppose $\tau = \lambda\beta : \kappa.\,\tau'$. Then $\tau\{\tau_1/\alpha\} = \lambda\beta{:}\kappa.\,\tau'\{\tau_1/\alpha\}$. By induction assume that $\tau'\{\tau_1/\alpha\} \mapsto \tau'\{\tau_2/\alpha\}$. This implies that for some context C, $\tau'\{\tau_1/\alpha\} = C\,[\tau'_1]$ and $\tau'\{\tau_2/\alpha\} = C\,[\tau'_2]$ and $\tau'_1 \rightsquigarrow \tau'_2$. Consider the context $C_0 = \lambda\beta{:}\kappa.\,C$. Then we get that $\lambda\beta{:}\kappa.\,\tau'\{\tau_1/\alpha\} = C_0\,[\tau'_1]$ and $\lambda\beta{:}\kappa.\,\tau'\{\tau_2/\alpha\} = C_0\,[\tau'_2]$.

Suppose $\tau = \tau' [\kappa]$. Then $\tau\{\tau_1/\alpha\} = (\tau'\{\tau_1/\alpha\}) [\kappa]$. By induction assume that $\tau'\{\tau_1/\alpha\} \mapsto \tau'\{\tau_2/\alpha\}$. This implies that for some context C, $\tau'\{\tau_1/\alpha\} = C [\tau_1']$ and $\tau'\{\tau_2/\alpha\} = C [\tau_2']$ and $\tau_1' \leadsto \tau_2'$. Consider the context $C_0 = C [\kappa]$. Then we get that $(\tau'\{\tau_1/\alpha\}) [\kappa] = C_0 [\tau_1']$ and $(\tau'\{\tau_2/\alpha\}) [\kappa] = C_0 [\tau_2']$.

Suppose $\tau = \tau' \tau''$. Then $\tau\{\tau_1/\alpha\} = (\tau'\{\tau_1/\alpha\}) (\tau''\{\tau_1/\alpha\})$. By induction assume that $\tau'\{\tau_1/\alpha\} \mapsto \tau'\{\tau_2/\alpha\}$ and $\tau''\{\tau_1/\alpha\} \mapsto \tau''\{\tau_2/\alpha\}$. This implies that for some context C, $\tau'\{\tau_1/\alpha\} = C [\tau_1']$ and $\tau'\{\tau_2/\alpha\} = C [\tau_2']$ and $\tau_1' \leadsto \tau_2'$. Consider the context $C_0 = C (\tau''\{\tau_1/\alpha\})$. Then we get that $(\tau'\{\tau_1/\alpha\}) (\tau''\{\tau_1/\alpha\}) = C_0 [\tau_1']$ and $(\tau'\{\tau_2/\alpha\}) (\tau''\{\tau_1/\alpha\}) = C_0 [\tau_2']$. Repeating the same process, but this time starting with $(\tau'\{\tau_2/\alpha\}) (\tau''\{\tau_1/\alpha\})$ leads to the lemma.

Suppose $\tau = \mathsf{Typerec}[\kappa]\ \tau'\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$. Then

$$\begin{array}{l}\tau\{\tau_1/\alpha\} = \\ \mathsf{Typerec}[\kappa]\ (\tau'\{\tau_1/\alpha\})\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau_1/\alpha\};\ \tau_{\rightarrow}\{\tau_1/\alpha\};\ \tau_{\forall}\{\tau_1/\alpha\};\ \tau_{\forall^+}\{\tau_1/\alpha\})\end{array}$$

By induction assume that $\tau'\{\tau_1/\alpha\} \mapsto \tau'\{\tau_2/\alpha\}$ and $\tau_{\mathsf{int}}\{\tau_1/\alpha\} \mapsto \tau_{\mathsf{int}}\{\tau_2/\alpha\}$ and $\tau_{\rightarrow}\{\tau_1/\alpha\} \mapsto \tau_{\rightarrow}\{\tau_2/\alpha\}$ and $\tau_{\forall}\{\tau_1/\alpha\} \mapsto \tau_{\forall}\{\tau_2/\alpha\}$ and $\tau_{\forall^+}\{\tau_1/\alpha\} \mapsto \tau_{\forall^+}\{\tau_2/\alpha\}$. This implies that for some context C, $\tau'\{\tau_1/\alpha\} = C [\tau_1']$ and $\tau'\{\tau_2/\alpha\} = C [\tau_2']$ and $\tau_1' \leadsto \tau_2'$. Consider the context

$$\begin{array}{l}C_0 = \\ \mathsf{Typerec}[\kappa]\ C\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau_1/\alpha\};\ \tau_{\rightarrow}\{\tau_1/\alpha\};\ \tau_{\forall}\{\tau_1/\alpha\};\ \tau_{\forall^+}\{\tau_1/\alpha\})\end{array}$$

Then we get that

$$\begin{array}{l}C_0 [\tau_1'] = \\ \mathsf{Typerec}[\kappa]\ (\tau'\{\tau_1/\alpha\})\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau_1/\alpha\};\ \tau_{\rightarrow}\{\tau_1/\alpha\};\ \tau_{\forall}\{\tau_1/\alpha\};\ \tau_{\forall^+}\{\tau_1/\alpha\})\end{array}$$

and

$$\begin{array}{l}C_0 [\tau_2'] = \\ \mathsf{Typerec}[\kappa]\ (\tau'\{\tau_2/\alpha\})\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau_1/\alpha\};\ \tau_{\rightarrow}\{\tau_1/\alpha\};\ \tau_{\forall}\{\tau_1/\alpha\};\ \tau_{\forall^+}\{\tau_1/\alpha\})\end{array}$$

Repeating this process with the other subtypes leads to the lemma. □

Theorem B.41 *If τ is strongly normalizing and locally confluent, then τ is confluent.*

Proof This is proved by induction over $\nu(\tau)$. □

To prove local confluence, we consider types with two holes. The contexts are specified in Figure 28. Given a type τ', we may write it as $C_1 [\tau_1]$ or as $C_2 [\tau_2]$. The two holes, τ_1 and τ_2 are said to overlap if one is a subterm of the other. If the two holes do not overlap, then τ' may be written as $D [\tau'', \tau''']$ and it is obvious that the reduction is locally confluent.

We therefore need to consider only overlapping holes, that is $\tau' = C_1 [\tau]$ and $\tau = C_2 [\tau_1]$. Without loss of generality, we may discard the outer context C_1.

The local confluence is now proved by considering each possible reduction of τ according to the reduction rules and for each case, showing that there exists another set of reductions that guarantees local confluence.

We show that if $\tau \leadsto \tau''$, then for each rule such that $\tau_1 \leadsto \tau_1'$, there exists a τ''' and a sequence of reductions that take τ'' to τ'''

$$\begin{array}{llll}
(\mathit{context}) & D & ::= & \rightarrow(C_1, C_2) \mid C_1\, C_2 \\
& & \mid & \mathsf{Typerec}[\kappa]\ C_1\ \mathsf{of}\ (C_2;\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+}) \\
& & \mid & \mathsf{Typerec}[\kappa]\ C_1\ \mathsf{of}\ (\tau_{\mathsf{int}};\ C_2;\ \tau_{\forall};\ \tau_{\forall^+}) \\
& & \mid & \mathsf{Typerec}[\kappa]\ C_1\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ C_2;\ \tau_{\forall^+}) \\
& & \mid & \mathsf{Typerec}[\kappa]\ C_1\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ C_2) \\
& & \mid & \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (C_1;\ C_2;\ \tau_{\forall};\ \tau_{\forall^+}) \\
& & \mid & \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (C_1;\ \tau_{\rightarrow};\ C_2;\ \tau_{\forall^+}) \\
& & \mid & \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (C_1;\ \tau_{\rightarrow};\ \tau_{\forall};\ C_2) \\
& & \mid & \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ C_1;\ C_2;\ \tau_{\forall^+}) \\
& & \mid & \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ C_1;\ \tau_{\forall};\ C_2) \\
& & \mid & \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ C_1;\ C_2) \\
& & \mid & C [D]
\end{array}$$

Figure 28: Type contexts with two holes

and $C_2 [\tau_1']$ to τ'''. We use a diagram to prove this. The left arrow represents the reduction from τ to τ'' and the right arrow shows the reduction from $C_2 [\tau_1]$ to $C_2 [\tau_1']$. The dashed arrows are then used to show the reductions that complete local confluence.

The set of reductions is shown in Figure 27. We use T to denote the complete set of reductions.

case β_1: Suppose τ is a beta redex $(\lambda\alpha : \kappa.\, \tau_1)\, \tau_2$. Suppose further that $\tau_1 \leadsto \tau_1'$ through any reduction in T apart from a eta-redex.

$$\begin{tikzcd}
 & (\lambda\alpha{:}\kappa.\,\tau_1)\,\tau_2 \arrow[dl, "\beta_1"'] \arrow[dr, "T-\eta_1"] & \\
\tau_1\{\tau_2/\alpha\} \arrow[dr, dashed, "\text{lemma B.38}"'] & & (\lambda\alpha{:}\kappa.\,\tau_1')\,\tau_2 \arrow[dl, dashed, "\beta_1"] \\
 & \tau_1'\{\tau_2/\alpha\} &
\end{tikzcd}$$

Suppose that $\tau_1 \rightsquigarrow \tau_1'$ through a eta-redex. Assume $\tau_1 = \tau_1'\,\alpha$.

$$\begin{tikzcd}
 & (\lambda\alpha{:}\kappa.\,\tau_1)\,\tau_2 \arrow[dl, "\beta_1"'] \arrow[dr, "\eta_1"] & \\
\tau_1'\,\tau_2 \arrow[dr, dashed, "="'] & & \tau_1'\,\tau_2 \arrow[dl, dashed, "="] \\
 & \tau_1'\,\tau_2 &
\end{tikzcd}$$

Suppose that $\tau_2 \rightsquigarrow \tau_2'$ through any reduction in T.

$$\begin{tikzcd}
 & (\lambda\alpha{:}\kappa.\,\tau_1)\,\tau_2 \arrow[dl, "\beta_1"'] \arrow[dr, "T"] & \\
\tau_1\{\tau_2/\alpha\} \arrow[dr, dashed, "\text{lemma B.40}"'] & & (\lambda\alpha{:}\kappa.\,\tau_1)\,\tau_2' \arrow[dl, dashed, "\beta_1"] \\
 & \tau_1\{\tau_2'/\alpha\} &
\end{tikzcd}$$

case β_2: This is similar to the β_1 case. When τ_1 reduces by an eta redex, assume that $\tau_1 = \tau_1'\,[\chi_1]$

$$\begin{tikzcd}
 & (\Lambda\chi_1.\,\tau_1)\,[\kappa_1] \arrow[dl, "\beta_2"'] \arrow[dr, "T-\eta_2"] & \\
\tau_1\{\kappa_1/\chi_1\} \arrow[dr, dashed, "\text{lemma B.39}"'] & & (\Lambda\chi_1.\,\tau_1')\,[\kappa_1] \arrow[dl, dashed, "\beta_2"] \\
 & \tau_1'\{\kappa_1/\chi_1\} &
\end{tikzcd}$$

$$\begin{tikzcd}
 & (\Lambda\chi_1.\,\tau_1)\,[\kappa_1] \arrow[dl, "\beta_2"'] \arrow[dr, "\eta_2"] & \\
\tau_1\{\kappa_1/\chi_1\} \arrow[dr, dashed, "="'] & & \tau_1'\,[\kappa_1] \arrow[dl, dashed, "="] \\
 & \tau_1'\,[\kappa_1] &
\end{tikzcd}$$

case η_1: When the right arrow denotes a beta-redex, assume that $\tau = \lambda\beta{:}\kappa.\,\tau_1$

$$\begin{tikzcd}
& \lambda\alpha:\kappa.\,(\tau\,\alpha) \arrow[dl, "\eta_1"'] \arrow[dr, "T-\beta_1"] & \\
\tau \arrow[dr, dashed, "T-\beta_1"'] & & \lambda\alpha:\kappa.\,(\tau'\,\alpha) \arrow[dl, dashed, "\eta_1"] \\
& \tau' &
\end{tikzcd}$$

$$\begin{tikzcd}
& \lambda\alpha:\kappa.\,(\tau\,\alpha) \arrow[dl, "\eta_1"'] \arrow[dr, "\beta_1"] & \\
\lambda\beta:\kappa.\,\tau_1 \arrow[dr, dashed, "="'] & & \lambda\alpha:\kappa.\,\tau_1\{\alpha/\beta\} \arrow[dl, dashed, "\alpha_{conv}"] \\
& \lambda\beta:\kappa.\,\tau_1 &
\end{tikzcd}$$

case η_2: This is similar to the η_1 case. When the right arrow denotes a beta-redex, assume that $\tau = \Lambda\chi_1.\,\tau_1$.

$$\begin{tikzcd}
& \Lambda\chi.\,(\tau\,[\chi]) \arrow[dl, "\eta_2"'] \arrow[dr, "T-\beta_2"] & \\
\tau \arrow[dr, dashed, "T-\beta_2"'] & & \Lambda\chi.\,(\tau'\,[\chi]) \arrow[dl, dashed, "\eta_2"] \\
& \tau' &
\end{tikzcd}$$

$$\begin{tikzcd}
& \Lambda\chi.\,(\tau\,[\chi]) \arrow[dl, "\eta_2"'] \arrow[dr, "\beta_2"] & \\
\Lambda\chi_1.\,\tau_1 \arrow[dr, dashed, "="'] & & \Lambda\chi.\,\tau_1\{\chi/\chi_1\} \arrow[dl, dashed, "\alpha_{conv}"] \\
& \Lambda\chi_1.\,\tau_1 &
\end{tikzcd}$$

case t_1: We consider only the case of $\tau_{\mathsf{int}} \rightsquigarrow \tau'_{\mathsf{int}}$. The other possible reductions are locally confluent in an obvious way.

$$\begin{tikzcd}
& \mathsf{Typerec}[\kappa]\ \mathsf{int}\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+}) \arrow[dl, "t_1"'] \arrow[dr, "T"] & \\
\tau_{\mathsf{int}} \arrow[dr, dashed, "T"'] & & \mathsf{Typerec}[\kappa]\ \mathsf{int}\ \mathsf{of}\ (\tau'_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+}) \arrow[dl, dashed, "t_1"] \\
& \tau'_{\mathsf{int}} &
\end{tikzcd}$$

case t_2: There are six possible subcases from the reduction of either τ_1, τ_2, τ_{int}, $\tau_{\to}$, $\tau_{\forall}$, or $\tau_{\forall^+}$. The case for reduction of τ_1 and τ_2 are similar; we will show only the case for the reduction of τ_1. We use $\overline{\tau}$ to denote the types $(\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+})$

$$
\begin{tikzcd}
& \mathsf{Typerec}[\kappa]\ \tau_1 \to \tau_2 \text{ of } \overline{\tau} \arrow[dl, "t_2"'] \arrow[dr, "T"] & \\
\begin{array}{l} \tau_{\to}\ \tau_1\ \tau_2 \\ \quad (\mathsf{Typerec}[\kappa]\ \tau_1 \text{ of } \overline{\tau}) \\ \quad\quad (\mathsf{Typerec}[\kappa]\ \tau_2 \text{ of } \overline{\tau}) \end{array} \arrow[dr, dashed, "T^*"'] & & \mathsf{Typerec}[\kappa]\ \tau_1' \to \tau_2 \text{ of } \overline{\tau} \arrow[dl, dashed, "t_2"] \\
& \begin{array}{l} \tau_{\to}\ \tau_1'\ \tau_2 \\ \quad (\mathsf{Typerec}[\kappa]\ \tau_1' \text{ of } \overline{\tau}) \\ \quad\quad (\mathsf{Typerec}[\kappa]\ \tau_2 \text{ of } \overline{\tau}) \end{array} &
\end{tikzcd}
$$

We will also only show the reduction of $\tau_{\to}$. The rest of the reductions are locally confluent in an obvious way. We use $\overline{\tau}$ to denote the types $(\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+})$. We use $\overline{\tau'}$ to denote the types $(\tau_{\mathsf{int}};\ \tau_{\to}';\ \tau_{\forall};\ \tau_{\forall^+})$

$$
\begin{tikzcd}
& \mathsf{Typerec}[\kappa]\ \tau_1 \to \tau_2 \text{ of } \overline{\tau} \arrow[dl, "t_2"'] \arrow[dr, "T"] & \\
\begin{array}{l} \tau_{\to}\ \tau_1\ \tau_2 \\ \quad (\mathsf{Typerec}[\kappa]\ \tau_1 \text{ of } \overline{\tau}) \\ \quad\quad (\mathsf{Typerec}[\kappa]\ \tau_2 \text{ of } \overline{\tau}) \end{array} \arrow[dr, dashed, "T^*"'] & & \mathsf{Typerec}[\kappa]\ \tau_1 \to \tau_2 \text{ of } \overline{\tau'} \arrow[dl, dashed, "t_2"] \\
& \begin{array}{l} \tau_{\to}'\ \tau_1\ \tau_2 \\ \quad (\mathsf{Typerec}[\kappa]\ \tau_1 \text{ of } \overline{\tau'}) \\ \quad\quad (\mathsf{Typerec}[\kappa]\ \tau_2 \text{ of } \overline{\tau'}) \end{array} &
\end{tikzcd}
$$

case t_3: There are five possible subcases from the reduction of either τ_1, τ_{int}, $\tau_{\to}$, $\tau_{\forall}$, or $\tau_{\forall^+}$. We first show the reduction of τ_1. We use $\overline{\tau}$ to denote the types $(\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+})$

$$
\begin{tikzcd}
& \mathsf{Typerec}[\kappa]\ \forall\,[\kappa_1]\ \tau_1 \text{ of } \overline{\tau} \arrow[dl, "t_3"'] \arrow[dr, "T"] & \\
\begin{array}{l} \tau_{\forall}\ [\kappa_1]\ \tau_1 \\ \quad (\lambda\alpha\!:\!\kappa_1.\ \mathsf{Typerec}[\kappa]\ \tau_1\ \alpha \text{ of } \overline{\tau}) \end{array} \arrow[dr, dashed, "T^*"'] & & \mathsf{Typerec}[\kappa]\ \forall\,[\kappa_1]\ \tau_1' \text{ of } \overline{\tau} \arrow[dl, dashed, "t_3"] \\
& \begin{array}{l} \tau_{\forall}\ [\kappa_1]\ \tau_1' \\ \quad (\lambda\alpha\!:\!\kappa_1.\ \mathsf{Typerec}[\kappa]\ \tau_1'\ \alpha \text{ of } \overline{\tau}) \end{array} &
\end{tikzcd}
$$

We will also only show the reduction of $\tau_{\forall}$. The rest of the reductions are locally confluent in an obvious way. We use $\overline{\tau}$ to denote the types $(\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+})$. We use $\overline{\tau'}$ to denote the types $(\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall}';\ \tau_{\forall^+})$

$$
\begin{tikzcd}
& \mathsf{Typerec}[\kappa]\ \forall\,[\kappa_1]\ \tau_1 \text{ of } \overline{\tau} \arrow[dl, "t_3"'] \arrow[dr, "T"] & \\
\begin{array}{l} \tau_{\forall}\ [\kappa_1]\ \tau_1 \\ \quad (\lambda\alpha\!:\!\kappa_1.\ \mathsf{Typerec}[\kappa]\ \tau_1\ \alpha \text{ of } \overline{\tau}) \end{array} \arrow[dr, dashed, "T^*"'] & & \mathsf{Typerec}[\kappa]\ \forall\,[\kappa_1]\ \tau_1 \text{ of } \overline{\tau'} \arrow[dl, dashed, "t_3"] \\
& \begin{array}{l} \tau_{\forall}'\ [\kappa_1]\ \tau_1 \\ \quad (\lambda\alpha\!:\!\kappa_1.\ \mathsf{Typerec}[\kappa]\ \tau_1\ \alpha \text{ of } \overline{\tau'}) \end{array} &
\end{tikzcd}
$$

case t_4: There are five possible subcases from the reduction of either τ, τ_{int}, $\tau_{\to}$, $\tau_{\forall}$, or $\tau_{\forall^+}$. We first show the reduction of τ. We use $\overline{\tau}$ to denote the types $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$

$$
\begin{tikzcd}[column sep=small]
 & \mathsf{Typerec}[\kappa]\ \forall^{+}\tau\ \mathsf{of}\ \overline{\tau} \arrow[dl, "t_4"'] \arrow[dr, "T"] & \\
\tau_{\forall^+}\ \tau\ (\Lambda\chi.\ \mathsf{Typerec}[\kappa]\ \tau\ [\chi]\ \mathsf{of}\ \overline{\tau}) \arrow[dr, dashed, "T^*"'] & & \mathsf{Typerec}[\kappa]\ \forall^{+}\tau'\ \mathsf{of}\ \overline{\tau} \arrow[dl, dashed, "t_4"] \\
 & \tau_{\forall^+}\ \tau'\ (\Lambda\chi.\ \mathsf{Typerec}[\kappa]\ \tau'\ [\chi]\ \mathsf{of}\ \overline{\tau}) &
\end{tikzcd}
$$

We will also only show the reduction of $\tau_{\forall^+}$. The rest of the reductions are locally confluent in an obvious way. We use $\overline{\tau}$ to denote the types $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$. We use $\overline{\tau'}$ to denote the types $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau'_{\forall^+})$

$$
\begin{tikzcd}[column sep=small]
 & \mathsf{Typerec}[\kappa]\ \forall^{+}\tau\ \mathsf{of}\ \overline{\tau} \arrow[dl, "t_4"'] \arrow[dr, "T"] & \\
\tau_{\forall^+}\ \tau\ (\Lambda\chi.\ \mathsf{Typerec}[\kappa]\ \tau\ [\chi]\ \mathsf{of}\ \overline{\tau}) \arrow[dr, dashed, "T^*"'] & & \mathsf{Typerec}[\kappa]\ \forall^{+}\tau\ \mathsf{of}\ \overline{\tau'} \arrow[dl, dashed, "t_4"] \\
 & \tau'_{\forall^+}\ \tau\ (\Lambda\chi.\ \mathsf{Typerec}[\kappa]\ \tau\ [\chi]\ \mathsf{of}\ \overline{\tau'}) &
\end{tikzcd}
$$

$$(\lambda x\!:\!\tau.\,e)\,v \leadsto e\{v/x\} \qquad (\mathsf{fix}\,x\!:\!\tau.\,v)\,v' \leadsto (v\{\mathsf{fix}\,x\!:\!\tau.\,v/x\})\,v'$$

$$(\Lambda\alpha\!:\!\kappa.\,v)\,[\tau] \leadsto v\{\tau/\alpha\} \qquad (\mathsf{fix}\,x\!:\!\tau.\,v)\,[\tau] \leadsto (v\{\mathsf{fix}\,x\!:\!\tau.\,v/x\})\,[\tau]$$

$$(\Lambda^{+}\chi.\,v)\,[\kappa]^{+} \leadsto v\{\kappa/\chi\} \qquad (\mathsf{fix}\,x\!:\!\tau.\,v)\,[\kappa]^{+} \leadsto (v\{\mathsf{fix}\,x\!:\!\tau.\,v/x\})\,[\kappa]^{+}$$

$$\mathsf{unfold}\ (\mathsf{fold}\ v\ \mathsf{as}\ \tau)\ \mathsf{as}\ \tau \leadsto v$$

$$\mathsf{typecase}[\tau]\ \mathsf{int}\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \leadsto e_{\mathsf{int}}$$

$$\mathsf{typecase}[\tau]\ (\tau_1 \rightarrow \tau_2)\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \leadsto e_{\rightarrow}\,[\tau_1]\,[\tau_2]$$

$$\mathsf{typecase}[\tau]\ (\forall\,[\kappa]\,\tau')\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \leadsto e_{\forall}\,[\kappa]^{+}\,[\tau']$$

$$\mathsf{typecase}[\tau]\ (\forall^{+}\tau')\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \leadsto e_{\forall^{+}}\,[\tau']$$

$$\mathsf{typecase}[\tau]\ (\mu\tau')\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \leadsto e_{\mu}\,[\tau']$$

$$\frac{e \leadsto e_1}{e\,e' \leadsto e_1\,e'} \qquad \frac{e \leadsto e_1}{v\,e \leadsto v\,e_1} \qquad \frac{e \leadsto e_1}{e\,[\tau] \leadsto e_1\,[\tau]} \qquad \frac{e \leadsto e_1}{e\,[\kappa]^{+} \leadsto e_1\,[\kappa]^{+}}$$

$$\frac{e \leadsto e_1}{\mathsf{fold}\ e\ \mathsf{as}\ \tau \leadsto \mathsf{fold}\ e_1\ \mathsf{as}\ \tau} \qquad \frac{e \leadsto e_1}{\mathsf{unfold}\ e\ \mathsf{as}\ \tau \leadsto \mathsf{unfold}\ e_1\ \mathsf{as}\ \tau}$$

$$\frac{\varepsilon;\varepsilon \vdash \tau' \leadsto^{*} \nu' : \Omega \qquad \nu'\ \text{is normal form}}{\begin{array}{l}\mathsf{typecase}[\tau]\ \tau'\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \leadsto \\ \qquad \mathsf{typecase}[\tau]\ \nu'\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu})\end{array}}$$

Figure 29: Operational semantics of λ_i^Q

$$\begin{array}{llll}
(\mathit{value}) & v & ::= & i \mid \lambda x\!:\!\tau.\,e \mid \mathsf{fold}\ v\ \mathsf{as}\ \tau \mid \mathsf{unfold}\ v\ \mathsf{as}\ \tau \\
& & \mid & \Lambda\alpha\!:\!\kappa.\,v \mid \Lambda^{+}\chi.\,v \mid \mathsf{fix}\,x\!:\!\tau.\,v \\[1ex]
(\mathit{context}) & E & ::= & [\,] \mid E\,e \mid v\,E \mid E\,[\tau] \mid E\,[\kappa]^{+} \\
& & \mid & \mathsf{fold}\ E\ \mathsf{as}\ \tau \mid \mathsf{unfold}\ E\ \mathsf{as}\ \tau \\[1ex]
(\mathit{redex}) & r & ::= & (\lambda x\!:\!\tau.\,e)\,v \mid (\Lambda\alpha\!:\!\kappa.\,v)\,[\tau] \mid (\Lambda^{+}\chi.\,v)\,[\kappa]^{+} \\
& & \mid & (\mathsf{fix}\,x\!:\!\tau.\,v)\,v' \mid (\mathsf{fix}\,x\!:\!\tau.\,v)\,[\tau'] \\
& & \mid & (\mathsf{fix}\,x\!:\!\tau.\,v)\,[\kappa]^{+} \\
& & \mid & \mathsf{unfold}\ (\mathsf{fold}\ v\ \mathsf{as}\ \tau)\ \mathsf{as}\ \tau \\
& & \mid & \mathsf{typecase}[\tau]\ \tau'\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \\
& & \mid & \mathsf{typecase}[\tau]\ \mathsf{int}\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \\
& & \mid & \mathsf{typecase}[\tau]\ (\tau' \rightarrow \tau'')\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \\
& & \mid & \mathsf{typecase}[\tau]\ (\forall\,[\kappa]\,\tau')\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \\
& & \mid & \mathsf{typecase}[\tau]\ (\forall^{+}\tau')\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu}) \\
& & \mid & \mathsf{typecase}[\tau]\ (\mu\tau')\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu})
\end{array}$$

Figure 30: Term contexts

$$\begin{array}{lll}
\nu^0 & ::= & \alpha \mid \nu^0\,\nu \mid \nu^0\,[\kappa] \\
& \mid & \mathsf{Typerec}[\kappa]\ \nu^0\ \mathsf{of}\ (\nu_{\mathsf{int}};\, \nu_{\rightarrow};\, \nu_{\forall};\, \nu_{\forall^{+}}) \\
\nu & ::= & \nu^0 \mid \mathsf{int} \mid \mathsf{int}\,[\kappa] \mid \mathring{\twoheadrightarrow} \mid \mathring{\twoheadrightarrow}\,[\kappa] \mid (\mathring{\twoheadrightarrow})\,[\kappa]\,\nu \\
& \mid & (\mathring{\twoheadrightarrow})\,[\kappa]\,\nu\,\nu' \mid \mathring{\forall} \mid \mathring{\forall}\,[\kappa] \mid \mathring{\forall}\,[\kappa]\,[\kappa'] \mid \mathring{\forall}\,[\kappa]\,[\kappa']\,\nu \\
& \mid & \mathring{\forall}^{+} \mid \mathring{\forall}^{+}[\kappa] \mid \mathring{\forall}^{+}[\kappa]\,\nu \mid \mathring{\mu} \mid \mathring{\mu}\,[\kappa] \mid \mathring{\mu}\,[\kappa]\,\nu \\
& \mid & \mathsf{Place} \mid \mathsf{Place}\,[\kappa] \mid \mathsf{Place}\,[\kappa]\,\nu \\
& \mid & \lambda\alpha\!:\!\kappa.\,\nu,\ \text{where}\ \forall\nu^0.\ \nu \neq \nu^0\,\alpha\ \text{or}\ \alpha \in \mathit{ftv}(\nu^0) \\
& \mid & \Lambda\chi.\,\nu,\ \text{where}\ \forall\nu^0.\ \nu \neq \nu^0\,[\chi]\ \text{or}\ \chi \in \mathit{fkv}(\nu^0)
\end{array}$$

Figure 31: Normal forms in the λ_i^Q type language

C Properties of λ_i^Q

C.1 Soundness of λ_i^Q

Lemma C.1 (Normal form of types) *If $\varepsilon;\varepsilon \vdash \nu : \Omega$, then ν is one of* int, $\nu' \rightarrow \nu''$, $\forall\,[\kappa]\,\nu'$, $\forall^{+}\nu'$, *or* $\mu\nu'$.

Proof Since ν is kind checked in an empty environment, ν can not be a ν^0 since the head of a ν^0 is a type variable. From the kind, ν must be a int or of the form $\Lambda\chi.\,\nu_1$ and $\varepsilon,\chi;\varepsilon \vdash \nu_1 : \natural\chi$. From the kind, it is obvious that the only possible forms for ν_1 are $\mathsf{int}\,[\chi]$, $(\mathring{\twoheadrightarrow})\,[\chi]\,\nu_1'\,\nu_2'$, $\mathring{\forall}\,[\chi]\,[\kappa]\,\nu_1'$, $\mathring{\forall}^{+}[\chi]\,\nu_1'$, $\mathring{\mu}\,[\chi]\,\nu_1'$. It can not have a Place constructor because of the following reason. The only way it can have a Place constructor is if it is of the form $\mathsf{Place}\,[\chi]\,\nu_1'$. But this requires ν_1' to have the kind χ. This is not possible since none of the ν normal forms can have this kind and ν_1' can not have an occurence of ν^0 since the kinding is in an empty type environment.

The normal form $\Lambda\chi.\,(\mathsf{int}\,[\chi])$ is equivalent to int by eta reduction. The normal form $\Lambda\chi.\,((\mathring{\twoheadrightarrow})\,[\chi]\,\nu_1'\,\nu_2')$ is equivalent to $(\Lambda\chi.\,\nu_1') \rightarrow (\Lambda\chi.\,\nu_2')$. The normal form $\Lambda\chi.\,(\mathring{\forall}\,[\chi]\,[\kappa]\,\nu_1')$ is equivalent to $\forall\,[\kappa]\,(\lambda\alpha : \kappa.\,\Lambda\chi.\,\nu_1'\,\alpha)$. The normal form $\Lambda\chi.\,(\mathring{\forall}^{+}[\chi]\,\nu_1')$ is equivalent to $\forall^{+}(\Lambda\chi_1.\,\Lambda\chi.\,\nu_1'\,[\chi_1])$. The normal form $\Lambda\chi.\,(\mathring{\mu}\,[\chi]\,\nu_1')$ is equivalent to $\mu(\Lambda\chi.\,\nu_1')$. (See the rules at the bottom of Figure 11). $\square$

Lemma C.2 (Decomposition of terms) *If $\vdash e : \tau$, then e is either a value or can be decomposed into a unique E and a unique redex e' such that $e = E\,[e']$.*

Proof Proved by induction over the structure of e. Each of the cases follows similarly. We will consider only the interesting cases.

Suppose $e = e_1\,e_2$. By assumption, $\vdash e_1\,e_2 : \tau$. Therefore $\vdash e_1 : \tau_1 \rightarrow \tau$ and $\vdash e_2 : \tau_1$ for some type τ_1. Apply the inductive hypothesis now to e_1 and e_2. If both e_1 and e_2 are values v_1 and v_2, then the only possible reduction is $[\,]\,[v_1\,v_2]$. If $e_2 = E_2\,[e_2']$, then set E to be $v_1\,E_2$ and e' to be e_2'. If $e_1 = E_1\,[e_1']$, then set E to be $E_1\,e_2$ and e' to be e_1'.

Suppose $e = \mathsf{typecase}[\tau]\ \tau'\ \mathsf{of}\ (e_{\mathsf{int}};\, e_{\rightarrow};\, e_{\forall};\, e_{\forall^{+}};\, e_{\mu})$. If τ' is not a normal form, then E is the empty context and e is the redex. If τ' is a normal form, then by lemma C.1 e is still a redex and E is therefore the empty context. $\square$

Lemma C.3 *If $\vdash E\,[e] : \tau$, then there exists a τ' such that $\vdash e : \tau'$, and for all e' such that $\vdash e' : \tau'$ we have $\vdash E\,[e'] : \tau$.*

Proof The proof is by induction over the derivation of $\vdash E\,[e] : \tau$. All the cases are proved similarly. We will consider only one of the new cases.

Suppose $E = \mathsf{fold}\ E_1\ \mathsf{as}\ \tau$. Then we have that $\vdash E_1\,[e] : \tau_1$ for some type τ_1. Applying the inductive hypothesis to E_1, we get that there exists a τ' such that $\vdash e : \tau'$ and and for all e' of type τ', we have that $\vdash E_1\,[e'] : \tau_1$. $\square$

Corollary C.4 (Progress) *If $\vdash e : \tau$, then either e is a value or there exists an e_1 such that $e \mapsto e_1$.*

Proof By lemma C.2, we know that if $\vdash e : \tau$, then either e is a value or there exists an E and a redex e' such that $e = E\,[e']$. Since e' is a redex, there exists a reduct e'' such that $e' \leadsto e''$. Therefore, $e \mapsto e_1$ for $e_1 = E\,[e'']$.

We now prove a bunch of substitution lemmas.

Lemma C.5 *If $\mathcal{E}, \chi \vdash \kappa$ and $\mathcal{E} \vdash \kappa'$, then $\mathcal{E} \vdash \kappa\{\kappa'/\chi\}$.*

Lemma C.6 *If $\mathcal{E}; \chi; \Delta \vdash \tau : \kappa$ and $\mathcal{E} \vdash \kappa'$, then $\mathcal{E}; \Delta\{\kappa'/\chi\} \vdash \tau\{\kappa'/\chi\} : \kappa\{\kappa'/\chi\}$.*

Proof The proof is by induction over the structure of τ. All the cases follow in a straightforward manner by applying the inductive hypothesis to the subtypes. □

Lemma C.7 *If $\mathcal{E}; \Delta, \alpha : \kappa' \vdash \tau : \kappa$ and $\mathcal{E}; \Delta \vdash \tau' : \kappa'$, then $\mathcal{E}; \Delta \vdash \tau\{\tau'/\alpha\} : \kappa$.*

Proof The proof follows in a straightforward way by induction over the structure of τ. □

Lemma C.8 *If $\mathcal{E}; \Delta, \alpha : \kappa; \Gamma \vdash e : \tau$ and $\mathcal{E}; \Delta \vdash \tau' : \kappa$, then $\mathcal{E}; \Delta; \Gamma\{\tau'/\alpha\} \vdash e\{\tau'/\alpha\} : \tau\{\tau'/\alpha\}$.*

Proof The proof is by induction over the structure of e and is similar to the proof of this lemma for λ_i^P. □

Lemma C.9 *If $\mathcal{E}; \Delta; \Gamma, x : \tau' \vdash e : \tau$ and $\mathcal{E}; \Delta; \Gamma \vdash e' : \tau'$, then $\mathcal{E}; \Delta; \Gamma \vdash e\{e'/x\} : \tau$.*

Proof The proof is by induction over the structure of e and is similar to the proof of this lemma for λ_i^P. □

Lemma C.10 *If $\mathcal{E}, \chi; \Delta; \Gamma \vdash e : \tau$ and $\mathcal{E} \vdash \kappa$, then $\mathcal{E}; \Delta\{\kappa/\chi\}; \Gamma\{\kappa/\chi\} \vdash e\{\kappa/\chi\} : \tau\{\kappa/\chi\}$.*

Proof The proof follows in a straightforward way by induction over the structure of e and is similar to the proof of the other substitution lemmas. □

Definition C.11 *e evaluates to e' (written $e \mapsto e'$) if there exist E, e_1, and e_2 such that $e = E[e_1]$ and $e' = E[e_2]$ and $e_1 \rightsquigarrow e_2$.*

Theorem C.12 (Subject reduction) *If $\vdash e : \tau$ and $e \mapsto e'$, then $\vdash e' : \tau$.*

Proof By lemma C.2, we know that there exists a unique E and a unique redex e_1 such that $e = E[e_1]$. Since $e \mapsto e'$, there exists an e_1' such that $e' = E[e_1']$ and $e_1 \rightsquigarrow e_1'$. By lemma C.3, we know that for some τ_1 we have that $\vdash e_1 : \tau_1$. By the same lemma, we only need to prove that $\vdash e_1' : \tau_1$. We prove the theorem by considering each possible redex.

Suppose $e_1 = (\lambda x : \tau.\, e)\, v$. Then $e_1' = e\{v/x\}$. We know that $\varepsilon; \varepsilon; \varepsilon, x : \tau \vdash e : \tau'$ for some type τ' and $\varepsilon; \varepsilon; \varepsilon \vdash v : \tau$. Applying lemma C.9 leads to the result.

Suppose $e_1 = (\Lambda\alpha : \kappa.\, v)\, [\tau]$. Then $e_1' = v\{\tau/\alpha\}$. We know that $\varepsilon; \varepsilon, \alpha : \kappa; \varepsilon \vdash v : \tau'$ for some type τ' and $\varepsilon; \varepsilon \vdash \tau : \kappa$. Applying lemma C.8 leads to the result.

The case of $e_1 = (\Lambda^{+}\chi.\, e)\, [\kappa]^{+}$ is similar to the previous two cases and requires lemma C.10.

All of the fix reduction cases are proved similarly. We will consider only one case here. Suppose $e_1 = (\mathsf{fix}\, x : \tau.\, v)\, v'$. Then $e_1' = (v\{\mathsf{fix}\, x : \tau.\, v/x\})\, v'$. We have that $\vdash (\mathsf{fix}\, x : \tau.\, v)\, v' : \tau_1$. By the typing rules for term application we get that for some τ_2,

$\vdash \mathsf{fix}\, x : \tau.\, v : \tau_2 \to \tau_1$ and
$\vdash v' : \tau_2$

By the typing rule for fix we get that,

$\vdash \tau = \tau_2 \to \tau_1$ and
$\varepsilon; \varepsilon; \varepsilon, x : \tau_2 \to \tau_1 \vdash v : \tau_2 \to \tau_1$

Using Lemma C.9 and the typing rule for application, we obtain the desired judgment

$\vdash (v\{\mathsf{fix}\, x : \tau.\, v/x\})\, v' : \tau_1$

The unfold case follows trivially from the typing rules.

Suppose $e_1 = \mathsf{typecase}[\tau]\ \tau_1$ of $(e_{\mathsf{int}};\ e_{\to};\ e_{\forall};\ e_{\forall^+};\ e_{\mu})$. If τ_1 is in normal form ν_1, by the second premise of the typing rule for typecase and Lemma C.1 we have five cases for ν_1. In each case the contraction has the desired type $\tau\, \nu_1$, according to the corresponding premises of the typecase typing rule and the rules for type and kind applications. If τ_1 is not in normal form, then e_1 reduces to $\mathsf{typecase}[\tau]\ \nu_1$ of $(e_{\mathsf{int}};\ e_{\to};\ e_{\forall};\ e_{\forall^+};\ e_{\mu})$ where ν_1 is the corresponding normal form. Since the type system is strongly normalising, this reduction always terminates and since the type system is confluent, $\tau\, \tau_1 = \tau\, \nu_1$. □

$$
\begin{array}{llcl}
(\mathit{kinds}) & \kappa & ::= & \natural\kappa \mid \kappa \to \kappa' \mid \chi \mid \forall\chi.\,\kappa \\
(\mathit{types}) & \tau & ::= & \mathsf{int} \mid \mathring{\twoheadrightarrow} \mid \mathring{\forall} \mid \mathring{\forall}^{+} \mid \mathring{\mu} \mid \mathsf{Place} \\
 & & \mid & \alpha \mid \Lambda\chi.\,\tau \mid \lambda\alpha{:}\kappa.\,\tau \mid \tau\,[\kappa] \mid \tau\,\tau' \\
 & & \mid & \mathsf{Typerec}[\kappa]\ \tau\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})
\end{array}
$$

Figure 32: The λ_i^Q type language

$$
\begin{array}{lcl}
(\beta_1) & ::= & (\lambda\alpha{:}\kappa.\,\tau)\,\tau' \rightsquigarrow \tau\{\tau'/\alpha\} \\
(\beta_2) & ::= & (\Lambda\chi.\,\tau)\,[\kappa] \rightsquigarrow \tau\{\kappa/\chi\} \\
(\eta_1) & ::= & \lambda\alpha{:}\kappa.\,\tau\,\alpha \rightsquigarrow \tau \qquad \alpha \notin \mathit{ftv}(\tau) \\
(\eta_2) & ::= & \Lambda\chi.\,\tau\,[\chi] \rightsquigarrow \tau \qquad \chi \notin \mathit{fkv}(\tau) \\
(t_1) & ::= & \mathsf{Typerec}[\kappa]\ (\mathsf{int}\,[\kappa])\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu}) \rightsquigarrow \tau_{\mathsf{int}} \\
(t_2) & ::= & \mathsf{Typerec}[\kappa]\ (\mathring{\twoheadrightarrow}\,[\kappa]\tau_1\,\tau_2)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu}) \rightsquigarrow \\
 & & \quad \tau_{\to}\,\tau_1\,\tau_2 \\
 & & \quad (\mathsf{Typerec}[\kappa]\ \tau_1\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})) \\
 & & \quad (\mathsf{Typerec}[\kappa]\ \tau_2\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})) \\
(t_3) & ::= & \mathsf{Typerec}[\kappa]\ (\mathring{\forall}\,[\kappa]\,[\kappa']\,\tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu}) \rightsquigarrow \\
 & & \quad \tau_{\forall}\,[\kappa']\,\tau \\
 & & \quad (\lambda\alpha{:}\kappa'. \\
 & & \qquad \mathsf{Typerec}[\kappa]\ (\tau\,\alpha)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})) \\
(t_4) & ::= & \mathsf{Typerec}[\kappa]\ (\mathring{\forall}^{+}[\kappa]\,\tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu}) \rightsquigarrow \\
 & & \quad \tau_{\forall^+}\,\tau \\
 & & \quad (\Lambda\chi.\ \mathsf{Typerec}[\kappa]\ \tau\,[\chi]\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})) \\
(t_5) & ::= & \mathsf{Typerec}[\kappa]\ (\mathring{\mu}\,[\kappa]\tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu}) \rightsquigarrow \\
 & & \quad \tau_{\mu}\,\tau \\
 & & \quad (\lambda\alpha{:}\kappa.\ \mathsf{Typerec}[\kappa]\ (\tau\ (\mathsf{Place}\,[\kappa]\,\alpha))\ \mathsf{of} \\
 & & \qquad\qquad (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})) \\
(t_6) & ::= & \mathsf{Typerec}[\kappa]\ (\mathsf{Place}\,[\kappa]\,\tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu}) \rightsquigarrow \\
 & & \quad \tau
\end{array}
$$

Figure 33: Type reductions

C.2 Strong Normalisation of λ_i^Q

The type language is shown in Figure 32. The single step reduction relation ($\tau \rightsquigarrow \tau'$) is shown in Figure 33.

Lemma C.13 *If $\mathcal{E};\Delta \vdash \tau : \kappa$ and $\tau \rightsquigarrow \tau'$, then $\mathcal{E};\Delta \vdash \tau' : \kappa$.*

Proof (Sketch) The proof follows from a case analysis of the reduction relation ($\rightsquigarrow$). $\square$

Lemma C.14 *If $\tau_1 \rightsquigarrow \tau_2$, then $\tau_1\{\tau/\alpha\} \rightsquigarrow \tau_2\{\tau/\alpha\}$.*

Proof The proof is by enumerating each possible reduction from τ_1 to τ_2. We will only show the cases that are different from λ_i^P.

case t_1: $\tau_1 = \mathsf{Typerec}[\kappa]\ (\mathsf{int}\,[\kappa])$ of $(\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})$ and $\tau_2 = \tau_{\mathsf{int}}$. We get that

$$\begin{array}{l}\tau_1\{\tau/\alpha\} = \\ \mathsf{Typerec}[\kappa]\ (\mathsf{int}\,[\kappa])\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\to}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\};\ \tau_{\mu}\{\tau/\alpha\})\end{array}$$

But this reduces by the t_1 reduction to $\tau_{\mathsf{int}}\{\tau/\alpha\}$.

case t_2: $\tau_1 = \mathsf{Typerec}[\kappa]\ ((\overset{\circ}{\twoheadrightarrow})\ [\kappa]\tau'\tau'')$ of $(\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})$ and

$$\begin{array}{l}\tau_2 = \tau_{\to}\ \tau'\ \tau''\ (\mathsf{Typerec}[\kappa]\ \tau'\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})) \\ \qquad\qquad (\mathsf{Typerec}[\kappa]\ \tau''\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu}))\end{array}$$

We get that

$$\begin{array}{l}\tau_1\{\tau/\alpha\} = \\ \mathsf{Typerec}[\kappa]\ ((\overset{\circ}{\twoheadrightarrow})\ [\kappa](\tau'\{\tau/\alpha\})(\tau''\{\tau/\alpha\}))\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\to}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\};\ \tau_{\mu}\{\tau/\alpha\})\end{array}$$

This reduces by t_2 to

$$\begin{array}{l}\tau_{\to}\{\tau/\alpha\}\ (\tau'\{\tau/\alpha\})\ (\tau''\{\tau/\alpha\}) \\ \ (\mathsf{Typerec}[\kappa]\ (\tau'\{\tau/\alpha\})\ \mathsf{of} \\ \qquad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\to}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\};\ \tau_{\mu}\{\tau/\alpha\})) \\ \ (\mathsf{Typerec}[\kappa]\ (\tau''\{\tau/\alpha\})\ \mathsf{of} \\ \qquad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\to}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\};\ \tau_{\mu}\{\tau/\alpha\}))\end{array}$$

But this is syntactically equal to $\tau_2\{\tau/\alpha\}$.

case t_3: $\tau_1 = \mathsf{Typerec}[\kappa]\ (\mathring{\forall}\,[\kappa]\,[\kappa']\,\tau')$ of $(\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})$ and

$$\tau_2 = \tau_{\forall}\ [\kappa']\ \tau'\ (\lambda\beta\!:\!\kappa'.\ \mathsf{Typerec}[\kappa]\ (\tau'\ \beta)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu}))$$

We get that

$$\begin{array}{l}\tau_1\{\tau/\alpha\} = \\ \mathsf{Typerec}[\kappa]\ (\mathring{\forall}\,[\kappa]\,[\kappa']\,\tau'\{\tau/\alpha\})\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\to}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\};\ \tau_{\mu}\{\tau/\alpha\})\end{array}$$

This reduces by t_3 to

$$\begin{array}{l}\tau_{\forall}\{\tau/\alpha\}\ [\kappa']\ (\tau'\{\tau/\alpha\}) \\ \ (\lambda\beta\!:\!\kappa'.\ \mathsf{Typerec}[\kappa]\ ((\tau'\{\tau/\alpha\})\ \beta)\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\to}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\};\ \tau_{\mu}\{\tau/\alpha\}))\end{array}$$

But this is syntactically equivalent to $\tau_2\{\tau/\alpha\}$.

case t_4: $\tau_1 = \mathsf{Typerec}[\kappa]\ (\mathring{\forall}^+[\kappa]\,\tau')$ of $(\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})$ and

$$\tau_2 = \tau_{\forall^+}\ \tau'\ (\Lambda\chi.\ \mathsf{Typerec}[\kappa]\ (\tau'\ [\chi])\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu}))$$

We get that

$$\begin{array}{l}\tau_1\{\tau/\alpha\} = \\ \mathsf{Typerec}[\kappa]\ (\mathring{\forall}^+[\kappa]\,\tau'\{\tau/\alpha\})\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\to}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\};\ \tau_{\mu}\{\tau/\alpha\})\end{array}$$

This reduces by t_4 to

$$\begin{array}{l}\tau_{\forall^+}\{\tau/\alpha\}\ (\tau'\{\tau/\alpha\}) \\ \ (\Lambda\chi.\ \mathsf{Typerec}[\kappa]\ ((\tau'\{\tau/\alpha\})\ [\chi])\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\to}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\};\ \tau_{\mu}\{\tau/\alpha\}))\end{array}$$

But this is syntactically equal to $\tau_2\{\tau/\alpha\}$.

case t_5: $\tau_1 = \mathsf{Typerec}[\kappa]\ (\mathring{\mu}\,[\kappa]\,\tau')$ of $(\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})$ and

$$\begin{array}{l}\tau_2 = \tau_{\mu}\ \tau' \\ (\lambda\beta\!:\!\kappa.\ \mathsf{Typerec}[\kappa]\ (\tau'\ (\mathsf{Place}\,[\kappa]\,\beta))\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu}))\end{array}$$

We get that

$$\begin{array}{l}\tau_1\{\tau/\alpha\} = \\ \mathsf{Typerec}[\kappa]\ (\mathring{\mu}\,[\kappa]\,\tau'\{\tau/\alpha\})\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\to}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\};\ \tau_{\mu}\{\tau/\alpha\})\end{array}$$

This reduces by t_5 to

$$\begin{array}{l}\tau_{\mu}\{\tau/\alpha\}\ (\tau'\{\tau/\alpha\}) \\ \ (\lambda\beta\!:\!\kappa.\ \mathsf{Typerec}[\kappa]\ ((\tau'\{\tau/\alpha\})\ (\mathsf{Place}\,[\kappa]\,\beta))\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\to}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\};\ \tau_{\mu}\{\tau/\alpha\}))\end{array}$$

But this is syntactically equal to $\tau_2\{\tau/\alpha\}$.

case t_6: $\tau_1 = \mathsf{Typerec}[\kappa]\ (\mathsf{Place}\,[\kappa]\,\tau')$ of $(\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})$ and $\tau_2 = \tau'$. We get that

$$\begin{array}{l}\tau_1\{\tau/\alpha\} = \\ \mathsf{Typerec}[\kappa]\ (\mathsf{Place}\,[\kappa]\,\tau'\{\tau/\alpha\})\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\tau/\alpha\};\ \tau_{\to}\{\tau/\alpha\};\ \tau_{\forall}\{\tau/\alpha\};\ \tau_{\forall^+}\{\tau/\alpha\};\ \tau_{\mu}\{\tau/\alpha\})\end{array}$$

This reduces by t_6 to $\tau'\{\tau/\alpha\}$. $\square$

Lemma C.15 *If $\tau_1 \rightsquigarrow \tau_2$, then $\tau_1\{\kappa'/\chi'\} \rightsquigarrow \tau_2\{\kappa'/\chi'\}$.*

Proof This is proved by case analysis of the type reduction relation. We will only show the cases that are different from λ_i^P.

case t_1: $\tau_1 = \mathsf{Typerec}[\kappa]\ (\mathsf{int}\,[\kappa])$ of $(\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})$ and $\tau_2 = \tau_{\mathsf{int}}$. We get that

$$\begin{array}{l}\tau_1\{\kappa'/\chi'\} = \\ \mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ (\mathsf{int}\,[\kappa\{\kappa'/\chi'\}])\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\kappa'/\chi'\};\ \tau_{\to}\{\kappa'/\chi'\};\ \tau_{\forall}\{\kappa'/\chi'\};\ \tau_{\forall^+}\{\kappa'/\chi'\};\ \tau_{\mu}\{\kappa'/\chi'\})\end{array}$$

But this reduces by the t_1 reduction to $\tau_{\mathsf{int}}\{\kappa'/\chi'\}$.

case t_2: $\tau_1 = \mathsf{Typerec}[\kappa]\ ((\overset{\circ}{\twoheadrightarrow})\ [\kappa]\tau'\tau'')$ of $(\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})$ and

$$\begin{array}{l}\tau_2 = \tau_{\to}\ \tau'\ \tau''\ (\mathsf{Typerec}[\kappa]\ \tau'\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})) \\ \qquad\qquad (\mathsf{Typerec}[\kappa]\ \tau''\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\to};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu}))\end{array}$$

We get that

$$\begin{array}{l}\tau_1\{\kappa'/\chi'\} = \\ \mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ ((\overset{\circ}{\twoheadrightarrow})\ [\kappa\{\kappa'/\chi'\}]\tau'\{\kappa'/\chi'\}\tau''\{\kappa'/\chi'\})\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\kappa'/\chi'\};\ \tau_{\to}\{\kappa'/\chi'\};\ \tau_{\forall}\{\kappa'/\chi'\};\ \tau_{\forall^+}\{\kappa'/\chi'\};\ \tau_{\mu}\{\kappa'/\chi'\})\end{array}$$

This reduces by t_2 to

$$\begin{array}{l}\tau_{\to}\{\kappa'/\chi'\}\ (\tau'\{\kappa'/\chi'\})\ (\tau''\{\kappa'/\chi'\}) \\ \ (\mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ (\tau'\{\kappa'/\chi'\})\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\kappa'/\chi'\};\ \tau_{\to}\{\kappa'/\chi'\};\ \tau_{\forall}\{\kappa'/\chi'\};\ \tau_{\forall^+}\{\kappa'/\chi'\};\ \tau_{\mu}\{\kappa'/\chi'\})) \\ \ (\mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ (\tau''\{\kappa'/\chi'\})\ \mathsf{of} \\ \quad (\tau_{\mathsf{int}}\{\kappa'/\chi'\};\ \tau_{\to}\{\kappa'/\chi'\};\ \tau_{\forall}\{\kappa'/\chi'\};\ \tau_{\forall^+}\{\kappa'/\chi'\};\ \tau_{\mu}\{\kappa'/\chi'\}))\end{array}$$

But this is syntactically equal to $\tau_2\{\kappa'/\chi'\}$.

case t_3: $\tau_1 = \mathsf{Typerec}[\kappa]\ (\mathring{\forall}\,[\kappa]\,[\kappa_1]\,\tau')$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})$ and

$$\begin{aligned}\tau_2 = {} & \tau_{\forall}\,[\kappa_1]\,\tau' \\ & (\lambda\beta{:}\kappa_1.\ \mathsf{Typerec}[\kappa]\ (\tau'\,\beta)\ \text{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu}))\end{aligned}$$

We get that

$$\begin{aligned}&\tau_1\{\kappa'/\chi'\} = \\ &\mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ (\mathring{\forall}\,[\kappa\{\kappa'/\chi'\}]\,[\kappa_1\{\kappa'/\chi'\}]\,\tau'\{\kappa'/\chi'\})\ \text{of} \\ &\quad (\tau_{\mathsf{int}}\{\kappa'/\chi'\};\ \tau_{\rightarrow}\{\kappa'/\chi'\};\ \tau_{\forall}\{\kappa'/\chi'\};\ \tau_{\forall^+}\{\kappa'/\chi'\};\ \tau_{\mu}\{\kappa'/\chi'\})\end{aligned}$$

This reduces by t_3 to

$$\begin{aligned}&\tau_{\forall}\{\kappa'/\chi'\}\,[\kappa_1\{\kappa'/\chi'\}]\,(\tau'\{\kappa'/\chi'\}) \\ &\ (\lambda\beta{:}\kappa_1\{\kappa'/\chi'\}.\ \mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ ((\tau'\{\kappa'/\chi'\})\,\beta)\ \text{of} \\ &\ (\tau_{\mathsf{int}}\{\kappa'/\chi'\};\ \tau_{\rightarrow}\{\kappa'/\chi'\};\ \tau_{\forall}\{\kappa'/\chi'\};\ \tau_{\forall^+}\{\kappa'/\chi'\};\ \tau_{\mu}\{\kappa'/\chi'\}))\end{aligned}$$

But this is syntactically equivalent to $\tau_2\{\kappa'/\chi'\}$.

case t_4: $\tau_1 = \mathsf{Typerec}[\kappa]\ (\mathring{\forall}^+[\kappa]\,\tau')$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})$ and

$$\tau_2 = \tau_{\forall^+}\,\tau'\ (\Lambda\chi.\ \mathsf{Typerec}[\kappa]\ (\tau'\,[\chi])\ \text{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu}))$$

We get that

$$\begin{aligned}&\tau_1\{\kappa'/\chi'\} = \\ &\mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ (\mathring{\forall}^+[\kappa\{\kappa'/\chi'\}]\,\tau'\{\kappa'/\chi'\})\ \text{of} \\ &\quad (\tau_{\mathsf{int}}\{\kappa'/\chi'\};\ \tau_{\rightarrow}\{\kappa'/\chi'\};\ \tau_{\forall}\{\kappa'/\chi'\};\ \tau_{\forall^+}\{\kappa'/\chi'\};\ \tau_{\mu}\{\kappa'/\chi'\})\end{aligned}$$

This reduces by t_4 to

$$\begin{aligned}&\tau_{\forall^+}\{\kappa'/\chi'\}\,(\tau'\{\kappa'/\chi'\}) \\ &\ (\Lambda\chi.\ \mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ ((\tau'\{\kappa'/\chi'\})\,[\chi])\ \text{of} \\ &\ (\tau_{\mathsf{int}}\{\kappa'/\chi'\};\ \tau_{\rightarrow}\{\kappa'/\chi'\};\ \tau_{\forall}\{\kappa'/\chi'\};\ \tau_{\forall^+}\{\kappa'/\chi'\};\ \tau_{\mu}\{\kappa'/\chi'\}))\end{aligned}$$

But this is syntactically equal to $\tau_2\{\kappa'/\chi'\}$.

case t_5: $\tau_1 = \mathsf{Typerec}[\kappa]\ (\mathring{\mu}\,[\kappa]\,\tau')$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})$ and

$$\begin{aligned}\tau_2 = {} & \tau_{\mu}\,\tau' \\ & (\lambda\alpha{:}\kappa.\ \mathsf{Typerec}[\kappa]\ (\tau'\,(\mathsf{Place}\,[\kappa]\,\alpha))\ \text{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu}))\end{aligned}$$

We get that

$$\begin{aligned}&\tau_1\{\kappa'/\chi'\} = \\ &\mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ (\mathring{\mu}\,[\kappa\{\kappa'/\chi'\}]\,\tau'\{\kappa'/\chi'\})\ \text{of} \\ &\quad (\tau_{\mathsf{int}}\{\kappa'/\chi'\};\ \tau_{\rightarrow}\{\kappa'/\chi'\};\ \tau_{\forall}\{\kappa'/\chi'\};\ \tau_{\forall^+}\{\kappa'/\chi'\};\ \tau_{\mu}\{\kappa'/\chi'\})\end{aligned}$$

This reduces by t_5 to

$$\begin{aligned}&\tau_{\mu}\{\kappa'/\chi'\}\,(\tau'\{\kappa'/\chi'\}) \\ &\ (\lambda\alpha{:}\kappa\{\kappa'/\chi'\}. \\ &\qquad \mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ ((\tau'\{\kappa'/\chi'\})\,(\mathsf{Place}\,[\kappa\{\kappa'/\chi'\}]\,\alpha))\ \text{of} \\ &\ (\tau_{\mathsf{int}}\{\kappa'/\chi'\};\ \tau_{\rightarrow}\{\kappa'/\chi'\};\ \tau_{\forall}\{\kappa'/\chi'\};\ \tau_{\forall^+}\{\kappa'/\chi'\};\ \tau_{\mu}\{\kappa'/\chi'\}))\end{aligned}$$

But this is syntactically equal to $\tau_2\{\kappa'/\chi'\}$.

case t_6: $\tau_1 = \mathsf{Typerec}[\kappa]\ (\mathsf{Place}\,[\kappa]\,\tau')$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})$ and $\tau_2 = \tau'$. We get that

$$\begin{aligned}&\tau_1\{\kappa'/\chi'\} = \\ &\mathsf{Typerec}[\kappa\{\kappa'/\chi'\}]\ (\mathsf{Place}\,[\kappa\{\kappa'/\chi'\}]\,\tau'\{\kappa'/\chi'\})\ \text{of} \\ &\quad (\tau_{\mathsf{int}}\{\kappa'/\chi'\};\ \tau_{\rightarrow}\{\kappa'/\chi'\};\ \tau_{\forall}\{\kappa'/\chi'\};\ \tau_{\forall^+}\{\kappa'/\chi'\};\ \tau_{\mu}\{\kappa'/\chi'\})\end{aligned}$$

This reduces by t_6 to $\tau'\{\kappa'/\chi'\}$. □

Definition C.16 *A type τ is strongly normalisable if every reduction sequence from τ terminates. We use $\nu(\tau)$ to denote the length of the largest reduction sequence from τ to a normal form.*

Definition C.17 *We define neutral types, n, as*

$$\begin{aligned} n_0 &::= \Lambda\chi.\,\tau \mid \lambda\alpha{:}\kappa.\,\tau \\ n &::= \alpha \mid n_0\,\tau \mid n\,\tau \mid n_0\,[\kappa] \mid n\,[\kappa] \\ &\quad\mid \mathsf{Typerec}[\kappa]\ \tau\ \text{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+})\end{aligned}$$

Definition C.18 *A reducibility candidate (also referred to as a candidate) of kind κ is a set $\mathcal{C}$ of types of kind κ such that*

1. *if $\tau \in \mathcal{C}$, then τ is strongly normalisable.*
2. *if $\tau \in \mathcal{C}$ and $\tau \leadsto \tau'$, then $\tau' \in \mathcal{C}$.*
3. *if τ is neutral and if for all τ' such that $\tau \leadsto \tau'$, we have that $\tau' \in \mathcal{C}$, then $\tau \in \mathcal{C}$.*

This implies that the candidates are never empty since if α has kind κ, then α belongs to candidates of kind κ.

Definition C.19 *Let κ be an arbitrary kind. Let $\mathcal{C}_\kappa$ be a candidate of kind κ. Let $\mathcal{C}_{\natural\kappa\rightarrow\natural\kappa\rightarrow\kappa\rightarrow\kappa\rightarrow\kappa}$ be a candidate of kind $\natural\kappa \rightarrow \natural\kappa \rightarrow \kappa \rightarrow \kappa \rightarrow \kappa$. Let $\mathcal{C}_{\forall\chi.\,(\chi\rightarrow\natural\kappa)\rightarrow(\chi\rightarrow\kappa)\rightarrow\kappa}$ be a candidate of kind $\forall\chi.\,(\chi \rightarrow \natural\kappa) \rightarrow (\chi \rightarrow \kappa) \rightarrow \kappa$. Let $\mathcal{C}_{(\forall\chi.\,\natural\kappa)\rightarrow(\forall\chi.\,\kappa)\rightarrow\kappa}$ be a candidate of kind $(\forall\chi.\,\natural\kappa) \rightarrow (\forall\chi.\,\kappa) \rightarrow \kappa$. Let $\mathcal{C}_{(\natural\kappa\rightarrow\natural\kappa)\rightarrow(\kappa\rightarrow\kappa)\rightarrow\kappa}$ be a candidate of kind $(\natural\kappa \rightarrow \natural\kappa) \rightarrow (\kappa \rightarrow \kappa) \rightarrow \kappa$. We then define the set $\natural_R\mathcal{C}_\kappa$ of types of kind $\natural\kappa$ as*

$$\begin{aligned}&\tau \in \natural_R\mathcal{C}_\kappa \quad \textit{iff} \\ &\quad \forall\tau_{\mathsf{int}} \in \mathcal{C}_\kappa, \\ &\quad \forall\tau_{\rightarrow} \in \mathcal{C}_{\natural\kappa\rightarrow\natural\kappa\rightarrow\kappa\rightarrow\kappa\rightarrow\kappa}, \\ &\quad \forall\tau_{\forall} \in \mathcal{C}_{\forall\chi.\,(\chi\rightarrow\natural\kappa)\rightarrow(\chi\rightarrow\kappa)\rightarrow\kappa}, \\ &\quad \forall\tau_{\forall^+} \in \mathcal{C}_{(\forall\chi.\,\natural\kappa)\rightarrow(\forall\chi.\,\kappa)\rightarrow\kappa} \\ &\quad \forall\tau_{\mu} \in \mathcal{C}_{(\natural\kappa\rightarrow\natural\kappa)\rightarrow(\kappa\rightarrow\kappa)\rightarrow\kappa} \\ &\qquad \Rightarrow \mathsf{Typerec}[\kappa]\ \tau\ \text{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu}) \in \mathcal{C}_\kappa\end{aligned}$$

Lemma C.20 *If $\mathcal{C}_\kappa$ is a candidate of kind κ, then $\natural_R\mathcal{C}_\kappa$ is a candidate of kind $\natural\kappa$.*

Proof Suppose $\tau \in \natural_R\mathcal{C}_\kappa$. Suppose τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, $\tau_{\forall^+}$, and τ_μ belong to $\mathcal{C}_\kappa$, $\mathcal{C}_{\natural\kappa\rightarrow\natural\kappa\rightarrow\kappa\rightarrow\kappa\rightarrow\kappa}$, $\mathcal{C}_{\forall\chi.\,(\chi\rightarrow\natural\kappa)\rightarrow(\chi\rightarrow\kappa)\rightarrow\kappa}$, $\mathcal{C}_{(\forall\chi.\,\natural\kappa)\rightarrow(\forall\chi.\,\kappa)\rightarrow\kappa}$, and $\mathcal{C}_{(\natural\kappa\rightarrow\natural\kappa)\rightarrow(\kappa\rightarrow\kappa)\rightarrow\kappa}$ respectively, where the candidates are of the appropriate kinds (see definition C.19).

Consider $\tau' = \mathsf{Typerec}[\kappa]\ \tau$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})$. By definition this belongs to $\mathcal{C}_\kappa$. By property 1 of definition C.18, τ' is strongly normalisable and therefore τ must be strongly normalisable.

Consider $\tau' = \mathsf{Typerec}[\kappa]\ \tau$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})$. Suppose $\tau \leadsto \tau_1$. Then $\tau' \leadsto \mathsf{Typerec}[\kappa]\ \tau_1$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})$. Since $\tau' \in \mathcal{C}_\kappa$, $\mathsf{Typerec}[\kappa]\ \tau_1$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})$ belongs to $\mathcal{C}_\kappa$ by property 2 of definition C.18. Therefore, by definition, τ_1 belongs to $\natural_R\mathcal{C}_\kappa$.

Suppose τ is neutral and for all τ_1 such that $\tau \leadsto \tau_1$, $\tau_1 \in \natural_R\mathcal{C}_\kappa$. Consider $\tau' = \mathsf{Typerec}[\kappa]\ \tau$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})$. Since we know that τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, $\tau_{\forall^+}$, and τ_μ are strongly normalisable, we can induct over $len = \nu(\tau_{\mathsf{int}}) + \nu(\tau_{\rightarrow}) + \nu(\tau_{\forall}) + \nu(\tau_{\forall^+}) + \nu(\tau_\mu)$. We will prove that for all values of len, the type $\mathsf{Typerec}[\kappa]\ \tau$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_{\mu})$ always reduces to a type that belongs to $\mathcal{C}_\kappa$; given that $\tau_{\mathsf{int}} \in \mathcal{C}_\kappa$, and $\tau_{\rightarrow} \in \mathcal{C}_{\natural\kappa\rightarrow\natural\kappa\rightarrow\kappa\rightarrow\kappa\rightarrow\kappa}$, and $\tau_{\forall} \in \mathcal{C}_{\forall\chi.\,(\chi\rightarrow\natural\chi)\rightarrow(\chi\rightarrow\kappa)\rightarrow\kappa}$, and $\tau_{\forall^+} \in \mathcal{C}_{(\forall\chi.\,\natural\kappa)\rightarrow(\forall\chi.\,\kappa)\rightarrow\kappa}$, and $\tau_\mu \in \mathcal{C}_{(\natural\kappa\rightarrow\natural\kappa)\rightarrow(\kappa\rightarrow\kappa)\rightarrow\kappa}$.

- $len = 0$ Then $\tau' \rightsquigarrow \mathsf{Typerec}[\kappa]\, \tau_1$ of $(\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+};\, \tau_\mu)$ is the only possible reduction since τ is neutral. By the assumption on τ_1, this belongs to $\mathcal{C}_\kappa$.

- $len = k + 1$ For the inductive case, assume that the hypothesis is true for $len = k$. That is, for $len = k$, the type $\mathsf{Typerec}[\kappa]\, \tau$ of $(\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+};\, \tau_\mu)$ always reduces to a type that belongs to $\mathcal{C}_\kappa$; given that τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, $\tau_{\forall^+}$, and τ_μ belong to $\mathcal{C}_\kappa$, $\mathcal{C}_{\natural\kappa \to \natural\kappa \to \kappa \to \kappa \to \kappa}$, $\mathcal{C}_{\forall\chi.\,(\chi \to \natural\chi) \to (\chi \to \kappa) \to \kappa}$, $\mathcal{C}_{(\natural\kappa \to \natural\kappa) \to (\kappa \to \kappa) \to \kappa}$, and $\mathcal{C}_{(\natural\kappa \to \natural\kappa) \to (\kappa \to \kappa) \to \kappa}$ respectively. By property 3 of definition C.18, $\mathsf{Typerec}[\kappa]\, \tau$ of $(\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+};\, \tau_\mu)$ belongs to $\mathcal{C}_\kappa$ for $len = k$. Consider $\tau' = \mathsf{Typerec}[\kappa]\, \tau$ of $(\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+};\, \tau_\mu)$ for $len = k + 1$. This can reduce to $\mathsf{Typerec}[\kappa]\, \tau_1$ of $(\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+};\, \tau_\mu)$ which belongs to $\mathcal{C}_\kappa$. The other possible reductions are to $\mathsf{Typerec}[\kappa]\, \tau$ of $(\tau'_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+};\, \tau_\mu)$ where $\tau_{\mathsf{int}} \rightsquigarrow \tau'_{\mathsf{int}}$, or to $\mathsf{Typerec}[\kappa]\, \tau$ of $(\tau_{\mathsf{int}};\, \tau'_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+};\, \tau_\mu)$ where $\tau_{\rightarrow} \rightsquigarrow \tau'_{\rightarrow}$, or to $\mathsf{Typerec}[\kappa]\, \tau$ of $(\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau'_{\forall};\, \tau_{\forall^+};\, \tau_\mu)$ where $\tau_{\forall} \rightsquigarrow \tau'_{\forall}$, or to $\mathsf{Typerec}[\kappa]\, \tau$ of $(\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau'_{\forall^+};\, \tau_\mu)$ where $\tau_{\forall^+} \rightsquigarrow \tau'_{\forall^+}$, or to $\mathsf{Typerec}[\kappa]\, \tau$ of $(\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+};\, \tau'_\mu)$ where $\tau_\mu \rightsquigarrow \tau'_\mu$. By property 2 of definition C.18, each of τ'_{int}, $\tau'_{\rightarrow}$, $\tau'_{\forall}$, $\tau'_{\forall^+}$, and τ'_μ belong to the same candidate as before. Moreover, $len = k$ for each of the reducts. By the inductive hypothesis, each of the reducts belongs to $\mathcal{C}_\kappa$.

Therefore, by property 3 of definition C.18, $\mathsf{Typerec}[\kappa]\, \tau$ of $(\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+};\, \tau_\mu)$ belongs to $\mathcal{C}_\kappa$. Therefore, $\tau \in \natural_R \mathcal{C}_\kappa$. □

Definition C.21 *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two candidates of kinds κ_1 and κ_2. We then define the set $\mathcal{C}_1 \to \mathcal{C}_2$, of types of kind $\kappa_1 \to \kappa_2$, as*

$$\tau \in \mathcal{C}_1 \to \mathcal{C}_2 \quad \textit{iff} \quad \forall \tau' (\tau' \in \mathcal{C}_1 \Rightarrow \tau\, \tau' \in \mathcal{C}_2)$$

Lemma C.22 *If $\mathcal{C}_1$ and $\mathcal{C}_2$ are candidates of kinds κ_1 and κ_2, then $\mathcal{C}_1 \to \mathcal{C}_2$ is a candidate of kind $\kappa_1 \to \kappa_2$.*

Proof Same as lemma B.22 for λ_i^P. □

Definition C.23 *We use $\overline{\chi}$ to denote the set $\chi_1, \ldots, \chi_n$ of χ. We use a similar syntax to denote a set of other constructs.*

Definition C.24 *Let $\kappa[\overline{\chi}]$ be a kind where $\overline{\chi}$ contains all the free kind variables of κ. Let $\overline{\kappa}$ be a sequence of closed kinds of the same length and $\overline{\mathcal{C}}$ be a sequence of candidates of the corresponding kind. We now define the set $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$ of types of kind $\kappa\{\overline{\kappa}/\overline{\chi}\}$ as*

1. *if $\kappa = \natural\kappa'$, then $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}] = \natural_R \mathcal{S}_{\kappa'}[\overline{\mathcal{C}}/\overline{\chi}]$.*
2. *if $\kappa = \chi_i$, then $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}] = \mathcal{C}_i$.*
3. *if $\kappa = \kappa_1 \to \kappa_2$, then $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}] = \mathcal{S}_{\kappa_1}[\overline{\mathcal{C}}/\overline{\chi}] \to \mathcal{S}_{\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$.*
4. *if $\kappa = \forall\chi.\,\kappa'$, then $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$ = the set of types τ of kind $\kappa\{\overline{\kappa}/\overline{\chi}\}$ such that for every kind κ'' and reducibility candidate $\mathcal{C}''$ of this kind, $\tau\,[\kappa''] \in \mathcal{S}_{\kappa'}[\overline{\mathcal{C}}, \mathcal{C}''/\overline{\chi}, \chi]$.*

Lemma C.25 *$\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$ is a reducibility candidate of kind $\kappa\{\overline{\kappa}/\overline{\chi}\}$.*

Proof For $\kappa = \natural\kappa'$, the lemma follows from the inductive hypothesis on κ' and lemma C.20. The rest of the proof is the same as lemma B.25 for λ_i^P. □

Lemma C.26 $\mathcal{S}_{\kappa\{\kappa'/\chi'\}}[\overline{\mathcal{C}}/\overline{\chi}] = \mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{S}_{\kappa'}[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi']$

Proof The proof is by induction over the structure of κ. Suppose $\kappa = \natural\kappa_1$. Then the LHS is equal to $\natural_R \mathcal{S}_{\kappa_1\{\kappa'/\chi'\}}[\overline{\mathcal{C}}/\overline{\chi}]$. By the inductive hypothesis on κ_1, this is equal to $\natural_R \mathcal{S}_{\kappa_1}[\overline{\mathcal{C}}, \mathcal{S}_{\kappa'}[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi']$. By definition, the RHS is equal to $\natural_R \mathcal{S}_{\kappa_1}[\overline{\mathcal{C}}, \mathcal{S}_{\kappa'}[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi']$.

The other cases are the same as lemma B.26 for λ_i^P. □

Proposition C.27 *From lemma C.25, we know that $\mathcal{S}_{\natural\kappa \to \natural\kappa \to \kappa \to \kappa \to \kappa}[\overline{\mathcal{C}}/\overline{\chi}]$ is a candidate of kind $(\natural\kappa \to \natural\kappa \to \kappa \to \kappa \to \kappa)\{\overline{\kappa}/\overline{\chi}\}$, that $\mathcal{S}_{\forall\chi.\,(\chi \to \natural\kappa) \to (\chi \to \kappa) \to \kappa}[\overline{\mathcal{C}}/\overline{\chi}]$ is a candidate of kind $(\forall\chi.\,(\chi \to \natural\kappa) \to (\chi \to \kappa) \to \kappa)\{\overline{\kappa}/\overline{\chi}\}$, that $\mathcal{S}_{(\forall\chi.\,\natural\kappa) \to (\forall\chi.\,\kappa) \to \kappa}[\overline{\mathcal{C}}/\overline{\chi}]$ is a candidate of kind $((\forall\chi.\,\natural\kappa) \to (\forall\chi.\,\kappa) \to \kappa)\{\overline{\kappa}/\overline{\chi}\}$, and $\mathcal{S}_{(\natural\kappa \to \natural\kappa) \to (\kappa \to \kappa) \to \kappa}[\overline{\mathcal{C}}/\overline{\chi}]$ is a candidate of kind $((\natural\kappa \to \natural\kappa) \to (\kappa \to \kappa) \to \kappa)\{\overline{\kappa}/\overline{\chi}\}$. In the rest of the section, we will refer to the above candidates as $\mathcal{S}_{\rightarrow}[\overline{\mathcal{C}}/\overline{\chi}]$, $\mathcal{S}_{\forall}[\overline{\mathcal{C}}/\overline{\chi}]$, $\mathcal{S}_{\forall^+}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\mathcal{S}_\mu[\overline{\mathcal{C}}/\overline{\chi}]$ respectively.*

Lemma C.28 $\mathsf{int} \in \mathcal{S}_{\forall\chi.\,\natural\chi}[\overline{\mathcal{C}}/\overline{\chi}]$

Proof This is true if for all kinds $\kappa\{\overline{\kappa}/\overline{\chi}\}$ and the corresponding candidate $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, $\mathsf{int}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]$ belongs to $\mathcal{S}_{\natural\chi}[\overline{\mathcal{C}}, \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi]$. This is true if $\mathsf{int}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]$ belongs to $\natural_R \mathcal{S}_\chi[\overline{\mathcal{C}}, \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi]$. This implies that $\mathsf{int}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]$ belongs to $\natural_R \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. This is true if $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\, \mathsf{int}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]$ of $(\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+};\, \tau_\mu)$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$; given that $\tau_{\mathsf{int}} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\rightarrow} \in \mathcal{S}_{\rightarrow}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall} \in \mathcal{S}_{\forall}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall^+} \in \mathcal{S}_{\forall^+}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\mu \in \mathcal{S}_\mu[\overline{\mathcal{C}}/\overline{\chi}]$.

Since τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, $\tau_{\forall^+}$, and τ_μ are strongly normalisable, we will induct over $len = \nu(\tau_{\mathsf{int}}) + \nu(\tau_{\rightarrow}) + \nu(\tau_{\forall}) + \nu(\tau_{\forall^+}) + \nu(\tau_\mu)$. We will prove that for all values of len, the type $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\, \mathsf{int}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]$ of $(\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+};\, \tau_\mu)$ always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. The conditions for the hypothesis are that $\tau_{\mathsf{int}} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\rightarrow} \in \mathcal{S}_{\rightarrow}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall} \in \mathcal{S}_{\forall}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall^+} \in \mathcal{S}_{\forall^+}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\mu \in \mathcal{S}_\mu[\overline{\mathcal{C}}/\overline{\chi}]$. Consider the neutral type

$$\tau = \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\, \mathsf{int}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}] \text{ of } (\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+};\, \tau_\mu)$$

- $len = 0$ The only reduction of τ is to τ_{int} which by assumption belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.

- $len = k + 1$ Assume that the inductive hypothesis is true for $len = k$. That is, for $len = k$, the type $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\, \mathsf{int}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]$ of $(\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+};\, \tau_\mu)$ always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$; given that $\tau_{\mathsf{int}} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\rightarrow} \in \mathcal{S}_{\rightarrow}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall} \in \mathcal{S}_{\forall}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall^+} \in \mathcal{S}_{\forall^+}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\mu \in \mathcal{S}_\mu[\overline{\mathcal{C}}/\overline{\chi}]$. By property 3 of definition B.18, for $len = k$, the type $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\, \mathsf{int}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]$ of $(\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+};\, \tau_\mu)$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. Consider the case for $len = k + 1$. Apart from the t_1 reduction, the other possible reductions are to $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\, \mathsf{int}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]$ of $(\tau'_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+};\, \tau_\mu)$ where $\tau_{\mathsf{int}} \rightsquigarrow \tau'_{\mathsf{int}}$, or to $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\, \mathsf{int}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]$ of $(\tau_{\mathsf{int}};\, \tau'_{\rightarrow};\, \tau_{\forall};\, \tau_{\forall^+};\, \tau_\mu)$ where $\tau_{\rightarrow} \rightsquigarrow \tau'_{\rightarrow}$, or to $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\, \mathsf{int}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]$ of $(\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau'_{\forall};\, \tau_{\forall^+};\, \tau_\mu)$ where $\tau_{\forall} \rightsquigarrow \tau'_{\forall}$, or to $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\, \mathsf{int}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]$ of $(\tau_{\mathsf{int}};\, \tau_{\rightarrow};\, \tau_{\forall};\, \tau'_{\forall^+};\, \tau_\mu)$

where $\tau_{\forall^+} \leadsto \tau'_{\forall^+}$, or to $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \mathsf{int}\ [\kappa\{\overline{\kappa}/\overline{\chi}\}]$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau'_{\mu})$ where $\tau_\mu \leadsto \tau'_\mu$. By property 2 of definition C.18, each of τ'_{int}, $\tau'_{\rightarrow}$, $\tau'_{\forall}$, $\tau'_{\forall^+}$, and τ'_μ belong to the same candidate as before. Moreover, $len = k$ for each of the reducts. Therefore, from the inductive hypothesis, each of the reducts belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.

Therefore, the neutral type τ always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. By property 3 of definition C.18, $\tau \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. $\square$

Lemma C.29 $\overset{\circ}{\twoheadrightarrow} \in \mathcal{S}_{\forall\chi.\,\natural\chi\rightarrow\natural\chi\rightarrow\natural\chi}[\overline{\mathcal{C}}/\overline{\chi}]$

Proof This is true if for all kinds $\kappa\{\overline{\kappa}/\overline{\chi}\}$ and the corresponding candidate $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, we have that $\overset{\circ}{\twoheadrightarrow}[\kappa\{\overline{\kappa}/\overline{\chi}\}]$ belongs to $\mathcal{S}_{\natural\chi\rightarrow\natural\chi\rightarrow\natural\chi}[\overline{\mathcal{C}}, \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi]$. This is true if given $\tau_1 \in \mathcal{S}_{\natural\chi}[\overline{\mathcal{C}}, \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi]$ and given $\tau_2 \in \mathcal{S}_{\natural\chi}[\overline{\mathcal{C}}, \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi]$, we have that $\overset{\circ}{\twoheadrightarrow}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau_1\,\tau_2$ belongs to $\mathcal{S}_{\natural\chi}[\overline{\mathcal{C}}, \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi]$. This is true if $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\overset{\circ}{\twoheadrightarrow}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau_1\,\tau_2)$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu)$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$; given that $\tau_{\mathsf{int}} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\rightarrow} \in \mathcal{S}_{\rightarrow}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall} \in \mathcal{S}_{\forall}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall^+} \in \mathcal{S}_{\forall^+}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\mu \in \mathcal{S}_\mu[\overline{\mathcal{C}}/\overline{\chi}]$. Since the types τ_1, τ_2, τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, $\tau_{\forall^+}$, and τ_μ are strongly normalisable, we will induct over $len = \nu(\tau_1) + \nu(\tau_2) + \nu(\tau_{\mathsf{int}}) + \nu(\tau_{\rightarrow}) + \nu(\tau_{\forall}) + \nu(\tau_{\forall^+}) + \nu(\tau_\mu)$. We will prove that for all values of len, the type $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\overset{\circ}{\twoheadrightarrow}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau_1\,\tau_2)$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu)$ always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. The conditions for the hypothesis are that $\tau_1 \in \natural_R\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_2 \in \natural_R\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\mathsf{int}} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\rightarrow} \in \mathcal{S}_{\rightarrow}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall} \in \mathcal{S}_{\forall}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall^+} \in \mathcal{S}_{\forall^+}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\mu \in \mathcal{S}_\mu[\overline{\mathcal{C}}/\overline{\chi}]$. Consider the neutral type

$$\tau = \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\overset{\circ}{\twoheadrightarrow}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau_1\,\tau_2)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu)$$

- $len = 0$ Then the only possible reduction is $\tau' = \tau_{\rightarrow}\,\tau_1\,\tau_2$
  $(\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \tau_1\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu))$
  $(\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \tau_2\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu))$
  By the assumption on τ_1 and τ_2, both $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \tau_1$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu)$ and $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ \tau_2$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu)$ belong to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. We also know that $\mathcal{S}_{\rightarrow}[\overline{\mathcal{C}}/\overline{\chi}] = \mathcal{S}_{\natural\kappa\rightarrow\natural\kappa\rightarrow\kappa\rightarrow\kappa\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$. Therefore, we get that τ' belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.

- $len = k + 1$ The other possible reductions come from the reduction of one of the individual types τ_1, τ_2, τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, $\tau_{\forall^+}$, and τ_μ. The proof in this case is similar to the proof of the corresponding case for lemma C.28.

Therefore, the neutral type τ always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. By property 3 of definition C.18, $\tau \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. $\square$

Lemma C.30 *If for all* $\tau_1 \in \mathcal{S}_{\kappa_1}[\overline{\mathcal{C}}/\overline{\chi}]$, $\tau\{\tau_1/\alpha\} \in \mathcal{S}_{\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$, *then* $\lambda\alpha\!:\!\kappa_1\{\overline{\kappa}/\overline{\chi}\}.\,\tau \in \mathcal{S}_{\kappa_1\rightarrow\kappa_2}[\overline{\mathcal{C}}/\overline{\chi}]$.

Proof Same as lemma B.30 for λ_i^P. $\square$

Lemma C.31 $\mathring{\forall} \in \mathcal{S}_{\forall\chi.\,\forall\chi'.\,(\chi'\rightarrow\natural\chi)\rightarrow\natural\chi}[\overline{\mathcal{C}}/\overline{\chi}]$.

Proof This is true if for all kinds $\kappa\{\overline{\kappa}/\overline{\chi}\}$ and $\kappa_1\{\overline{\kappa}/\overline{\chi}\}$ and the corresponding candidates $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$ and $\mathcal{S}_{\kappa_1}[\overline{\mathcal{C}}/\overline{\chi}]$, and a type τ belonging to $\mathcal{S}_{\chi'\rightarrow\natural\chi}[\overline{\mathcal{C}}, \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}], \mathcal{S}_{\kappa_1}[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi, \chi']$, we have that $\mathring{\forall}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,[\kappa_1\{\overline{\kappa}/\overline{\chi}\}]\,\tau$ belongs to $\mathcal{S}_{\natural\chi}[\overline{\mathcal{C}}, \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}], \mathcal{S}_{\kappa_1}[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi, \chi']$. This implies that $\mathring{\forall}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,[\kappa_1\{\overline{\kappa}/\overline{\chi}\}]\,\tau$ must belong to $\natural_R\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. This is true if

$$\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\mathring{\forall}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,[\kappa_1\{\overline{\kappa}/\overline{\chi}\}]\,\tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu)$$

belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$; given that $\tau_{\mathsf{int}} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\rightarrow} \in \mathcal{S}_{\rightarrow}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall} \in \mathcal{S}_{\forall}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall^+} \in \mathcal{S}_{\forall^+}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\mu \in \mathcal{S}_\mu[\overline{\mathcal{C}}/\overline{\chi}]$. Since the types τ, τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, $\tau_{\forall^+}$, and τ_μ are strongly normalisable, we will induct over $len = \nu(\tau) + \nu(\tau_{\mathsf{int}}) + \nu(\tau_{\rightarrow}) + \nu(\tau_{\forall}) + \nu(\tau_{\forall^+}) + \nu(\tau_\mu)$. We will prove that for all values of len, the type

$$\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\mathring{\forall}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,[\kappa_1\{\overline{\kappa}/\overline{\chi}\}]\,\tau)\ \mathsf{of}\ (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu)$$

always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. The conditions for the hypothesis are that $\tau \in \mathcal{S}_{\kappa_1}[\overline{\mathcal{C}}/\overline{\chi}] \rightarrow \natural_R\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\mathsf{int}} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\rightarrow} \in \mathcal{S}_{\rightarrow}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall} \in \mathcal{S}_{\forall}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall^+} \in \mathcal{S}_{\forall^+}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\mu \in \mathcal{S}_\mu[\overline{\mathcal{C}}/\overline{\chi}]$. Consider the neutral type $\tau' = \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\mathring{\forall}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,[\kappa_1\{\overline{\kappa}/\overline{\chi}\}]\,\tau)$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu)$

- $len = 0$ The only possible reduction of τ' is to

  $$\begin{aligned}\tau'_1 = {} & \tau_{\forall}\,[\kappa_1\{\overline{\kappa}/\overline{\chi}\}]\,\tau \\ & (\lambda\alpha\!:\!\kappa_1\{\overline{\kappa}/\overline{\chi}\}.\ \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\tau\,\alpha)\ \mathsf{of} \\ & \qquad (\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu))\end{aligned}$$

  Consider $\tau'' = \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\tau\,\alpha)$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu)$. For all $\tau_1 \in \mathcal{S}_{\kappa_1}[\overline{\mathcal{C}}/\overline{\chi}]$, we get that $\tau''\{\tau_1/\alpha\} = \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\tau\,\tau_1)$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu)$. By definition, $\tau\,\tau_1$ belongs to $\mathcal{S}_{\natural\chi}[\overline{\mathcal{C}}, \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}], \mathcal{S}_{\kappa_1}[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi, \chi']$ which is equivalent to $\natural_R\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. By definition then, $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\tau\,\tau_1)$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu)$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. By lemma C.30, $\lambda\alpha : \kappa_1\{\overline{\kappa}/\overline{\chi}\}.\,\tau''$ belongs to $\mathcal{S}_{\kappa_1\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$. We also know that $\mathcal{S}_{\forall}[\overline{\mathcal{C}}/\overline{\chi}] = \mathcal{S}_{\forall\chi.\,(\chi\rightarrow\natural\kappa)\rightarrow(\chi\rightarrow\kappa)\rightarrow\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$. Therefore, we get that τ'_1 belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.

- $len = k + 1$ The other possible reductions come from the reduction of one of the individual types τ, τ_{int}, $\tau_{\rightarrow}$, $\tau_{\forall}$, $\tau_{\forall^+}$, and τ_μ. The proof in this case is similar to the proof of the corresponding case for lemma C.28.

Therefore, the neutral type τ' always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. By property 3 of definition C.18, $\tau' \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. $\square$

Lemma C.32 $\mathsf{Place} \in \mathcal{S}_{\forall\chi.\,\chi\rightarrow\natural\chi}[\overline{\mathcal{C}}/\overline{\chi}]$

Proof This is true if for all kinds $\kappa\{\overline{\kappa}/\overline{\chi}\}$ and the corresponding candidate $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and a type τ belonging to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, we have that $\mathsf{Place}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau$ belongs to $\mathcal{S}_{\natural\chi}[\overline{\mathcal{C}}, \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi]$. This implies that $\mathsf{Place}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau$ belongs to $\natural_R\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. This is true if $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\mathsf{Place}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau)$ of $(\tau_{\mathsf{int}};\ \tau_{\rightarrow};\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu)$

belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$; given that $\tau_{\mathsf{int}} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\to \in \mathcal{S}_\to[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall} \in \mathcal{S}_{\forall}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall^+} \in \mathcal{S}_{\forall^+}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\mu \in \mathcal{S}_\mu[\overline{\mathcal{C}}/\overline{\chi}]$. Since the types τ, τ_{int}, $\tau_\to$, $\tau_{\forall}$, $\tau_{\forall^+}$, and τ_μ are strongly normalisable, we will induct over $len = \nu(\tau) + \nu(\tau_{\mathsf{int}}) + \nu(\tau_\to) + \nu(\tau_{\forall}) + \nu(\tau_{\forall^+}) + \nu(\tau_\mu)$. We will prove that for all values of len, the type $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\mathsf{Place}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau)$ of $(\tau_{\mathsf{int}};\ \tau_\to;\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu)$ always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. The conditions for the hypothesis are that $\tau \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\mathsf{int}} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\to \in \mathcal{S}_\to[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall} \in \mathcal{S}_{\forall}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall^+} \in \mathcal{S}_{\forall^+}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\mu \in \mathcal{S}_\mu[\overline{\mathcal{C}}/\overline{\chi}]$. Consider the neutral type

$$\tau' = \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\mathsf{Place}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau) \text{ of } (\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\forall^+};\ \tau_\mu)$$

- $len = 0$ The only possible reduction of τ' is to τ. By assumption, this belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.
- $len = k + 1$ The other possible reductions come from the reduction of one of the individual types τ, τ_{int}, $\tau_\to$, $\tau_{\forall}$, $\tau_{\forall^+}$, and τ_μ. The proof in this case is similar to the proof of the corresponding case for lemma C.28.

Therefore, the neutral type τ' always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. By property 3 of definition C.18, $\tau' \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. □

Lemma C.33 $\mathring{\mu} \in \mathcal{S}_{\forall\chi.\,(\natural\chi\to\natural\chi)\to\natural\chi}[\overline{\mathcal{C}}/\overline{\chi}]$

Proof This is true if for all kinds $\kappa\{\overline{\kappa}/\overline{\chi}\}$ and the corresponding candidate $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and a type τ belonging to $\mathcal{S}_{\natural\chi\to\natural\chi}[\overline{\mathcal{C}}, \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi]$, we have that $\mathring{\mu}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau$ belongs to $\mathcal{S}_{\natural\chi}[\overline{\mathcal{C}}, \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi]$. This implies that $\mathring{\mu}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau$ belongs to $\natural_R\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. This is true if $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\mathring{\mu}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau)$ of $(\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\forall^+};\ \tau_\mu)$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$; given that $\tau_{\mathsf{int}} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\to \in \mathcal{S}_\to[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall} \in \mathcal{S}_{\forall}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall^+} \in \mathcal{S}_{\forall^+}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\mu \in \mathcal{S}_\mu[\overline{\mathcal{C}}/\overline{\chi}]$. Since the types τ, τ_{int}, $\tau_\to$, $\tau_{\forall}$, $\tau_{\forall^+}$, and τ_μ are strongly normalisable, we will induct over $len = \nu(\tau) + \nu(\tau_{\mathsf{int}}) + \nu(\tau_\to) + \nu(\tau_{\forall}) + \nu(\tau_{\forall^+}) + \nu(\tau_\mu)$. We will prove that for all values of len, the type $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\mathring{\mu}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau)$ of $(\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\forall^+};\ \tau_\mu)$ always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. The conditions for the hypothesis are that $\tau \in \mathcal{S}_{\natural\chi\to\natural\chi}[\overline{\mathcal{C}}, \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi]$, and $\tau_{\mathsf{int}} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\to \in \mathcal{S}_\to[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall} \in \mathcal{S}_{\forall}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall^+} \in \mathcal{S}_{\forall^+}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\mu \in \mathcal{S}_\mu[\overline{\mathcal{C}}/\overline{\chi}]$. Consider the neutral type $\tau' = \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\mathring{\mu}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau)$ of $(\tau_{\mathsf{int}};\ \tau_\to;\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu)$

- $len = 0$ The only possible reduction is to
  $$\tau_1' = \tau_\mu\,\tau\,(\lambda\alpha\!:\!\kappa\{\overline{\kappa}/\overline{\chi}\}.\ \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\tau\,(\mathsf{Place}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\alpha)) \text{ of } (\tau_{\mathsf{int}};\ \tau_\to;\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu))$$
  Consider
  $$\tau'' = \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\tau\,(\mathsf{Place}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\alpha)) \text{ of } (\tau_{\mathsf{int}};\ \tau_\to;\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu)$$
  For any type τ_1 belonging to the candidate $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, we get that
  $$\tau''\{\tau_1/\alpha\} = \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\tau\,(\mathsf{Place}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau_1)) \text{ of } (\tau_{\mathsf{int}};\ \tau_\to;\ \tau_{\forall};\ \tau_{\forall^+};\ \tau_\mu)$$
  By lemma C.32, $\mathsf{Place} \in \mathcal{S}_{\forall\chi.\,\chi\to\natural\chi}[\overline{\mathcal{C}}/\overline{\chi}]$. Therefore, $\mathsf{Place}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau_1$ belongs to $\natural_R\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. Therefore, $\tau\,(\mathsf{Place}\,[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau_1)$ also belongs to $\natural_R\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. Therefore, by definition, $\tau''\{\tau_1/\alpha\}$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. By lemma C.30, $\lambda\alpha\!:\!\kappa\{\overline{\kappa}/\overline{\chi}\}.\,\tau''$ belongs to $\mathcal{S}_{\kappa\to\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$. We also know that $\mathcal{S}_\mu[\overline{\mathcal{C}}/\overline{\chi}] = \mathcal{S}_{(\natural\kappa\to\natural\kappa)\to(\kappa\to\kappa)\to\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$. This implies that τ_1' belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.
- $len = k + 1$ The other possible reductions come from the reduction of one of the individual types τ, τ_{int}, $\tau_\to$, $\tau_{\forall}$, $\tau_{\forall^+}$, and τ_μ. The proof in this case is similar to the proof of the corresponding case for lemma C.28.

Therefore, the neutral type τ' always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. By property 3 of definition C.18, $\tau' \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. □

Lemma C.34 *If for every kind κ' and reducibility candidate $\mathcal{C}'$ of this kind, $\tau\{\kappa'/\chi'\} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$, then $\Lambda\chi'.\,\tau \in \mathcal{S}_{\forall\chi'.\,\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$.*

Proof Same as lemma B.32 for λ_i^P. □

Lemma C.35 *If $\tau \in \mathcal{S}_{\forall\chi.\,\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$, then $\tau\,[\kappa'\{\overline{\kappa}/\overline{\chi}\}] \in \mathcal{S}_{\kappa\{\kappa'/\chi\}}[\overline{\mathcal{C}}/\overline{\chi}]$ for every kind $\kappa'\{\overline{\kappa}/\overline{\chi}\}$.*

Proof By definition, $\tau\,[\kappa'\{\overline{\kappa}/\overline{\chi}\}]$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi]$, for every kind $\kappa'\{\overline{\kappa}/\overline{\chi}\}$ and reducibility candidate $\mathcal{C}'$ of this kind. Set $\mathcal{C}' = \mathcal{S}_{\kappa'}[\overline{\mathcal{C}}/\overline{\chi}]$. Applying lemma C.26 leads to the result. □

Lemma C.36 $\mathring{\forall}^{+} \in \mathcal{S}_{\forall\chi.\,(\forall\chi_1.\,\natural\chi)\to\natural\chi}[\overline{\mathcal{C}}/\overline{\chi}]$.

Proof This is true if for all kinds $\kappa\{\overline{\kappa}/\overline{\chi}\}$, and the corresponding candidate $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and a type τ belonging to $\mathcal{S}_{\forall\chi_1.\,\natural\chi}[\overline{\mathcal{C}}, \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi]$, we have that $\mathring{\forall}^{+}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau$ belongs to $\mathcal{S}_{\natural\chi}[\overline{\mathcal{C}}, \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi]$. This implies that $\mathring{\forall}^{+}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau$ belongs to $\natural_R\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. This is true if $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\mathring{\forall}^{+}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau)$ of $(\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\forall^+};\ \tau_\mu)$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$; given that $\tau_{\mathsf{int}} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\to \in \mathcal{S}_\to[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall} \in \mathcal{S}_{\forall}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall^+} \in \mathcal{S}_{\forall^+}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\mu \in \mathcal{S}_\mu[\overline{\mathcal{C}}/\overline{\chi}]$. Since the types τ, τ_{int}, $\tau_\to$, $\tau_{\forall}$, $\tau_{\forall^+}$, and τ_μ are strongly normalisable, we will induct over $len = \nu(\tau) + \nu(\tau_{\mathsf{int}}) + \nu(\tau_\to) + \nu(\tau_{\forall}) + \nu(\tau_{\forall^+}) + \nu(\tau_\mu)$. We will prove that for all values of len, the type $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\mathring{\forall}^{+}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau)$ of $(\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\forall^+};\ \tau_\mu)$ always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. The conditions for the hypothesis are that $\tau \in \mathcal{S}_{\forall\chi_1.\,\natural\chi}[\overline{\mathcal{C}}, \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]/\overline{\chi}, \chi]$, and $\tau_{\mathsf{int}} \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\to \in \mathcal{S}_\to[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall} \in \mathcal{S}_{\forall}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall^+} \in \mathcal{S}_{\forall^+}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\mu \in \mathcal{S}_\mu[\overline{\mathcal{C}}/\overline{\chi}]$. Consider the neutral type $\tau' = \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\mathring{\forall}^{+}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\,\tau)$ of $(\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\forall^+};\ \tau_\mu)$

- $len = 0$ The only possible reduction of τ' is to $\tau_{\forall^+}\,\tau\,(\Lambda\chi.\,\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\tau\,[\chi])$ of $(\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\forall^+};\ \tau_\mu))$
  Consider
  $$\tau'' = \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\tau\,[\chi]) \text{ of } (\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\forall^+};\ \tau_\mu)$$
  For an arbitrary kind κ' and corresponding candidate $\mathcal{C}'$, we get that
  $$\tau''\{\kappa'/\chi\} = \mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\tau\,[\kappa']) \text{ of } (\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\forall^+};\ \tau_\mu)$$
  By the assumption on τ, we get that $\tau\,[\kappa']$ belongs to $\natural_R\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. By definition, $\mathsf{Typerec}[\kappa\{\overline{\kappa}/\overline{\chi}\}]\ (\tau\,[\kappa'])$ of $(\tau_{\mathsf{int}};\ \tau_\to;\ \tau_\forall;\ \tau_{\forall^+};\ \tau_\mu)$

belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. Since χ does not occur free in κ, we may also write that $\tau''\{\kappa'/\chi\}$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi]$. By lemma C.34, this implies that $\Lambda\chi.\,\tau''$ belongs to $\mathcal{S}_{\forall\chi.\,\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$. We also know that $\tau_{\forall^+} \in \mathcal{S}_{(\forall\chi.\,\natural\kappa)\to(\forall\chi.\,\kappa)\to\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$. Also, χ does not occur free in κ. Therefore, we get that $\tau_{\forall^+}\,\tau\,(\Lambda\chi.\,\tau'')$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.

- $len = k+1$ The other possible reductions come from the reduction of one of the individual types τ, τ_{int}, $\tau_\to$, $\tau_\forall$, $\tau_{\forall^+}$, and τ_μ. The proof in this case is similar to the proof of the corresponding case for lemma C.28.

Therefore, the neutral type τ' always reduces to a type that belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. By property 3 of definition C.18, $\tau' \in \mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$. $\square$

We now come to the main result of this section.

Theorem C.37 (Candidacy) *Let τ be a type of kind κ. Suppose all the free type variables of τ are in $\alpha_1 \ldots \alpha_n$ of kinds $\kappa_1 \ldots \kappa_n$ and all the free kind variables of κ, $\kappa_1 \ldots \kappa_n$ are among $\chi_1 \ldots \chi_m$. If $\mathcal{C}_1 \ldots \mathcal{C}_m$ are candidates of kinds $\kappa'_1 \ldots \kappa'_m$ and $\tau_1 \ldots \tau_n$ are types of kind $\kappa_1\{\overline{\kappa'}/\overline{\chi}\} \ldots \kappa_n\{\overline{\kappa'}/\overline{\chi}\}$ which are in $\mathcal{S}_{\kappa_1}[\overline{\mathcal{C}}/\overline{\chi}] \ldots \mathcal{S}_{\kappa_n}[\overline{\mathcal{C}}/\overline{\chi}]$, then $\tau\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.*

Proof The proof is by induction over the structure of τ.

The cases of int, $\mathring{\twoheadrightarrow}$, $\mathring{\forall}$, $\mathring{\forall}^+$, $\mathring{\mu}$, and Place are covered by lemmas C.28 C.29 C.31 C.36 C.33 C.32.

Suppose $\tau = \alpha_i$ and $\kappa = \kappa_i$. Then $\tau\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\} = \tau_i$. By assumption, this belongs to $\mathcal{S}_{\kappa_i}[\overline{\mathcal{C}}/\overline{\chi}]$.

Suppose $\tau = \tau'_1\,\tau'_2$. Then $\tau'_1 : \kappa' \to \kappa$ for some kind κ' and $\tau'_2 : \kappa'$. By the inductive hypothesis, $\tau'_1\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_{\kappa'\to\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$ and $\tau'_2\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_{\kappa'}[\overline{\mathcal{C}}/\overline{\chi}]$. Therefore, $(\tau'_1\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\})\,(\tau'_2\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\})$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.

Suppose $\tau = \tau'\,[\kappa']$. Then $\tau' : \forall\chi_1.\,\kappa_1$ and $\kappa = \kappa_1\{\kappa'/\chi_1\}$. By the inductive hypothesis, $\tau'\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_{\forall\chi_1.\,\kappa_1}[\overline{\mathcal{C}}/\overline{\chi}]$. By lemma C.35 $\tau'\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}\,[\kappa'\{\overline{\kappa'}/\overline{\chi}\}]$ belongs to $\mathcal{S}_{\kappa_1\{\kappa'/\chi_1\}}[\overline{\mathcal{C}}/\overline{\chi}]$ which is equivalent to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.

Suppose $\tau = \mathsf{Typerec}[\kappa]\;\tau'\;\mathsf{of}\;(\tau_{\mathsf{int}};\,\tau_\to;\,\tau_\forall;\,\tau_{\forall^+};\,\tau_\mu)$. Then $\tau' : \natural\kappa$, and $\tau_{\mathsf{int}} : \kappa$, and $\tau_\to : \natural\kappa \to \natural\kappa \to \kappa \to \kappa \to \kappa$, and $\tau_\forall : \forall\chi.\,(\chi \to \natural\kappa) \to (\chi \to \kappa) \to \kappa$, and $\tau_{\forall^+} : (\forall\chi.\,\natural\kappa) \to (\forall\chi.\,\kappa) \to \kappa$ and $\tau_\mu : (\natural\kappa \to \natural\kappa) \to (\kappa \to \kappa) \to \kappa$. By the inductive hypothesis $\tau'\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_{\natural\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\mathsf{int}}\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\to\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_{\natural\kappa\to\natural\kappa\to\kappa\to\kappa\to\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\forall\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_{\forall\chi.\,(\chi\to\natural\kappa)\to(\chi\to\kappa)\to\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_{\forall^+}\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_{(\forall\chi.\,\natural\kappa)\to(\forall\chi.\,\kappa)\to\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$, and $\tau_\mu\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_{(\natural\kappa\to\natural\kappa)\to(\kappa\to\kappa)\to\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$. By definition of $\mathcal{S}_{\natural\kappa}[\overline{\mathcal{C}}/\overline{\chi}]$,

$$\begin{array}{l}\mathsf{Typerec}[\kappa\{\overline{\kappa'}/\overline{\chi}\}]\;\tau'\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\}\;\mathsf{of}\\ \quad(\tau_{\mathsf{int}}\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\};\tau_\to\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\};\\ \quad\;\tau_\forall\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\};\tau_{\forall^+}\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\};\\ \quad\;\tau_\mu\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\})\end{array}$$

belongs to $\mathcal{S}_\kappa[\overline{\mathcal{C}}/\overline{\chi}]$.

Suppose $\tau = \lambda\alpha' : \kappa'.\,\tau_1$. Then $\tau_1 : \kappa''$ where the free type variables of τ_1 are in $\alpha_1, \ldots, \alpha_n, \alpha'$ and $\kappa = \kappa' \to \kappa''$. By the inductive hypothesis, $\tau_1\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}, \tau'/\overline{\alpha}, \alpha'\}$ belongs to $\mathcal{S}_{\kappa''}[\overline{\mathcal{C}}/\overline{\chi}]$ where τ' is of kind $\kappa'\{\overline{\kappa'}/\overline{\chi}\}$ and belongs to $\mathcal{S}_{\kappa'}[\overline{\mathcal{C}}/\overline{\chi}]$. This implies that $(\tau_1\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\})\{\tau'/\alpha'\}$ (since α' occurs free only in τ_1) belongs to $\mathcal{S}_{\kappa''}[\overline{\mathcal{C}}/\overline{\chi}]$. By lemma C.30, $\lambda\alpha' : \kappa'\{\overline{\kappa'}/\overline{\chi}\}.\,(\tau_1\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\})$ belongs to $\mathcal{S}_{\kappa'\to\kappa''}[\overline{\mathcal{C}}/\overline{\chi}]$.

Suppose $\tau = \Lambda\chi'.\,\tau'$. Then $\tau' : \kappa''$ and $\kappa = \forall\chi'.\,\kappa''$. By the inductive hypothesis, $\tau'\{\overline{\kappa'}, \kappa'/\overline{\chi}, \chi'\}\{\overline{\tau}/\overline{\alpha}\}$ belongs to $\mathcal{S}_{\kappa''}[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$ for an arbitrary kind κ' and candidate $\mathcal{C}'$ of kind κ'. Since χ' occurs free only in τ', we get that $(\tau'\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\})\{\kappa'/\chi'\}$ belongs to $\mathcal{S}_{\kappa''}[\overline{\mathcal{C}}, \mathcal{C}'/\overline{\chi}, \chi']$. By lemma C.34, $\Lambda\chi'.\,(\tau'\{\overline{\kappa'}/\overline{\chi}\}\{\overline{\tau}/\overline{\alpha}\})$ belongs to $\mathcal{S}_{\forall\chi'.\,\kappa''}[\overline{\mathcal{C}}/\overline{\chi}]$. $\square$

Suppose SN_i is the set of strongly normalisable types of kind κ_i.

Corollary C.38 *All types are strongly normalisable.*

Proof Follows from theorem C.37 by putting $\mathcal{C}_i = SN_i$ and $\tau_i = \alpha_i$. $\square$

C.3 Confluence

Confluence for the λ_i^Q type reduction relation is proved in the same way as the λ_i^P type reduction confluence. The additional cases follow in a straightforward manner.