Index-Stratified Types

Rohan Jacob-Rao
Digital Asset, Sydney, Australia
rohanjr@digitalasset.com

Brigitte Pientka
McGill University, Montreal, Canada
bpientka@cs.mcgill.ca

David Thibodeau
McGill University, Montreal, Canada
david.thibodeau@mail.mcgill.ca

Abstract

We present Tores, a core language for encoding metatheoretic proofs. The novel features we introduce are well-founded Mendler-style (co)recursion over indexed data types and a form of recursion over objects in the index language to build new types. The latter, which we call *index-stratified types*, are analogue to the concept of large elimination in dependently typed languages. These features combined allow us to encode sophisticated case studies such as normalization for lambda calculi and normalization by evaluation. We prove the soundness of Tores as a programming and proof language via the key theorems of subject reduction and termination.

2012 ACM Subject Classification Theory of computation → Type theory
Theory of computation → Logic and verification

Keywords and phrases Indexed types, (co)recursive types, logical relations

Digital Object Identifier 10.4230/LIPIcs.FSCD.2018.19

Funding This research was funded by the Natural Science and Engineering Research Council (NSERC) Canada.

Acknowledgements We thank Andrew Cave for the idea of stratified types and for guiding the initial development.

1 Introduction

Recursion is a fundamental tool for writing useful programs in functional languages. When viewed from a logical perspective via the Curry-Howard correspondence, well-founded recursion corresponds to inductive reasoning. Dually, well-founded corecursion corresponds to coinductive reasoning. However, concentrating only on well-founded (co)recursive definitions is not sufficient to support the encoding of meta-theoretic proofs. There are two missing ingredients: 1) To express fine-grained properties we often rely on first-order logic which is analogous to *indexed types* in programming languages. 2) Many common notions cannot be directly characterized by well-founded (co)recursive definitions. An example is Girard's notion of reducibility for functions: a term M is reducible at type $A \to B$ if, for all terms N that are reducible at type A, we have that $M\ N$ is reducible at type B. This definition is well-founded because it is by structural recursion on the type indices (A and B), so we want to admit such definitions.

Our contribution in this paper is a core language called Tores that features indexed types and (co)inductive reasoning via well-founded (co)recursion. The primary forms of types are

© Rohan Jacob-Rao and Brigitte Pientka and David Thibodeau;
licensed under Creative Commons License CC-BY

indexed (co)recursive types, over which we support reasoning via Mendler-style (co)recursion. Additionally, TORES features *index-stratified types*, which allow further definitions of types via well-founded recursion over indices. The main difference between the two forms is that (co)recursive types are more flexible, allowing (co)induction, while stratified types only support unfolding based on their indices. The combination of the two features is especially powerful for formalizing metatheory involving logical relations. This is partly because type definitions in TORES do not require positivity, a condition used in other systems to ensure termination and in turn logical consistency. Despite this, we are able to prove termination of TORES programs using a semantic interpretation of types.

How to justify definitions that are recursively defined on a given index in addition to well-founded (co)recursive definitions has been explored in proof theory (see for example Tiu [2012], Baelde and Nadathur [2012]). While this line of work is more general, it is also more complex and further from standard programming practice. In dependent type theories, large eliminations achieve the same. Our approach, grounded in the Curry-Howard isomorphism, provides a complementary perspective on this problem where we balance expressiveness and ease of programming with a compact metatheory. We believe this may be an advantage when considering more sophisticated index languages and reasoning techniques.

The combination of indexed (co)recursive types and stratified types is already used in the programming and proof environment BELUGA, where the index language is an extension of the logical framework LF together with first-class contexts and substitutions [Nanevski et al., 2008, Pientka, 2008, Cave and Pientka, 2012]. This allows elegant implementations of proofs using logical relations [Cave and Pientka, 2013, 2015] and normalization by evaluation [Cave and Pientka, 2012]. TORES can be seen as small kernel into which we elaborate total BELUGA programs, thereby providing post-hoc justification of viewing BELUGA programs as (co)inductive proofs.

2 Index Language for Tores

The design of TORES is parametric over an index language. Following Thibodeau et al. [2016] we stay as abstract as possible and state the general conditions the index language must satisfy. Whenever we require inspection of the particular index language, namely the structure of stratified types and induction terms, we will draw attention to it.

To illustrate the required structure for a concrete index language, we use natural numbers. In practice, however, we can consider other index languages such as those of strings, types [Cheney and Hinze, 2003, Xi et al., 2003], or (contextual) LF [Pientka, 2008, Cave and Pientka, 2012]. It is important to note that, for most of our design, we accommodate a general index language up to the complexity of Contextual LF. Thus we treat index types and TORES kinds as dependently typed, although we use natural numbers in stratified types and induction terms.

The abstract requirements of our index language are listed throughout this section. To summarize them here, they are: decidable type checking, decidable equality, standard substitution principles, decidable unification as well as sound and complete matching. Implicitly, we also require that each index type intended for use in stratified types and induction terms should have a well-founded recursion scheme, i.e. an induction principle. For an index language of Contextual LF, for example, the recursion scheme can be generated using a covering set of index terms for each index type [Pientka and Abel, 2015]. This inductive structure is necessary to show decidability of type checking (Thm 9) and termination (Thm 32) of TORES.

2.1 General Structure

We refer to a term in the index language as an *index term* M, which may have an *index type* U. In the case of natural numbers, there is a single index type nat, and index terms are built from 0, suc, and variables u which must be declared in an *index context* Δ.

$$\begin{array}{llll@{\qquad}lll} \text{Index types} & U & ::= \mathsf{nat} & & \text{Index contexts} & \Delta & ::= \cdot \mid \Delta, u{:}U \\ \text{Index terms} & M & ::= 0 \mid \mathsf{suc}\, M \mid u & & \text{Index substitutions} & \Theta & ::= \cdot \mid \Theta, M/u \end{array}$$

Tores relies on typing for index terms which we give for natural numbers in Fig. 1. The equality judgment for natural numbers is given simply by reflexivity (syntactic equality). We also give typing for index substitutions, which supply an index term for each index variable in the domain Δ and describe well-formed contexts. These definitions are generic.

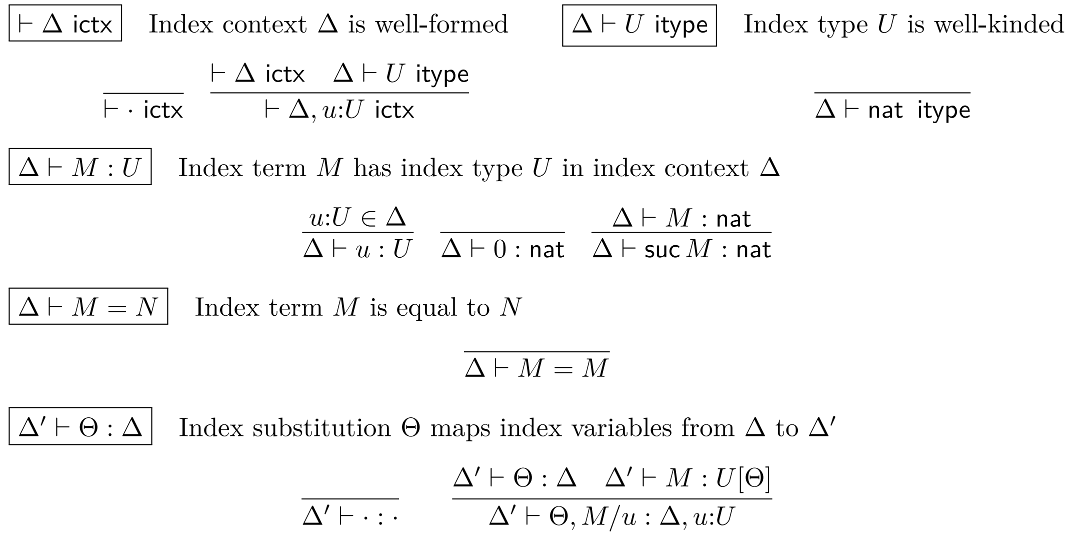

Figure 1 Index language structure

We require that both typing and equality of index terms be decidable in order for type checking of Tores programs to be decidable.

▶ Requirement 1. Index type checking is decidable.

▶ Requirement 2. Index equality is decidable.

We can lift the kinding, typing, equality and matching rules to *spines* of index terms and types generically. We write $\cdot$ and $(\cdot)$ for the empty spines of terms and types respectively. If M_0 is an index term and $\overrightarrow{M}$ is a spine, then $M_0, \overrightarrow{M}$ is a spine. Similarly if $u_0{:}U_0$ is an index type assignment and $(\overrightarrow{u{:}U})$ is a type spine, then $(u_0{:}U_0, \overrightarrow{u{:}U})$ is a type spine. Spines are convenient for setting up the types and terms of Tores. Unlike index substitutions Θ which are built from right to left, spines are built from left to right.

Below we define well-kinded spines of index types and well-typed spines of index terms,

108 which are generic to the particular index language.

$\boxed{\Delta \vdash (\overrightarrow{u{:}U})\ \mathsf{itype}}$ Spine $\overrightarrow{u{:}U}$ of index types is well-kinded

$$\frac{}{\Delta \vdash (\cdot)\ \mathsf{itype}} \qquad \frac{\Delta \vdash U_0\ \mathsf{itype} \quad \Delta, u_0{:}U_0 \vdash (\overrightarrow{u : U})\ \mathsf{itype}}{\Delta \vdash (u_0{:}U_0, \overrightarrow{u : U})\ \mathsf{itype}}$$

109

$\boxed{\Delta \vdash \vec{M} : (\overrightarrow{u{:}U})}$ Spine $\vec{M}$ of index terms have index types $(\overrightarrow{u{:}U})$

$$\frac{}{\Delta \vdash \cdot : (\cdot)} \qquad \frac{\Delta \vdash M_0 : U_0 \quad \Delta \vdash \vec{M} : (\overrightarrow{u{:}U})[M_0/u_0]}{\Delta \vdash M_0, \vec{M} : (u_0{:}U_0, \overrightarrow{u{:}U})}$$

110 ▶ **Lemma 1.** *Type checking of index spines is decidable.*

111 **Proof.** Simply rely on decidable type checking of single index terms (Req. 1). ◀

112 2.2 Substitutions

113 Throughout our development we use both a single index substitution operation $M[N/u]$ and
114 a simultaneous substitution operation $M[\Theta]$. For composition of simultaneous substitutions
115 we write $\Theta_1[\Theta_2]$.

116 ▶ **Definition 2** (Composition of index substitutions)**.** Suppose $\Delta_1 \vdash \Theta_1 : \Delta$ and $\Delta_2 \vdash \Theta_2 : \Delta_1$.
117 Then $\Delta_2 \vdash \Theta_1[\Theta_2] : \Delta$ where

118 $$(\cdot)[\Theta_2] = \Theta_2$$
119 120 $$(\Theta_1', M/u)[\Theta_2] = \Theta_1'[\Theta_2], M[\Theta_2]/u$$

121 We rely on standard properties of single and simultaneous substitutions which we sum-
122 marize below. These say that substitutions preserve typing (3.1 and 3.2) and equality (3.3)
123 and are associative (3.4).

124 125 ▶ Requirement 3 (Index substitution principles).

126 **3.1.** If $\Delta_1, u{:}U', \Delta_2 \vdash M : U$ and $\Delta_1 \vdash N : U'$ then $\Delta_1, \Delta_2[N/u] \vdash M[N/u] : U[N/u]$.
127 **3.2.** If $\Delta' \vdash \Theta : \Delta$ and $\Delta \vdash M : U$ then $\Delta' \vdash M[\Theta] : U[\Theta]$.
128 **3.3.** If $\Delta' \vdash \Theta : \Delta$ and $\Delta \vdash M = N$ then $\Delta' \vdash M[\Theta] = N[\Theta]$.
129 **3.4.** If $\Delta \vdash M : U$ and $\Delta_1 \vdash \Theta_1 : \Delta$ and $\Delta_2 \vdash \Theta_2 : \Delta_1$ then $M[\Theta_1][\Theta_2] = M[\Theta_1[\Theta_2]]$.

130 2.3 Unification and Matching

131 Type checking of Tores relies on a unification procedure to generate a *most general unifier*
132 *(MGU)*. A *unifier* for index terms M and N in a context Δ is a substitution Θ which transforms
133 M and N into syntactically equal terms in another context Δ'. That is, $\Delta' \vdash \Theta : \Delta$ and
134 $\Delta' \vdash M[\Theta] = N[\Theta]$. Θ is "most general" if it does not make more commitments to variables
135 than absolutely necessary. A unifying substitution Θ only makes sense together with its
136 range Δ', so we usually write them as a pair $(\Delta' \mid \Theta)$. In general, there may be more than
137 one MGU for a particular unification problem, or none at all. However, we require here
138 that each problem has at most one MGU up to α-equivalence. We write the generation of
139 an MGU using the judgment $\Delta \vdash M \doteqdot N \searrow P$, where P is either the MGU $(\Delta' \mid \Theta)$ if it
140 exists or # representing that unification failed. To illustrate, we show the unification rules

141 for natural numbers. We write id_i for the identity substitution that maps index variables
142 from Δ_i to themselves.

$$\boxed{\Delta \vdash M \doteqdot N \searrow P} \text{ Index terms } M \text{ and } N \text{ have MGU } P$$

$$\frac{}{\Delta \vdash 0 \doteqdot 0 \searrow (\Delta \mid \mathsf{id})} \quad \frac{\Delta \vdash M \doteqdot N \searrow P}{\Delta \vdash \mathsf{suc}\, M \doteqdot \mathsf{suc}\, N \searrow P} \quad \frac{}{\Delta \vdash u \doteqdot u \searrow (\Delta \mid \mathsf{id})}$$

143
$$\frac{u \notin \mathrm{FV}(M) \quad \Delta = \Delta_1, u{:}U, \Delta_2 \quad \Delta' = \Delta_1, \Delta_2[M/u]}{\Delta \vdash u \doteqdot M \searrow (\Delta' \mid \mathsf{id}_1, M/u, \mathsf{id}_2)} \quad \text{(same for } M \doteqdot u\text{)}$$

$$\frac{}{\Delta \vdash 0 \doteqdot \mathsf{suc}\, M \searrow \#} \quad \frac{}{\Delta \vdash \mathsf{suc}\, M \doteqdot 0 \searrow \#} \quad \frac{u \in \mathrm{FV}(M) \quad M \neq u}{\Delta \vdash u \doteqdot M \searrow \#} \quad \text{(same for } M \doteqdot u\text{)}$$

144 The unification procedure is required for type checking the equality elimination forms
145 $\mathtt{eq}\, s\, \mathtt{with}\, (\Delta'.\Theta \mapsto t)$ and $\mathtt{eq_abort}\, s$ in TORES, which we explain in Section 3.2. In each
146 form, the term s is a witness of an index equality $\Delta \vdash M = N$. In order to use this equality
147 (or determine that it is spurious), we perform unification $\Delta \vdash M \doteqdot N \searrow P$ and check that
148 the result P matches the source term. For the term $\mathtt{eq}\, s\, \mathtt{with}\, (\Delta'.\Theta \mapsto t)$, we check that P
149 is an α-equivalent unifier to the provided one $(\Delta' \mid \Theta)$. For the failure term $\mathtt{eq_abort}\, s$ we
150 check that P is $\#$, yielding a contradiction. Hence our type checking algorithm for TORES
151 relies on a sound and complete unification procedure. We summarize our requirements for
152 unification below.

153 ▶ Requirement 4 (Decidable unification). Given index terms M and N in a context Δ, the
154 judgment $\Delta \vdash M \doteqdot N \searrow P$ is decidable. Either P is $(\Delta' \mid \Theta)$, the unique MGU up to
155 α-equivalence, or P is $\#$ and there is no unifier.

156 Finally, our operational semantics relies on index *matching*. This is an asymmetric form
157 of unification: given terms M in Δ and N in Δ', matching identifies a substitution Θ such
158 that $\Delta' \vdash M[\Theta] = N$. We describe it using the judgment $\Delta \vdash M \doteq N \searrow (\Delta' \mid \Theta)$.
159 Matching is used during evaluation of the equality elimination $\mathtt{eq}\, s\, \mathtt{with}\, (\Delta'.\Theta \mapsto t)$
160 to extend the substitution Θ to a full index environment (grounding substitution) θ. To
161 achieve this, we must lift the notion of matching to the level of index substitutions. This
162 can be done generically given an algorithm for matching index terms. The judgment
163 $\Delta \vdash \Theta_1 \doteq \Theta_2 \searrow (\Delta' \mid \Theta)$ says that matching discovered a substitution Θ such that
164 $\Delta' \vdash \Theta_1[\Theta] = \Theta_2$.
165 To illustrate an algorithm for index matching, we provide the rules for our natural
166 number domain in Fig. 2. We also show the generic lifting of the algorithm to match index
167 substitutions.
168 We then require that index (substitution) matching is both sound and complete. We
169 make these properties precise in our final requirements below. The notion of matching also
170 lifts to the level of index substitutions. We omit the full specifications here and instead state
171 the required properties.

172 ▶ Requirement 5 (Soundness of index matching).

174 **5.1.** If $\Delta \vdash M : U$ and $\Delta \vdash M \doteq N \searrow (\Delta' \mid \Theta)$ then $\Delta' \vdash \Theta : \Delta$ and $\Delta' \vdash M[\Theta] = N$.
175 **5.2.** If $\Delta_1 \vdash \Theta_1 : \Delta$ and $\Delta_1 \vdash \Theta_1 \doteq \Theta_2 \searrow (\Delta_2 \mid \Theta)$ then $\Delta_2 \vdash \Theta : \Delta_1$ and $\Delta_2 \vdash \Theta_1[\Theta] = \Theta_2$.

176 ▶ Requirement 6 (Completeness of index matching). Suppose $\vdash \theta : \Delta$ and $\vdash M[\theta] = N[\theta]$ and
177 $\Delta \vdash M \doteqdot N \searrow (\Delta' \mid \Theta)$. Then $\Delta' \vdash \Theta \doteq \theta \searrow (\cdot \mid \theta')$.

$\boxed{\Delta \vdash M \doteq N \searrow (\Delta' \mid \Theta)}$ Index term M matches index term N under Θ

$$\frac{\Delta = \Delta_1, u : \mathsf{nat}, \Delta_2 \quad \Delta' = \Delta_1, \Delta_2}{\Delta \vdash u \doteq N \searrow (\Delta' \mid \mathsf{id}_{\Delta_1}, N/u, \mathsf{id}_{\Delta_2})}$$

$$\frac{}{\Delta \vdash z \doteq z \searrow (\Delta \mid \mathsf{id}_\Delta)} \qquad \frac{\Delta \vdash M \doteq N \searrow (\Delta' \mid \Theta)}{\Delta \vdash \mathsf{suc}\, M \doteq \mathsf{suc}\, N \searrow (\Delta' \mid \Theta)}$$

$\boxed{\Delta \vdash \Theta_1 \doteq \Theta_2 \searrow (\Delta' \mid \rho)}$ Index substitution Θ_1 matches index substitution Θ_2 under ρ

$$\frac{}{\Delta \vdash \cdot \doteq \cdot \searrow (\Delta \mid \mathsf{id}_\Delta)} \qquad \frac{\Delta \vdash \Theta_1 \doteq \Theta_2 \searrow (\Delta_1 \mid \rho_1) \quad \Delta_1 \vdash M[\rho_1] \doteq N \searrow (\Delta_2 \mid \rho_2)}{\Delta \vdash (\Theta_1, M/u) \doteq (\Theta_2, N/u) \searrow (\Delta_2 \mid \rho_1[\rho_2])}$$

Figure 2 Index matching for natural numbers and generic substitutions

3 Specification of Tores

We now describe TORES, a programming language designed to express (co)inductive proofs and programs using Mendler-style (co)recursion. It also features *index-stratified* types, which allow definitions of types via well-founded recursion over indices.

3.1 Types and Kinds

Besides unit, products and sums, TORES includes a nonstandard function type $(\overrightarrow{u{:}U}); T_1 \to T_2$, which combines a dependent function type and a simple function type. It binds a number of index variables $\overrightarrow{u{:}U}$ which may appear in both T_1 and T_2. If the spine of type declarations is empty then $(\cdot); T_1 \to T_2$ degenerates to the simple function space. We can also quantify existentially over an index using the type $\Sigma u{:}U. T$, and have a type for index equality $M = N$. These two types are useful for expressing equality constraints on indices. We model (co)recursive and stratified types as type constructors of kind $\Pi\overrightarrow{u{:}U}. *$. These introduce type variables X, which we track in the type variable context Ξ. There is no positivity condition on recursive types, as the typing rules for Mendler-recursion enforce termination without it.

A stratified type is defined by primitive recursion on an index term. For the index type nat, the two branches correspond to the two constructors 0 and suc. Intuitively, $T_{\mathtt{Rec}}\, 0$ will behave like T_0 and $T_{\mathtt{Rec}}\, (\mathsf{suc}\, M)$ will behave like $T_s[M/u, T_{\mathtt{Rec}}\, M/X]$. For richer index languages such as Contextual LF we can generate an appropriate recursion scheme following Pientka and Abel [2015].

Kinds	K	$::= * \mid \Pi u{:}U.\, K$
Types	T	$::= 1 \mid T_1 \times T_2 \mid T_1 + T_2 \mid (\overrightarrow{u{:}U}); T_1 \to T_2 \mid \Sigma u{:}U. T \mid M = N$ $\mid T\, M \mid \Lambda\, u. T \mid X \mid \mu X{:}K. T \mid \nu X{:}K. T \mid T_{\mathtt{Rec}}$
Stratified Types	$T_{\mathtt{Rec}}$	$::= \mathtt{Rec}_K(0 \mapsto T_0 \mid \mathsf{suc}\, u,\ X \mapsto T_s)$
Index Contexts	Δ	$::= \cdot \mid \Delta, u{:}U$
Type Var. Contexts	Ξ	$::= \cdot \mid \Xi, X{:}K$
Typing Contexts	Γ	$::= \cdot \mid \Gamma, x{:}T$

▶ **Example 3.** We illustrate indexed recursive types and stratified types using vectors, i.e. lists indexed by their length, with elements of type A. Vectors are of kind $\Pi n{:}\, \mathsf{nat}. *$. We

omit the kind annotation for better readability in the subsequent type definitions. One way to define vectors is with an indexed recursive type, an explicit equality and an existential type: $\mathsf{Vec}_\mu \equiv \mu V.\,\Lambda\, n.\, n = 0 + \Sigma m{:}\,\mathsf{nat}\,.\, n = \mathsf{suc}\, m \times (A \times V\, m)$.

Alternatively, they can be defined as a stratified type: $\mathsf{Vec}_S \equiv \mathtt{Rec}\,(0 \mapsto 1 \mid \mathsf{suc}\, m,\, V \mapsto A \times V)$. In this case equality reasoning is implicit. While we have a choice how to define vectors, some types are only possible to encode using one form or the other.

▶ **Example 4.** A type that must be stratified is the encoding of reducibility for simply typed lambda terms. This example is explored in detail by Cave and Pientka [2013]; our work gives it theoretical justification.

Here the index objects are the simple types, `unit` and `arr` A B of index type `tp`, as well as lambda terms $()$, `lam` $x.M$ and `app` M N of index type `tm`. We can define reducibility as a stratified type of kind $\Pi a{:}\mathtt{tp}.\Pi m{:}\mathtt{tm}.*$. This relies on an indexed recursive type `Halt` (omitted here) that describes when a term m steps to a value.

$$\begin{aligned} \mathsf{Red} \equiv \mathtt{Rec}\ (\ & \mathtt{unit} && \mapsto \Lambda m.\mathtt{Halt}\ m \\ \mid\ & \mathtt{arr}\ a\ b, R_a, R_b && \mapsto \Lambda m.\mathtt{Halt}\ m \times (n{:}\mathtt{tm});\ R_a\ n \to R_b\ (\mathtt{app}\ m\ n)\ ) \end{aligned}$$

▶ **Example 5.** To illustrate a corecursive type, we define an indexed stream of bits following Thibodeau et al. [2016]. The index here guarantees that we are reading exactly m bits. Once $m = 0$, we read a new message consisting of the length of the message n together with a stream indexed by n. In contrast to the recursive type definition for vectors, here the equality constraints guard the observations we can make about a stream.

$$\begin{aligned} \mathsf{Stream} \equiv \nu \mathrm{Str}.\,\Lambda\, m.\quad & (\cdot);\ m = 0 && \to \Sigma n{:}\,\mathsf{nat}\,.\,\mathrm{Str}\ n \\ \times\ & (n{:}\,\mathsf{nat});\ m = \mathsf{suc}\, n && \to \mathrm{Str}\ n \\ \times\ & (n{:}\,\mathsf{nat});\ m = \mathsf{suc}\, n && \to \mathrm{Bit} \end{aligned}$$

3.2 Terms

Tores contains many common constructs found in functional programming languages, such as unit, pairs and case expressions. We focus on the less standard constructs: indexed functions, equality witnesses, well-founded recursion and index induction.

$$\begin{aligned} \text{Terms } t, s ::=\ & x \mid \langle\rangle \mid \lambda\, \vec{u}, x.\, t \mid t\, \vec{M}\, s \mid \langle t_1, t_2\rangle \mid \mathtt{split}\, s\, \mathtt{as}\, \langle x_1, x_2\rangle\, \mathtt{in}\, t \\ \mid\ & \mathtt{in}_i\, t \mid (\mathtt{case}\, t\, \mathtt{of}\, \mathtt{in}_1\ x_1 \mapsto t_1 \mid \mathtt{in}_2\ x_2 \mapsto t_2) \\ \mid\ & \mathtt{pack}\,(M, t) \mid \mathtt{unpack}\, t\, \mathtt{as}\, (u, x)\, \mathtt{in}\, s \\ \mid\ & \mathtt{refl} \mid \mathtt{eq}\, s\, \mathtt{with}\, (\Delta.\Theta\ \mapsto\ t) \mid \mathtt{eq_abort}\, s \\ \mid\ & \mathtt{in}_\mu\, t \mid \mathtt{rec}\, f.\, t \mid \mathtt{corec}\, f.\, t \mid \mathtt{out}_\nu\, t \mid \mathtt{in}_l\, t \mid \mathtt{out}_l\, t \mid \mathtt{ind}\, t_0\, (u,\ f.\, t_s) \mid t{:}T \end{aligned}$$

Since we combine the dependent and simple function types in $(\overrightarrow{u{:}U});\ T_1 \to T_2$, we similarly combine abstraction over index variables $\vec{u}$ and a term variable x in our function term $\lambda\, \vec{u}, x.\, t$. The corresponding application form is $t\, \vec{M}\, s$. The term t of function type $(\overrightarrow{u{:}U});\ T_1 \to T_2$ receives first a spine $\vec{M}$ of index objects followed by a term s. Each equality type $M = N$ has at most one inhabitant `refl` witnessing the equality. There are two elimination forms for equality: the term $\mathtt{eq}\, s\, \mathtt{with}\, (\Delta.\Theta\ \mapsto\ t)$ uses an equality proof s for $M = N$ together with a unifier Θ to refine the body t in a new index context Δ. It may also be the case that the equality witness s is false, in which case we have reached a contradiction and abort using the term $\mathtt{eq_abort}\, s$. Both forms are necessary to make use of equality constraints that arise from indexed type definitions and to show that some cases are impossible.

Recursive types are introduced by the "fold" syntax $\mathtt{in}_\mu$, and stratified types are introduced by $\mathtt{in}_l$ terms. Here l ranges over constructors in the index language such as 0 and suc. The important difference is how we eliminate recursive and stratified types. We can analyze data defined by a recursive type using Mendler-style recursion $\mathtt{rec}\, f.\, t$. This gives a powerful means of recursion while still ensuring termination. Stratified types can only be unfolded using $\mathtt{out}_l$ according to the index. To take full advantage of stratified types, we also allow programmers to use well-founded recursion over index objects, writing $\mathtt{ind}\, t_0\, (u,\ f.\, t_s)$. Intuitively, if the index object is 0, then we pick the first branch and execute t_0; if the index object is $\mathsf{suc}\, M$ then we pick the second branch instantiating u with M and allowing recursive calls f inside t_s. While this induction principle is specific to natural numbers, it can also be derived for other index domains, in particular contextual LF (see Pientka and Abel [2015]).

▶ **Example 6.** Recall that vectors can be defined using the indexed recursive type Vec_μ or the stratified type Vec_S. Which definition we choose impacts how we write programs that analyze vectors. We show the difference using a recursive function that copies a vector.

$$\begin{array}{l} \mathit{copy} : (n{:}\,\mathsf{nat});\ \mathsf{Vec}_\mu\, n \to \mathsf{Vec}_\mu\, n \equiv \mathtt{rec}\, f.\, \lambda\, n, v.\, \mathtt{case}\, v\, \mathtt{of} \\ \qquad \begin{array}{lll} \mid \mathtt{in}_1\, z & \mapsto & \mathtt{in}_\mu\, (\mathtt{in}_1\, z) \\ \mid \mathtt{in}_2\, s & \mapsto & \mathtt{unpack}\, s\, \mathtt{as}\, (m, p)\, \mathtt{in} \\ & & \mathtt{split}\, p\, \mathtt{as}\, \langle e, p' \rangle\, \mathtt{in} \\ & & \mathtt{split}\, p'\, \mathtt{as}\, \langle h, t \rangle\, \mathtt{in} \\ & & \mathtt{in}_\mu\, (\mathtt{in}_2\, (\mathtt{pack}\, (m, \langle e, \langle h, f\, m\, t \rangle \rangle))) \end{array} \end{array}$$

To analyze the recursively defined vector, we use recursion and case analysis of the input vector to reconstruct the output vector. If we receive a non-empty list, we take it apart and expose the equality proofs, before reassembling the list. The recursion is valid according to the Mendler typing rule since the recursive call to f is made on the tail of the input vector. The program is fairly verbose due to the need to unpack the Σ-type and to split pairs. We also need to inject values into the recursive type using the $\mathtt{in}_\mu$ tag. In general, we may also need to reason explicitly with equality constraints.

To contrast we show the program using induction on natural numbers and unfolding the stratified type definition of Vec_S. Note that the first argument is the natural number index n paired with a unit term argument, since index abstraction is always combined with term abstraction. The program analyzes n and in the suc case unfolds the input vector before reconstructing it using the result of the recursive call. In this version of *copy* the equality constraints are handled silently by the type checker.

$$\begin{array}{l} \mathit{copy} : (n{:}\,\mathsf{nat});\ 1 \to (\cdot);\ \mathsf{Vec}_S\, n \to \mathsf{Vec}_S\, n \equiv \\ \quad \begin{array}{lll} \mathtt{ind}\, (\ 0 & \mapsto & \lambda\, v.\, \mathtt{in}_0\, \langle\rangle \\ \quad \mid \mathsf{suc}\, m,\ f_m & \mapsto & \lambda\, v.\, \mathtt{split}\, (\mathtt{out}_{\mathsf{suc}}\, v)\, \mathtt{as}\, \langle h, t \rangle\, \mathtt{in}\ \ \mathtt{in}_{\mathsf{suc}}\, \langle h, f_m\, t \rangle) \end{array} \end{array}$$

▶ **Example 7.** Let us now build streams using Mendler-style corecursion. Streams of natural numbers are defined as $\nu X{:}K.\ \mathsf{nat} \times X$. We can define a stream of natural numbers starting from n as:

$$\mathit{natsFrom} : \mathsf{nat} \to \mathsf{Stream} \equiv \mathtt{corec}\, f.\, \lambda\, n.\, \langle n, f\ (\mathsf{suc}\ \ n) \rangle$$

Here, the corecursion constructor takes a function whose argument is the seed used to build the stream and produces the pair of the head and the tail of our stream. In this case, the seed is simply the natural number corresponding to the current head of the stream. As we move to the tail, we simply use the corecursive call made available by f on the successor of the seed, as the next element of the stream will be this new number.

273 Let us now a second example: the stream of Fibonacci numbers.

274 $$fibFrom : \mathsf{nat} \times \mathsf{nat} \to \mathsf{Stream} \equiv \mathtt{corec}\, f.\, \lambda\, s.\, \mathtt{split}\, s\, \mathtt{as}\, \langle m, n\rangle\, \mathtt{in}\, \langle m, f\ \langle n, m+n\rangle\rangle$$

275 In order to build the Fibonacci stream, we use as a seed a pair of two numbers corresponding
276 of the last two Fibonacci numbers that have been computed so far. From there, the head
277 of the stream is simply the first number of the pair, while the new seed is simply the
278 second number together with the sum of the two, representing the second and third number,
279 respectively. Hence, to obtain the whole Fibonacci stream, we simply write $fibFrom\ \langle 0, 1\rangle$.

280 ▶ **Example 8.** Note that TORES does not have an explicit notion of falsehood. This is
281 because it is definable using existing constructs: we can define the empty type as a recursive
282 type $\bot \equiv \mu X{:}\, *\, .\, X$, and a contradiction term $\mathtt{abort} \equiv \mathtt{rec}\, f.\, f : \bot \to C$, for any type C.
283 Our termination result with the logical relation in Section 4.3 shows that the $\bot$ type contains
284 no values and hence no closed terms, which implies logical consistency of TORES (not all
285 propositions can be proven).

286 ## 3.3 Typing Rules

287 We define a bidirectional type system in Fig. 3 with two mutually defined judgments: checking
288 a term t against a type T and synthesizing a type T for a term t.. We can move from checking
289 to synthesis via the conversion rule and from synthesis to checking using a type annotation.
290 The typing rules for unit, products, sums and existentials are standard.We focus here on
291 equality, recursive and stratified types.

292 The introduction for an index equality type is simply $\mathtt{refl}$, which is checked via equality
293 in the index domain. Both equality elimination forms rely on unification in the index domain
294 (see Section 2.3). Specifically, the $\mathtt{eq_abort}\, s$ term checks against any type because the
295 unification must fail, establishing a contradiction. For the term $\mathtt{eq}\, s\, \mathtt{with}\, (\Delta'.\Theta \mapsto t)$,
296 unification must result in the MGU which by Req. 4 is α-equivalent to the supplied unifier
297 $(\Delta' \mid \Theta)$. We then check the body t using the new index context Δ' and Θ applied to the
298 contexts Ξ and Γ and the goal type T.

299 This treatment of equality elimination is similar to the use of refinement substitutions
300 for dependent pattern matching [Pientka and Dunfield, 2008, Cave and Pientka, 2012], and
301 is inspired by equality elimination in proof theory [Tiu and Momigliano, 2012, McDowell
302 and Miller, 2002, Schroeder-Heister, 1993]. In the latter line of work, type checking involves
303 trying all unifiers from a *complete set of unifiers* (which may be infinite!), instead of a single
304 most general unifier. We believe our requirement for a unique MGU is a practical choice for
305 type checking.

306 Indexed recursive and stratified types are both introduced by injections ($\mathtt{in}_\mu$ and $\mathtt{in}_l$),
307 though their elimination forms are different. Stratified types are eliminated (unfolded) in
308 reverse to the corresponding fold rules. For recursive types on the other hand, the naive unfold
309 rules lead to nontermination, so we use a Mendler-style recursion form $\mathtt{rec}\, f.\, t$, generalizing
310 the original formulation [Mendler, 1988] to an indexed type system. The idea is to constrain
311 the type of the function variable f so that it can only be applied to structurally smaller data.
312 This is achieved by declaring f of type $(\overrightarrow{u{:}U})$; $X\ \vec{u} \to T$ in the premise of the rule. Here X
313 represents types exactly one constructor smaller than the recursive type, so the use of f is
314 guaranteed to be well-founded.

315 ▶ **Theorem 9.** *Type checking of terms is decidable.*

$\boxed{\Delta;\Xi;\Gamma \vdash t \Leftarrow T}$ Term t checks against input type T

$$\frac{\Delta;\Xi;\Gamma \vdash t_1 \Leftarrow T_1 \quad \Delta;\Xi;\Gamma \vdash t_2 \Leftarrow T_2}{\Delta;\Xi;\Gamma \vdash \langle t_1, t_2\rangle \Leftarrow T_1 \times T_2} \qquad \frac{\Delta;\Xi;\Gamma \vdash p \Rightarrow T_1 \times T_2 \quad \Delta;\Xi;\Gamma, x_1{:}T_1, x_2{:}T_2 \vdash t \Leftarrow T}{\Delta;\Xi;\Gamma \vdash \texttt{split}\, p\, \texttt{as}\, \langle x_1, x_2\rangle\, \texttt{in}\, t \Leftarrow T}$$

$$\frac{\Delta;\Xi;\Gamma \vdash t \Leftarrow T_i}{\Delta;\Xi;\Gamma \vdash \texttt{in}_i\, t \Leftarrow T_1 + T_2} \qquad \frac{\Delta;\Xi;\Gamma \vdash t \Rightarrow T_1 + T_2 \quad \Delta;\Xi;\Gamma, x_1{:}T_1 \vdash t_1 \Leftarrow S \quad \Delta;\Xi;\Gamma, x_2{:}T_2 \vdash t_2 \Leftarrow S}{\Delta;\Xi;\Gamma \vdash (\texttt{case}\, t\, \texttt{of}\, \texttt{in}_1\, x_1 \mapsto t_1 \mid \texttt{in}_2\, x_2 \mapsto t_2) \Leftarrow S}$$

$$\frac{\Delta \vdash M : U \quad \Delta;\Xi;\Gamma \vdash t \Leftarrow T[M/u]}{\Delta;\Xi;\Gamma \vdash \texttt{pack}\,(M, t) \Leftarrow \Sigma u{:}U.\,T} \qquad \frac{\Delta;\Xi;\Gamma \vdash t \Rightarrow \Sigma u{:}U.\,T \quad \Delta, u{:}U;\Xi;\Gamma, x{:}T \vdash s \Leftarrow S}{\Delta;\Xi;\Gamma \vdash \texttt{unpack}\, t\, \texttt{as}\,(u, x)\, \texttt{in}\, s \Leftarrow S}$$

$$\frac{\Delta \vdash M = N}{\Delta;\Xi;\Gamma \vdash \texttt{refl} \Leftarrow M = N} \qquad \frac{\Delta;\Xi;\Gamma \vdash s \Rightarrow M = N \quad \Delta \vdash M \doteq N \searrow \#}{\Delta;\Xi;\Gamma \vdash \texttt{eq_abort}\, s \Leftarrow T}$$

$$\frac{\Delta;\Xi;\Gamma \vdash s \Rightarrow M = N \quad \Delta \vdash M \doteq N \searrow (\Delta' \mid \Theta) \quad \Delta';\Xi[\Theta];\Gamma[\Theta] \vdash t \Leftarrow T[\Theta]}{\Delta;\Xi;\Gamma \vdash \texttt{eq}\, s\, \texttt{with}\,(\Delta'.\Theta \mapsto t) \Leftarrow T}$$

$$\frac{\Delta;\Xi;\Gamma \vdash t \Leftarrow T[\overrightarrow{M/u}; \mu X{:}K.\,\Lambda \vec{u}.\,T/X]}{\Delta;\Xi;\Gamma \vdash \texttt{in}_\mu\, t \Leftarrow (\mu X{:}K.\,\Lambda \vec{u}.\,T)\,\vec{M}} \qquad \frac{\Delta;\Xi, X{:}K;\Gamma, f{:}((\overrightarrow{u{:}U});\, X\vec{u} \to T) \vdash t \Leftarrow (\overrightarrow{u{:}U});\, S[\overrightarrow{u/v}] \to T}{\Delta;\Xi;\Gamma \vdash \texttt{rec}\, f.\,t \Leftarrow (\overrightarrow{u{:}U});\, (\mu X{:}K.\,\Lambda \vec{v}.\,S)\,\vec{u} \to T}$$

$$\frac{\Delta;\Xi, X{:}K;\Gamma, f{:}((\overrightarrow{u{:}U});\, S \to X\vec{u}) \vdash t \Leftarrow (\overrightarrow{u{:}U});\, S \to T[\overrightarrow{u/v}]}{\Delta;\Xi;\Gamma \vdash \texttt{corec}\, f.\,t \Leftarrow (\overrightarrow{u{:}U});\, S \to (\nu X{:}K.\,\Lambda \vec{v}.\,T)\,\vec{u}} \qquad \frac{\Delta, \overrightarrow{u{:}U};\Xi;\Gamma, x{:}S \vdash t \Leftarrow T}{\Delta;\Xi;\Gamma \vdash \lambda\, \vec{u}, x.\,t \Leftarrow (\overrightarrow{u{:}U});\, S \to T}$$

$$\frac{\Delta;\Xi;\Gamma \vdash t_0 \Leftarrow T[0/u] \quad \Delta, u{:}\,\mathsf{nat};\Xi;\Gamma, f{:}T \vdash t_s \Leftarrow T[\mathsf{suc}\, u/u]}{\Delta;\Xi;\Gamma \vdash \texttt{ind}\, t_0\,(u,\ f.\,t_s) \Leftarrow (u{:}\,\mathsf{nat});\, 1 \to T} \qquad \frac{}{\Delta;\Xi;\Gamma \vdash \langle\rangle \Leftarrow 1}$$

$$\frac{\Delta;\Xi;\Gamma \vdash t \Leftarrow T_0\, \vec{M}}{\Delta;\Xi;\Gamma \vdash \texttt{in}_0\, t \Leftarrow T_{\mathsf{Rec}}\, 0\, \vec{M}} \qquad \frac{\Delta;\Xi;\Gamma \vdash t \Leftarrow T_s[N/u; (T_{\mathsf{Rec}}\, N)/X]\, \vec{M}}{\Delta;\Xi;\Gamma \vdash \texttt{in}_{\mathsf{suc}}\, t \Leftarrow T_{\mathsf{Rec}}\,(\mathsf{suc}\, N)\, \vec{M}} \qquad \frac{\Delta;\Xi;\Gamma \vdash t \Rightarrow T}{\Delta;\Xi;\Gamma \vdash t \Leftarrow T}$$

$\boxed{\Delta;\Xi;\Gamma \vdash t \Rightarrow T}$ Term t synthesizes output type T

$$\frac{x{:}T \in \Gamma}{\Delta;\Xi;\Gamma \vdash x \Rightarrow T} \qquad \frac{\Delta;\Xi;\Gamma \vdash t \Rightarrow (\overrightarrow{u{:}U});\, S \to T \quad \Delta \vdash \vec{M} : (\overrightarrow{u{:}U}) \quad \Delta;\Xi;\Gamma \vdash s \Leftarrow S[\overrightarrow{M/u}]}{\Delta;\Xi;\Gamma \vdash t\, \vec{M}\, s \Rightarrow T[\overrightarrow{M/u}]}$$

$$\frac{\Delta;\Xi;\Gamma \vdash t \Rightarrow T_{\mathsf{Rec}}\, 0\, \vec{M}}{\Delta;\Xi;\Gamma \vdash \texttt{out}_0\, t \Rightarrow T_0\, \vec{M}} \qquad \frac{\Delta;\Xi;\Gamma \vdash t \Rightarrow T_{\mathsf{Rec}}\,(\mathsf{suc}\, N)\, \vec{M}}{\Delta;\Xi;\Gamma \vdash \texttt{out}_{\mathsf{suc}}\, t \Rightarrow T_s[N/u; (T_{\mathsf{Rec}}\, N)/X]\, \vec{M}} \qquad \frac{\Delta;\Xi;\Gamma \vdash t \Leftarrow T}{\Delta;\Xi;\Gamma \vdash t{:}T \Rightarrow T}$$

$$\frac{\Delta;\Xi;\Gamma \vdash t \Rightarrow (\nu X{:}K.\,\Lambda \vec{u}.\,T)\,\vec{M}}{\Delta;\Xi;\Gamma \vdash \texttt{out}_\nu\, t \Rightarrow T[\overrightarrow{M/u}; \nu X{:}K.\,\Lambda \vec{u}.\,T/X]}$$

Figure 3 Typing rules for TORES

Proof. Since the typing rules are syntax directed, it is straight-forward to extract a type checking algorithm. Note that the algorithm relies on decidability of judgments in the index language, namely index type checking (Req. 1), equality (Req. 2) and unification (Req. 4). ◀

3.4 Operational Semantics

We define a big-step operational semantics using environments, which provide closed values for the free variables that may occur in a term.

$$\begin{array}{llll} \text{Term environments} & \sigma & := \cdot \mid \sigma, v/x \\ \text{Function values} & g & := \lambda\,\vec{u}, x.\,t \mid \texttt{rec}\,f.\,t \mid \texttt{corec}\,f.\,t \mid \texttt{ind}\,t_0\,(u,\ f.\,t_s) \\ \text{Closures} & c & := (g)[\theta;\sigma] \mid (\texttt{corec}\,f.\,t)[\theta;\sigma]\cdot\vec{N}\ v \\ \text{Values} & v & := c \mid \langle\rangle \mid \langle v_1, v_2\rangle \mid \texttt{in}_i\,v \mid \texttt{pack}\,(M,v) \mid \texttt{refl} \mid \texttt{in}_\mu\,v \mid \texttt{in}_l\,v \end{array}$$

Values consist of unit, pairs, injections, reflexivity, and closures. Typing for values and environments, which is used to state the subject reduction theorem, are given in Fig. 6 in the appendixin the appendix.

The main evaluation judgment, $t[\theta;\sigma] \Downarrow v$, describes the evaluation of a term t under environments $\theta;\sigma$ to a value v. Here, t stands for a term in an index context Δ and term variable context Γ. The index environment θ provides closed index objects for all the index variables in Δ, while σ provides closed values for all the variables declared in Γ, i.e. $\vdash \theta : \Delta$ and $\sigma : \Gamma[\theta]$. For convenience, we factor out the application of a closure c to values $\vec{N}$ and v resulting in a value w, using a second judgment written $c \cdot \vec{N}\, v \Downarrow w$. This allows us to treat application of functions (lambdas, recursion and induction) uniformly. Similarly, we factor out the application of $\texttt{out}_\nu$ to a closure c in an additional judgment written $c \cdot_{\texttt{out}_\nu} \Downarrow w$. This simplifies the type interpretation used to prove termination.

We only explain the evaluation rule for equality elimination $\texttt{eq}\,s\,\texttt{with}\,(\Delta.\Theta \mapsto t)$. We first evaluate the equality witness s under environments $\theta;\sigma$ to the value $\texttt{refl}$. This ensures that θ respects the index equality $M = N$ witnessed by s. From type checking we know that $\Delta \vdash M[\Theta] = N[\Theta]$: the key is how we extend Θ at run-time to produce a new index environment θ' that is consistent with θ. This relies on sound and complete index substitution matching (see Section 2.3) to generate θ' such that $\cdot \vdash \theta' : \Delta$ and $\cdot \vdash \Theta[\theta'] = \theta$. We can then evaluate the body t under the new index environment θ' and the same term environment σ to produce a value v.

Notably absent is an evaluation rule for $\texttt{eq_abort}\,t$. This term is used in a branch of a case split that we know statically to be impossible. Such branches are never reached at run time, so there is no need for an evaluation rule. For example, consider a type-safe "head" function, which receives a nonempty vector as input. As we write each branch of a case split explicitly, the empty list case must use $\texttt{eq_abort}\,t$, but is never executed. We now state subject reduction for TORES.

▶ **Theorem 10** (Subject Reduction)**.**

1. *If* $t[\theta;\sigma] \Downarrow v$ *where* $\Delta;\cdot;\Gamma \vdash t \Leftarrow T$ *or* $\Delta;\cdot;\Gamma \vdash t \Rightarrow T$*, and* $\vdash \theta : \Delta$ *and* $\sigma : \Gamma[\theta]$*, then* $v : T[\theta]$.
2. *If* $g[\theta;\sigma] \cdot \vec{N}\, v \Downarrow w$ *where* $\Delta;\cdot;\Gamma \vdash g \Leftarrow (\overrightarrow{u{:}U});\, S \to T$ *and* $\vdash \theta : \Delta$ *and* $\sigma : \Gamma[\theta]$ *and* $\vdash \vec{N} : (\overrightarrow{u{:}U})[\theta]$ *and* $v : S[\theta, \overrightarrow{N/u}]$*, then* $w : T[\theta, \overrightarrow{N/u}]$.
3. *If* $c \cdot_{\texttt{out}_\nu} \Downarrow w$ *where* $c : (\nu X{:}K.\, \Lambda\,\vec{u}.\,T)\ \vec{M}$ *then* $w : T[\overrightarrow{M/u}; (\nu X{:}K.\,\Lambda\,\vec{u}.\,T)/X]$.

4 Termination Proof

We now describe our main technical result: termination of evaluation. Our proof uses the logical predicate technique of Tait [1967] and Girard [1972]. We interpret each language construct (index types, kinds, types, etc.) into a semantic model of sets and functions.

$\boxed{t[\theta;\sigma] \Downarrow v}$ Term t under environments θ and σ evaluates to v

$$\frac{\sigma(x) = v}{x[\theta;\sigma] \Downarrow v} \quad \frac{}{\langle\rangle[\theta;\sigma] \Downarrow \langle\rangle} \quad \frac{t_1[\theta;\sigma] \Downarrow v_1 \quad t_2[\theta;\sigma] \Downarrow v_2}{\langle t_1, t_2\rangle[\theta;\sigma] \Downarrow \langle v_1, v_2\rangle} \quad \frac{t[\theta;\sigma] \Downarrow \langle v_1, v_2\rangle \quad s[\theta;\sigma, v_1/x_1, v_2/x_2] \Downarrow v}{(\mathtt{split}\, t\, \mathtt{as}\, \langle x_1, x_2\rangle\, \mathtt{in}\, s)[\theta;\sigma] \Downarrow v}$$

$$\frac{t[\theta;\sigma] \Downarrow v}{(\mathtt{in}_i\, t)[\theta;\sigma] \Downarrow \mathtt{in}_i\, v} \quad \frac{t[\theta;\sigma] \Downarrow \mathtt{in}_i\, v' \quad t_i[\theta;\sigma, v'/x_i] \Downarrow v}{(\mathtt{case}\, t\, \mathtt{of}\, \mathtt{in}_1\; x_1 \mapsto t_1 \mid \mathtt{in}_2\; x_2 \mapsto t_2)[\theta;\sigma] \Downarrow v} \quad \frac{t[\theta;\sigma] \Downarrow v}{(t{:}T)[\theta;\sigma] \Downarrow v}$$

$$\frac{t[\theta;\sigma] \Downarrow v}{(\mathtt{pack}\,(M, t))[\theta;\sigma] \Downarrow \mathtt{pack}\,(M[\theta], v)} \quad \frac{t[\theta;\sigma] \Downarrow \mathtt{pack}\,(N, v') \quad s[\theta, N/u;\sigma, v'/x] \Downarrow v}{(\mathtt{unpack}\, t\, \mathtt{as}\, (u, x)\, \mathtt{in}\, s)[\theta;\sigma] \Downarrow v}$$

$$\frac{}{\mathtt{refl}[\theta;\sigma] \Downarrow \mathtt{refl}} \quad \frac{s[\theta;\sigma] \Downarrow \mathtt{refl} \quad \Delta \vdash \Theta \doteq \theta \searrow (\cdot \mid \theta') \quad t[\theta';\sigma] \Downarrow v}{(\mathtt{eq}\, s\, \mathtt{with}\, (\Delta.\Theta \;\mapsto\; t)\,)[\theta;\sigma] \Downarrow v} \quad \frac{t[\theta;\sigma] \Downarrow v}{(\mathtt{in}_l\, t)[\theta;\sigma] \Downarrow \mathtt{in}_l\, v}$$

$$\frac{}{(\lambda\, \vec{u}, x.\, t)[\theta;\sigma] \Downarrow (\lambda\, \vec{u}, x.\, t)[\theta;\sigma]} \quad \frac{}{(\mathtt{rec}\, f.\, t)[\theta;\sigma] \Downarrow (\mathtt{rec}\, f.\, t)[\theta;\sigma]} \quad \frac{t[\theta;\sigma] \Downarrow \mathtt{in}_l\, v}{(\mathtt{out}_l\, t)[\theta;\sigma] \Downarrow v}$$

$$\frac{}{(\mathtt{corec}\, f.\, t)[\theta;\sigma] \Downarrow (\mathtt{corec}\, f.\, t)[\theta;\sigma]} \quad \frac{t[\theta;\sigma] \Downarrow c \quad c \cdot_{\mathtt{out}_\nu} \Downarrow w}{(\mathtt{out}_\nu\, t)[\theta;\sigma] \Downarrow w}$$

$$\frac{}{(\mathtt{ind}\, t_0\; (u,\; f.\, t_s))[\theta;\sigma] \Downarrow (\mathtt{ind}\, t_0\; (u,\; f.\, t_s))[\theta;\sigma]} \quad \frac{t[\theta;\sigma] \Downarrow c \quad s[\theta;\sigma] \Downarrow v \quad c \cdot \overrightarrow{M[\theta]}\, v \Downarrow w}{(t\, \vec{M}\, s)[\theta;\sigma] \Downarrow w}$$

$\boxed{c \cdot \vec{N}\, v \Downarrow w}$ Closure c applied to values $\vec{N}$ and v evaluates to w

$$\frac{t[\theta, \overrightarrow{N/u};\sigma, v/x] \Downarrow w}{(\lambda\, \vec{u}, x.\, t)[\theta;\sigma] \cdot \vec{N}\, v \Downarrow w} \quad \frac{t[\theta;\sigma, (\mathtt{rec}\, f.\, t)[\theta;\sigma]/f] \Downarrow c \quad c \cdot \vec{N}\, v \Downarrow w}{(\mathtt{rec}\, f.\, t)[\theta;\sigma] \cdot \vec{N}\, (\mathtt{in}_\mu\, v) \Downarrow w}$$

$$\frac{}{(\mathtt{corec}\, f.\, t)[\theta;\sigma] \cdot \vec{N}\, v \Downarrow (\mathtt{corec}\, f.\, t)[\theta;\sigma] \cdot \vec{N}\;\, v}$$

$$\frac{t_0[\theta;\sigma] \Downarrow w}{(\mathtt{ind}\, t_0\; (u,\; f.\, t_s))[\theta;\sigma] \cdot 0\, \langle\rangle \Downarrow w} \quad \frac{(\mathtt{ind}\, t_0\; (u,\; f.\, t_s))[\theta;\sigma] \cdot N\, \langle\rangle \Downarrow v \quad t_s[\theta, N/u;\sigma, v/f] \Downarrow w}{(\mathtt{ind}\, t_0\; (u,\; f.\, t_s))[\theta;\sigma] \cdot (\mathsf{suc}\, N)\, \langle\rangle \Downarrow w}$$

$\boxed{c \cdot_{\mathtt{out}_\nu} \Downarrow w}$ Closure c applied to observation $\mathtt{out}_\nu$ evaluates to w

$$\frac{t[\theta;\sigma, (\mathtt{corec}\, f.\, t)[\theta;\sigma]/f] \Downarrow c \quad c \cdot \vec{N}\, v \Downarrow w}{((\mathtt{corec}\, f.\, t)[\theta;\sigma] \cdot \vec{N}\;\, v)\; \cdot_{\mathtt{out}_\nu} \Downarrow w}$$

Figure 4 Big-step evaluation rules

4.1 Interpretation of Index Language

We start with the interpretations for index types and spines. In general, our index language may be dependently typed, as it is if we choose Contextual LF. Hence our interpretation for index types U must take into account an environment θ containing instantiations for index variables u. Such an index environment θ is simply a grounding substitution $\vdash \theta : \Delta$.

▶ **Definition 11** (Interpretation of index types $[\![U]\!]$ and index spines $[\![\overrightarrow{u{:}U}]\!]$).

$$\begin{array}{lcl} [\![U]\!](\theta) & = & \{M \mid \cdot \vdash M : U[\theta]\} \\ [\![(\cdot)]\!](\theta) & = & \{\cdot\} \\ [\![(u_0{:}U_0, \overrightarrow{u{:}U})]\!](\theta) & = & \{M_0, \vec{M} \mid M_0 \in [\![U_0]\!](\theta), \vec{M} \in [\![(\overrightarrow{u{:}U})]\!](\theta, M_0/u_0)\} \end{array}$$

The interpretation of an index type U under environment θ is the set of closed terms of type $U[\theta]$. The interpretation lifts to index spines $(\overrightarrow{u{:}U})$. With these definitions, the following lemma follows from the substitution principles of index terms (Req. 3).

▶ **Lemma 12** (Interpretation of index substitution)**.**

12.1. *If* $\Delta \vdash M : U$ *and* $\vdash \theta : \Delta$ *then* $M[\theta] \in [\![U]\!](\theta)$*.*

12.2. *If* $\Delta \vdash \vec{M} : (\overrightarrow{u{:}U})$ *and* $\vdash \theta : \Delta$ *then* $\overrightarrow{M[\theta]} \in [\![(\overrightarrow{u{:}U})]\!](\theta)$*.*

4.2 Lattice Interpretation of Kinds

We now describe the lattice structure that underlies the interpretation of kinds in our language. The idea is that types are interpreted as sets of term-level values and type constructors as functions taking indices to sets of values. We call the set of all term-level values Ω and write its power set as $\mathcal{P}(\Omega)$. The interpretation is defined inductively on the structure of kinds.

▶ **Definition 13** (Interpretation of kinds $[\![K]\!]$)**.**

$$\begin{array}{lcl} [\![*]\!](\theta) & = & \mathcal{P}(\Omega) \\ [\![\Pi u{:}U.\, K]\!](\theta) & = & \{\mathcal{C} \mid \forall M \in [\![U]\!](\theta).\, \mathcal{C}(M) \in [\![K]\!](\theta, M/u)\} \end{array}$$

A key observation in our metatheory is that each $[\![K]\!](\theta)$ forms a *complete lattice*. In the base case, $[\![*]\!](\theta) = \mathcal{P}(\Omega)$ is a complete lattice under the subset ordering, with meet and join given by intersection and union respectively. For a kind $K = \Pi u{:}U.\, K'$, we induce a lattice structure on $[\![K]\!](\theta)$ by lifting the lattice operations pointwise. Precisely, we define

$$\mathcal{A} \leq_{[\![K]\!](\theta)} \mathcal{B} \quad \text{iff} \quad \forall M \in [\![U]\!](\theta).\, \mathcal{A}(M) \leq_{[\![K']\!](\theta, M/u)} \mathcal{B}(M).$$

The meet and join operations can similarly be lifted pointwise.

This structure is important because it allows us to define pre-fixed points for operators on the lattice, which is central to our interpretation of recursive types. Here we rely on the existence of arbitrary meets, as we take the meet over an impredicatively defined subset of $\mathcal{L}$.

▶ **Definition 14** (Mendler-style pre-fixed and post-fixed points)**.** Suppose $\mathcal{L}$ is a complete lattice and $\mathcal{F} : \mathcal{L} \to \mathcal{L}$. Define $\boldsymbol{\mu}_{\mathcal{L}} : (\mathcal{L} \to \mathcal{L}) \to \mathcal{L}$ by

$$\boldsymbol{\mu}_{\mathcal{L}}\mathcal{F} = \bigwedge \{\mathcal{C} \in \mathcal{L} \mid \forall \mathcal{X} \in \mathcal{L}.\, \mathcal{X} \leq_{\mathcal{L}} \mathcal{C} \implies \mathcal{F}(\mathcal{X}) \leq_{\mathcal{L}} \mathcal{C}\}.$$

and $\boldsymbol{\nu}_{\mathcal{L}} : (\mathcal{L} \to \mathcal{L}) \to \mathcal{L}$ by

$$\boldsymbol{\nu}_{\mathcal{L}}\mathcal{F} = \bigvee \{\mathcal{C} \in \mathcal{L} \mid \forall \mathcal{X} \in \mathcal{L}.\, \mathcal{C} \leq_{\mathcal{L}} \mathcal{X} \implies \mathcal{C} \leq_{\mathcal{L}} \mathcal{F}(\mathcal{X})\}.$$

We will mostly omit the subscript denoting the underlying lattice $\mathcal{L}$ of the order $\leq$ and pre-fixed and post-fixed points, $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$.

Note that a usual treatment of recursive types would define the least pre-fixed point of a monotone operator as $\bigwedge\{\mathcal{C} \in \mathcal{L} \mid \mathcal{F}(\mathcal{C}) \leq \mathcal{C}\}$ and the greatest post-fixed point of a monotone operator as $\bigvee\{\mathcal{C} \in \mathcal{L} \mid \mathcal{C} \leq \mathcal{F}(\mathcal{C})\}$, using the Knaster-Tarski theorem. However, our unconventional definition (following Jacob-Rao et al. [2016]) more closely models Mendler-style (co)recursion and does not require $\mathcal{F}$ to be monotone (thereby avoiding a positivity restriction on recursive types).

401 ## 4.3 Interpretation of Types

402 In order to interpret the types of our language, it is helpful to define semantic versions of
403 some syntactic constructs. We first define a semantic form of our indexed function type
404 $(\overrightarrow{u{:}U});\, T_1 \to T_2$, which helps us formulate the interaction of function types with fixed points
405 and recursion.

406 ▶ **Definition 15** (Semantic function space). For a spine interpretation $\vec{\mathcal{U}}$ and functions
407 $\mathcal{A}, \mathcal{B} : \vec{\mathcal{U}} \to \mathcal{P}(\Omega)$, define $\vec{\mathcal{U}},\, \mathcal{A} \to \mathcal{B} = \{c \mid \forall \vec{M} \in \vec{\mathcal{U}}.\ \forall v \in \mathcal{A}(\vec{M}).\ c \cdot \vec{M}\, v \Downarrow w \in \mathcal{B}(\vec{M})\}$.

408 It will also be convenient to lift term-level `in` tags to the level of sets and functions
409 in the lattice $[\![K]\!](\theta)$. We define the lifted tags $\mathtt{in}^*\ : [\![K]\!](\theta) \to [\![K]\!](\theta)$ inductively on K.
410 If $\mathcal{V} \in [\![*]\!](\theta) = \mathcal{P}(\Omega)$ then $\mathtt{in}^*\, \mathcal{V} = \mathtt{in}\, \mathcal{V} = \{\mathtt{in}\, v \mid v \in \mathcal{V}\}$. If $\mathcal{C} \in [\![\Pi u{:}U.\, K']\!](\theta)$ then
411 $(\mathtt{in}^*\, \mathcal{C})(M) = \mathtt{in}^*\, (\mathcal{C}(M))$ for all $M \in [\![U]\!](\theta)$. Essentially, the $\mathtt{in}^*$ function attaches a tag to
412 every element in the set produced after the index arguments are received.

413 Dually we define $\mathtt{out}^*_\nu\ : [\![K]\!](\theta) \to [\![K]\!](\theta)$. If $\mathcal{V} \in [\![*]\!](\theta) = \mathcal{P}(\Omega)$ then $\mathtt{out}^*_\nu\, \mathcal{V} =$
414 $\mathtt{out}_\nu\, \mathcal{V} = \{c \mid c \cdot_{\mathtt{out}_\nu} \Downarrow w \in \mathcal{V}\}$. If $\mathcal{C} \in [\![\Pi u{:}U.\, K']\!](\theta)$ then $(\mathtt{out}^*_\nu\, \mathcal{C})(M) = \mathtt{out}^*_\nu\, (\mathcal{C}(M))$ for all
415 $M \in [\![U]\!](\theta)$.

416 Finally, we define the interpretation of type variable contexts Ξ. These describe semantic
417 environments η mapping each type variable to an object in its respective kind interpretation.
418 Such environments are necessary to interpret type expressions with free type variables.

▶ **Definition 16** (Interpretation of type variable contexts $[\![\Xi]\!]$).

419
$$\begin{array}{lcl} [\![\cdot]\!](\theta) & = & \{\cdot\} \\ [\![\Xi, X{:}K]\!](\theta) & = & \{\eta, \mathcal{X}/X \mid \eta \in [\![\Xi]\!](\theta), \mathcal{X} \in [\![K]\!](\theta)\} \end{array}$$

420 We are now able to define the interpretation of types T under environments θ and η.
421 This is done inductively on the structure of T.

▶ **Definition 17** (Interpretation of types and constructors).

422
$$\begin{array}{lcl}
[\![1]\!](\theta;\eta) & = & \{\langle\rangle\} \\
[\![T_1 \times T_2]\!](\theta;\eta) & = & \{\langle v_1, v_2\rangle \mid v_1 \in [\![T_1]\!](\theta;\eta), v_2 \in [\![T_2]\!](\theta;\eta)\} \\
[\![T_1 + T_2]\!](\theta;\eta) & = & \mathtt{in}_1\, [\![T_1]\!](\theta;\eta) \bigcup \mathtt{in}_2\, [\![T_2]\!](\theta;\eta) \\
[\![(\overrightarrow{u{:}U});\, T_1 \to T_2]\!](\theta;\eta) & = & [\![(\overrightarrow{u{:}U})]\!](\theta),\, \mathcal{T}_1 \to \mathcal{T}_2 \\
 & & \text{where } \mathcal{T}_i(\vec{M}) = [\![T_i]\!](\theta, \overrightarrow{M/u};\eta) \text{ for } i \in \{1,2\} \\
[\![\Sigma u{:}U.\, T]\!](\theta;\eta) & = & \{\mathtt{pack}\,(M, v) \mid M \in [\![U]\!](\theta), v \in [\![T]\!](\theta, M/u;\eta)\} \\
[\![T\, M]\!](\theta;\eta) & = & [\![T]\!](\theta;\eta)(M[\theta]) \\
[\![M = N]\!](\theta;\eta) & = & \{\mathtt{refl} \mid\ \vdash M[\theta] = N[\theta]\} \\
[\![X]\!](\theta;\eta) & = & \eta(X) \\
[\![\Lambda\, u.\, T]\!](\theta;\eta) & = & (M \mapsto [\![T]\!](\theta, M/u;\eta)) \\
[\![\mu X{:}K.\, T]\!](\theta;\eta) & = & \boldsymbol{\mu}_{[\![K]\!](\theta)}(\mathcal{X} \mapsto \mathtt{in}^*_\mu\, ([\![T]\!](\theta;\eta, \mathcal{X}/X))) \\
[\![\nu X{:}K.\, T]\!](\theta;\eta) & = & \boldsymbol{\nu}_{[\![K]\!](\theta)}(\mathcal{X} \mapsto \mathtt{out}^*_\nu\, ([\![T]\!](\theta;\eta, \mathcal{X}/X))) \\
[\![\mathtt{Rec}_K\, (0 \mapsto T_z \mid \mathsf{suc}\, u,\ X \mapsto T_s)]\!](\theta;\eta) & = & \mathbf{Rec}_{[\![K]\!](\theta)}\, (\mathtt{in}^*_0\, [\![T_z]\!](\theta;\eta)) \\
 & & \quad (N \mapsto \mathcal{X} \mapsto \mathtt{in}^*_{\mathsf{suc}}\, [\![T_s]\!](\theta, N/u;\eta, \mathcal{X}/X))
\end{array}$$

424 423 where

425
$$\begin{array}{llllcl}
\mathbf{Rec}_{\mathcal{L}} : & \mathcal{L} \to & (\mathbb{N} \to \mathcal{L} \to \mathcal{L}) \to & \mathbb{N} & \to & \mathcal{L} \\
\mathbf{Rec}_{\mathcal{L}} & \mathcal{C} & \mathcal{F} & 0 & = & \mathcal{C} \\
\mathbf{Rec}_{\mathcal{L}} & \mathcal{C} & \mathcal{F} & (\mathsf{suc}\, N) & = & \mathcal{F}\, N\, (\mathbf{Rec}_{\mathcal{L}}\, \mathcal{C}\, \mathcal{F}\, N)
\end{array}$$

426 The interpretation of the indexed function type $[\![(\overrightarrow{u{:}U}); T_1 \to T_2]\!](\theta; \eta)$ contains closures
427 which, when applied to values in the appropriate input sets, evaluate to values in the
428 appropriate output set. The interpretation of the equality type $[\![M = N]\!](\theta; \eta)$ is the set
429 $\{\texttt{refl}\}$ if $\vdash M[\theta] = N[\theta]$ and the empty set otherwise. The interpretation of a recursive type
430 is the pre-fixed point of the function obtained from the underlying type expression. Finally,
431 interpretation of a stratified type built from `Rec` relies on an analogous semantic operator
432 **Rec**. It is defined by primitive recursion on the index argument, returning the first argument
433 in the base case and calling itself recursively in the step case. Note that the definition of
434 **Rec** is specific to the index type it recurses over. We only use the index language of natural
435 numbers here, so the appropriate set of index values is $[\![\mathsf{nat}]\!] = \mathbb{N}$.
436 Last, we give the interpretation for typing contexts Γ, describing well-formed term-level
437 environments σ.

▶ **Definition 18** (Interpretation of typing contexts).

$$[\![\cdot]\!](\theta; \eta) = \{\cdot\} \quad (438)$$

$$[\![\Gamma, x{:}T]\!](\theta; \eta) = \{\sigma, v/x \mid \sigma \in [\![\Gamma]\!](\theta; \eta), v \in [\![T]\!](\theta; \eta)\} \quad (439, 440)$$

441 4.4 Proof

442 We now sketch our proof using some key lemmas. The following two lemmas concern the
443 fixed point operators $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$, and are key for reasoning about (co)recursive types and
444 Mendler-style (co)recursion. These lemmas generalize those of Jacob-Rao et al. [2016] from
445 the simply typed setting.

446 ▶ **Lemma 19** (Soundness of pre-fixed point). *Suppose $\mathcal{L}$ is a complete lattice, $\mathcal{F} : \mathcal{L} \to \mathcal{L}$ and*
447 *$\boldsymbol{\mu}$ is as in Def. 14. Then $\mathcal{F}(\boldsymbol{\mu}\mathcal{F}) \leq \boldsymbol{\mu}\mathcal{F}$.*

448 ▶ **Lemma 20** (Function space from pre-fixed and post-fixed points). *Let $\mathcal{L} = \vec{\mathcal{U}} \to \mathcal{P}(\Omega)$ and*
449 *$\mathcal{B} \in \mathcal{L}$ and $\mathcal{F} : \mathcal{L} \to \mathcal{L}$.*

450 1. *If $\forall \mathcal{X} \in \mathcal{L}.\, c \in \vec{\mathcal{U}}, \mathcal{X} \to \mathcal{B} \implies c \in \vec{\mathcal{U}}, \mathcal{F}\mathcal{X} \to \mathcal{B}$, then $c \in \vec{\mathcal{U}}, \boldsymbol{\mu}\mathcal{F} \to \mathcal{B}$.*
451 2. *If $\forall \mathcal{X} \in \mathcal{L}.\, c \in \vec{\mathcal{U}}, \mathcal{B} \to \mathcal{X} \implies c \in \vec{\mathcal{U}}, \mathcal{B} \to \mathcal{F}\mathcal{X}$, then $c \in \vec{\mathcal{U}}, \mathcal{B} \to \boldsymbol{\nu}\mathcal{F}$.*

452 Another key result we rely on is that type-level substitutions associate with our semantic
453 interpretations. Note that single index (and spine) substitutions on types are handled as
454 special cases of the result for simultaneous index substitutions. We omit the definitions of
455 type substitutions for brevity.

456 ▶ **Lemma 21** (Type-level substitution associates with interpretation).
457 *Suppose $\Delta; \Xi \vdash T \Leftarrow K$ or $\Delta; \Xi \vdash T \Rightarrow K$, and $\vdash \theta : \Delta'$ and $\eta \in [\![\Xi']\!](\theta)$.*

458 1. *If $\Delta' \vdash \Theta : \Delta$ and $\Xi' = \Xi[\Theta]$ then $[\![\Xi']\!](\theta) = [\![\Xi]\!](\Theta[\theta])$ and $[\![T[\Theta]]\!](\theta; \eta) = [\![T]\!](\Theta[\theta]; \eta)$.*
459 2. *If $\Delta = \Delta'$ and $\Xi = \Xi', X{:}K$ and $\Delta'; \Xi' \vdash S \Leftarrow K$ or $\Delta'; \Xi' \vdash S \Rightarrow K$, then*
460 *$[\![T[S/X]]\!](\theta; \eta) = [\![T]\!](\theta; \eta, [\![S]\!](\theta; \eta)/X)$.*

461 **Proof.** By induction on the structure of T. ◀

462 The next two lemmas concern recursive types and terms respectively.

463 ▶ **Lemma 22** (Recursive type contains unfolding).
464 *Let $R = \mu X{:}K.\, \Lambda\, \vec{u}.\, S$ where $K = \Pi \overrightarrow{u{:}U}.\, *$ and $\Delta; \Xi \vdash R \Rightarrow K$, and $\Delta \vdash \vec{M} : (\overrightarrow{u{:}U})$ and*
465 *$\vdash \theta : \Delta$ and $\eta \in [\![\Xi]\!](\theta)$. Then $\boldsymbol{in}_{\mu}\, [\![S[\overrightarrow{M/u}; R/X]]\!](\theta; \eta) \subseteq [\![R\, \vec{M}]\!](\theta; \eta)$.*

▶ **Lemma 23** (Backward closure).
Let t be a term, θ and σ environments, and $\mathcal{A}, \mathcal{B} \in \vec{\mathcal{U}} \to \mathcal{P}(\Omega)$.

1. *If $t[\theta; \sigma, (\texttt{rec}\, f. t)[\theta; \sigma]/f] \Downarrow c' \in \vec{\mathcal{U}}, \mathcal{A} \to \mathcal{B}$, then $(\texttt{rec}\, f. t)[\theta; \sigma] \in \vec{\mathcal{U}}, in^*_\mu\, \mathcal{A} \to \mathcal{B}$.*
2. *If $t[\theta; \sigma, (\texttt{corec}\, f. t)[\theta; \sigma]/f] \Downarrow c' \in \vec{\mathcal{U}}, \mathcal{A} \to \mathcal{B}$, then $(\texttt{corec}\, f. t)[\theta; \sigma] \in \vec{\mathcal{U}}, \mathcal{A} \to out^*_\nu\, \mathcal{B}$.*

Our final lemma concerns the semantic equivalence of an applied stratified type with its unfolding. Note that here we only state and prove the lemma for an index language of natural numbers. For a different index language, one would need to reverify this lemma for the corresponding stratified type. This should be straight-forward once the semantic **Rec** operator is chosen to reflect the inductive structure of the index language.

▶ **Lemma 24** (Stratified types equivalent to unfolding).
Let $T_{\mathsf{Rec}} \equiv \mathsf{Rec}_K\, (0 \mapsto T_z \mid \mathsf{suc}\, n,\, X \mapsto T_s)$ where $K = \Pi n{:}\, \mathsf{nat}\, .\, \Pi\overrightarrow{u{:}U}. *$ and $\Delta; \Xi \vdash T_{\mathsf{Rec}} \Rightarrow K$, and $\Delta \vdash \vec{M} : (\overrightarrow{u{:}U})$ and $\Delta \vdash N : \mathsf{nat}$ and $\vdash \theta : \Delta$ and $\eta \in [\![\Xi]\!](\theta)$. Then

1. *$[\![T_{\mathsf{Rec}}\, 0\, \vec{M}]\!](\theta; \eta) = in_0\, ([\![T_z\, \vec{M}]\!](\theta; \eta))$ and*
2. *$[\![T_{\mathsf{Rec}}\, (\mathsf{suc}\, N)\, \vec{M}]\!](\theta; \eta) = in_{\mathsf{suc}}\, ([\![T_s[N/n; (T_{\mathsf{Rec}}\, N)/X]\, \vec{M}]\!](\theta; \eta))$.*

Finally we state the main termination theorem.

▶ **Theorem 25** (Termination of evaluation). *If $\Delta; \Xi; \Gamma \vdash t \Leftarrow T$ or $\Delta; \Xi; \Gamma \vdash t \Rightarrow T$, and $\vdash \theta : \Delta$ and $\eta \in [\![\Xi]\!](\theta)$ and $\sigma \in [\![\Gamma]\!](\theta; \eta)$, then $t[\theta; \sigma] \Downarrow v$ for some $v \in [\![T]\!](\theta; \eta)$.*

5 Related Work

Our work draws inspiration from two different areas: dependent type theory and proof theory.

Dependent type theories often support large eliminations that are definitions of dependent types by primitive recursion. For example, a large elimination on a natural number is of the form $\texttt{Rec}\ t\ T_0\ (X. T_{\mathsf{suc}})$, similar to our stratified type. In a dependent type theory, this large elimination reduces in the same way as term-level recursion, depending on the value of the natural number expression t.They reduce the same way as term-level recursion. Large eliminations are important for increasing the expressive power of dependent type theories, in particular allowing one to prove that constructors of inductive types are disjoint (e.g. $0 \neq 1$). Jan Smith [Smith, 1989] gave an account of large eliminations (calling them *propositional functions*) as an extension of Martin-Löf type theory. Werner [1992] was able to prove strong normalization for a language (dependently typed System F) with large eliminations over natural numbers. Werner's proof needs to consider the normalizability of the natural number argument to the large elimination. Our interpretation of stratified types is simplified by the fact that the natural number argument comes from an index language containing only normal forms. In general, our work shows how to gain the power of large eliminations in an indexed type system by simulating reduction on the level of types by unfolding stratified types in their typing rules.

In the world of proof theory, our core language corresponds to a first-order logic with equality, inductive and stratified (recursive) definitions. Momigliano and Tiu [2004], Tiu and Momigliano [2012] give comprehensive treatments of logics with induction and co-induction as well as first-class equality. They present their logics in a sequent calculus style and prove cut elimination (i.e. that the cut rule is admissible) which implies consistency of the logics. Their cut elimination proof extends Girard's proof technique of reducibility candidates, similar to ours. Note that they require strict positivity of inductive definitions, i.e. the head of a definition (analogous to the recursive type variable) is not allowed to occur to the left of an implication.

510 Tiu [2012] also develops a first-order logic with stratified definitions similar to our
511 stratified types. His notion of stratification comes from defining the "level" of a formula,
512 which measures its size. A recursive definition is then called stratified if the level does not
513 increase from the head of the definition to the body. This is a more general formulation than
514 our notion of stratification for types: we require the type to be stratified exactly according
515 to the structure of an index term, instead of a more general decreasing measure. However,
516 we could potentially replicate such a measure by suitably extending our index language.

517 Another approach to supporting recursive definitions in proof theory is via a rewriting
518 relation, as in the Deduction Modulo system [Dowek et al., 2003]. The idea of this system is
519 to generalize a given first-order logic to account for a congruence relation defined by a set
520 of rewrite rules. This rewriting could include recursive definitions in the same sense as Tiu.
521 Dowek and Werner [2003] show that such logics under congruences can be proven normalizing
522 given general conditions on the congruence. Specifically, they define the notion of a *pre-model*
523 of a congruence, whose existence is sufficient to prove cut elimination of the logic. Baelde and
524 Nadathur [2012] extend this work in the following way. First, they present a first-order logic
525 with inductive and co-inductive definitions, together with a general form of equality. They
526 show strong normalization for this logic using a reducibility candidate argument. Crucially,
527 their proof is in terms of a pre-model which anticipates the addition of recursive definitions
528 via a rewrite relation. Then they give conditions on the rewrite rules, essentially requiring
529 that each definition follows a well-founded order on its arguments. Under these conditions,
530 they are able to construct a pre-model for the relation, proving normalization as a result.
531 Although their notion of stratification of recursive definitions is slightly more general than
532 ours, our treatment is perhaps more direct as the rewriting of types takes place within our
533 typing rules, and our semantic model accounts for stratified types directly.

534 From a programmer's view, the proof theoretic foundations give rise to programs writ-
535 ten using iterators; our use of Mendler-style (co)recursion is arguably closer to standard
536 programming practice. Mendler-style recursion schemes for term-indexed languages have
537 been investigated by Ahn [2014]. Ahn describes an extension of System F_ω with erasable
538 term indices, called F_i. He combines this with fixed points of type operators, as in the Fix_ω
539 language by Abel and Matthes [2004], to produce the core language Fix_i. In Fix_i, one can
540 embed Mendler-style recursion over term-indexed data types by Church-style encodings.

541 Fundamentally, our use of indices is more liberal than in Ahn's core languages. In
542 F_i, term indices are drawn from the same term language as programs. They are treated
543 polymorphically, in analogue with polymorphic type indices, i.e. they must have closed types
544 and cannot be analyzed at runtime. Our approach is to separate the language of index terms
545 from the language of programs. In TORES, the indices that appear in types can be handled
546 and analyzed at runtime, may be dependently typed and have types with free variables. This
547 flexibility is crucial for writing inductive proofs over LF specifications as we do in Beluga.

548 Another difference from Ahn's work is our treatment of Mendler-style recursion. Ahn is
549 able to embed a variety of Mendler-style recursion schemes via Church encodings, taking
550 advantage of polymorphism and type-level functions inherited from System F_ω. Our work
551 does not include polymorphism and general type-level functions as we concentrate on a small
552 core language for inductive reasoning. For this purpose, TORES includes recursive types with
553 a Mendler-style elimination form. We believe this treatment is a more direct interpretation
554 of Mendler recursion for indexed recursive types.

6 Conclusion and Future Work

We presented a core language Tores extending an indexed type system with (co)recursive types and stratified types. We argued that Tores provides a sound and powerful foundation for programming (co)inductive proofs, in particular those involving logical relations. This power comes from the (co)induction principles on recursive types given by Mendler-style (co)recursion as well as the flexibility of recursive definitions given by stratified types. Type checking in Tores is decidable and types are preserved during evaluation. The soundness of our language is guaranteed by our logical predicate semantics and termination proof. We believe that Tores balances well the proof-theoretic power with a simple metatheory (especially when compared with full dependent types).

An important question to investigate in the future is how to compile a practical language that supports (co)pattern matching into the core language we propose in Tores. Such issues are important to solve in order to create a productive user experience for dependently typed programming and proving.

It would also be interesting to explore how our treatment of indexed recursive and stratified types could help (or hinder) proof search. Proof search is a fundamental technique to ease the development and maintenance of proofs, by automatically generating parts of proof terms. Like Baelde and Nadathur [2012], we are curious to see how our treatment of recursive definitions can be handled by search techniques, especially those derived from focusing [Andreoli, 1992].

References

Andreas Abel and Ralph Matthes. Fixed points of type constructors and primitive recursion. In *18th Int'l Workshop on Computer Science Logic (CSL'04)*, Lecture Notes in Computer Science (LNCS 3210), pages 190–204. Springer, 2004.

Ki Yung Ahn. *The Nax Language: Unifying Functional Programming and Logical Reasoning in a Language based on Mendler-style Recursion Schemes and Term-indexed Types.* PhD thesis, Portland State University, 2014.

Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

David Baelde and Gopalan Nadathur. Combining deduction modulo and logics of fixed-point definitions. In *27th Annual IEEE Symp. on Logic in Computer Science (LICS'12)*, pages 105–114. IEEE, 2012.

Andrew Cave and Brigitte Pientka. Programming with binders and indexed data-types. In *39th ACM Symp. on Principles of Programming Languages (POPL'12)*, pages 413–424. ACM, 2012.

Andrew Cave and Brigitte Pientka. First-class substitutions in contextual type theory. In *8th ACM Int'l Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP'13)*, pages 15–24. ACM, 2013.

Andrew Cave and Brigitte Pientka. A case study on logical relations using contextual types. In *10th Int'l Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP'15)*, pages 18–33. Electronic Proceedings in Theoretical Computer Science (EPTCS), 2015.

James Cheney and Ralf Hinze. First-class phantom types. Technical Report CUCIS TR2003-1901, Cornell University, 2003.

Gilles Dowek and Benjamin Werner. Proof normalization modulo. *J. Symb. Log.*, 68(4): 1289–1316, 2003.

Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem proving modulo. *Journal of Automated Reasoning*, 31(1):33–72, 2003.

J. Y Girard. *Interprétation fonctionnelle et elimination des coupures de l'arithmétique d'ordre supérieur.* These d'état, Université de Paris 7, 1972.

Rohan Jacob-Rao, Andrew Cave, and Brigitte Pientka. Mechanizing proofs about mendler-style recursion. In *11th Int'l Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP'16)*, pages 1 – 9. ACM, 2016.

Raymond C. McDowell and Dale A. Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Transactions on Computational Logic*, 3(1):80–136, 2002.

Nax Paul Francis Mendler. *Inductive Definition in Type Theory.* PhD thesis, Cornell University, Ithaca, NY, USA, 1988. AAI8804634.

Alberto Momigliano and Alwen Fernanto Tiu. Induction and co-induction in sequent calculus. In *Types for Proofs and Programs (TYPES'03)*, Lecture Notes in Computer Science (LNCS 3085), pages 293–308. Springer, 2004.

Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *ACM Transactions on Computational Logic*, 9(3):1–49, 2008.

Brigitte Pientka. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *35th ACM Symp. on Principles of Programming Languages (POPL'08)*, pages 371–382. ACM, 2008.

Brigitte Pientka and Andreas Abel. Structural recursion over contextual objects. In *13th Int'l Conf. on Typed Lambda Calculi and Applications (TLCA'15)*, pages 273–287. Leibniz Int'l Proceedings in Informatics (LIPIcs) of Schloss Dagstuhl, 2015.

Brigitte Pientka and Joshua Dunfield. Programming with proofs and explicit contexts. In *35th ACM Symp. on Principles and Practice of Declarative Programming (PPDP'08)*, pages 163–173. ACM, 2008.

Peter Schroeder-Heister. Rules of definitional reflection. In *8th Annual Symp. on Logic in Computer Science (LICS '93)*, pages 222–232. IEEE Computer Society, 1993.

Jan M. Smith. Propositional functions and families of types. In *Workshop on Programming Logic*, pages 140–159, 1989.

William Tait. Intensional Interpretations of Functionals of Finite Type I. *J. Symb. Log.*, 32 (2):198–212, 1967.

David Thibodeau, Andrew Cave, and Brigitte Pientka. Indexed codata. In *21st ACM Int'l Conf. on Functional Programming (ICFP'16)*, pages 351–363. ACM, 2016.

Alwen Tiu. Stratification in logics of definitions. In *6th Int'l Joint Conf. on Automated Reasoning (IJCAR'12)*, Lecture Notes in Computer Science (LNCS 7364), pages 544–558. Springer, 2012.

Alwen Tiu and Alberto Momigliano. Cut elimination for a logic with induction and co-induction. *J. Applied Logic*, 10(4):330–367, 2012.

Benjamin Werner. A normalization proof for an impredicative type system with large elimination over integers. In *Int'l Workshop on Types for Proofs and Programs (TYPES)*, pages 341–357, 1992.

Hongwei Xi, Chiyan Chen, and Gang Chen. Guarded recursive datatype constructors. In *30th ACM Symp. on Principles of Programming Languages (POPL'03)*, pages 224–235. ACM, 2003.

645 # A Appendix

646 ## A.1 Kinding

647 Our kinding rules are shown in Fig. 5. We employ a bidirectional kinding system to make it
648 evident when kinds can be inferred and when kinding annotations are necessary. Kinding
649 depends on two contexts: the index context Δ, since index variables may appear in types
650 and kinds, and the type variable context Ξ. Note that in general the kinds assigned to
651 type variables in Ξ may depend on the index variables in Δ. The checking judgment
652 $\Delta; \Xi \vdash T \Leftarrow K$ takes all expressions as inputs and verifies that the type is well-kinded. The
653 inference judgment $\Delta; \Xi \vdash T \Rightarrow K$ takes the contexts and type as input to produce a kind as
654 output.

655 In our rules, the kind $*$ of types is always checked, and may rely on inference via the
656 conversion rule. In addition, type-level lambdas $\Lambda\, u. T$ are checked against a kind $\Pi u{:}U. K$
657 by checking the body T under an extended index context $\Delta, u{:}U$. On the other hand, kinds
658 are inferred (synthesized) for type variables by looking them up in the context Ξ, as well as
659 for type-level applications, recursive types and stratified types. Subtly, the kinding for type
660 applications $T\, M$ requires that T synthesize a kind: in particular T cannot be a type-level
661 lambda, which would be checked against a kind. This means that types in $*$ do not arise
662 from reducible lambda applications: lambdas must occur within recursive or stratified types.
663 Finally, recursive and stratified types synthesize the kinds in their annotations. Recursive
664 type variables are added to the context Ξ for checking the body of the type. Stratified types
665 require checking the constituent types using the index information gleaned in each branch:
666 T_0 is checked against $K'[0/u]$ and T_s against $K'[\mathsf{suc}\, u/u]$.

$\boxed{\Delta; \Xi \vdash T \Leftarrow K}$ Check type T against kind K

$$\frac{}{\Delta; \Xi \vdash 1 \Leftarrow *} \qquad \frac{\Delta; \Xi \vdash T_1 \Leftarrow * \quad \Delta; \Xi \vdash T_2 \Leftarrow *}{\Delta; \Xi \vdash T_1 \times T_2 \Leftarrow *} \qquad \frac{\Delta; \Xi \vdash T_1 \Leftarrow * \quad \Delta; \Xi \vdash T_2 \Leftarrow *}{\Delta; \Xi \vdash T_1 + T_2 \Leftarrow *}$$

$$\frac{\Delta \vdash (\overrightarrow{u{:}U})\ \mathsf{itype} \quad \Delta, \overrightarrow{u{:}U}; \Xi \vdash S \Leftarrow * \quad \Delta, \overrightarrow{u{:}U}; \Xi \vdash T \Leftarrow *}{\Delta; \Xi \vdash (\overrightarrow{u{:}U});\ S \to T \Leftarrow *} \qquad \frac{\Delta \vdash U\ \mathsf{itype} \quad \Delta, u{:}U; \Xi \vdash T \Leftarrow *}{\Delta; \Xi \vdash \Sigma u{:}U. T \Leftarrow *}$$

$$\frac{\Delta \vdash M : U \quad \Delta \vdash N : U}{\Delta; \Xi \vdash M = N \Leftarrow *} \qquad \frac{\Delta, u{:}U; \Xi \vdash T \Leftarrow K}{\Delta; \Xi \vdash \Lambda\, u. T \Leftarrow \Pi u{:}U. K} \qquad \frac{\Delta; \Xi \vdash T \Rightarrow *}{\Delta; \Xi \vdash T \Leftarrow *}$$

$\boxed{\Delta; \Xi \vdash T \Rightarrow K}$ Infer a kind K for type T

$$\frac{X{:}K \in \Xi}{\Delta; \Xi \vdash X \Rightarrow K} \qquad \frac{\Delta; \Xi \vdash T \Rightarrow \Pi u{:}U. K \quad \Delta \vdash M : U}{\Delta; \Xi \vdash T\, M \Rightarrow K[M/u]}$$

$$\frac{\Delta; \Xi, X{:}K \vdash T \Leftarrow K}{\Delta; \Xi \vdash \mu X{:}K. T \Rightarrow K} \qquad \frac{\Delta; \Xi, X{:}K \vdash T \Leftarrow K}{\Delta; \Xi \vdash \nu X{:}K. T \Rightarrow K}$$

$$\frac{K = \Pi u{:}\,\mathsf{nat}\,. K' \quad \Delta; \Xi \vdash T_0 \Leftarrow K'[0/u] \quad \Delta, u{:}\,\mathsf{nat}; \Xi, X{:}K' \vdash T_s \Leftarrow K'[\mathsf{suc}\, u/u]}{\Delta; \Xi \vdash \mathsf{Rec}_K(0 \mapsto T_0 \mid \mathsf{suc}\, u,\ X \mapsto T_s) \Rightarrow K}$$

Figure 5 Kinding rules for TORES

A.2 Value Typing

Values are the results of evaluation. Note that values are closed, and hence their typing judgment does not require a context. However, closures do contain terms (typed with the main typing judgment) and environments (typed against the contexts Δ and Γ).

$\boxed{v : T}$ Value v has type T

$$\frac{\cdot \vdash \theta : \Delta \quad \sigma : \Gamma[\theta] \quad \Delta; \cdot; \Gamma \vdash g \Leftarrow T}{(g)[\theta; \sigma] : T[\theta]} \qquad \frac{}{\langle\rangle : 1} \qquad \frac{\cdot \vdash M = N}{\texttt{refl} : M = N} \qquad \frac{v_1 : T_1 \quad v_2 : T_2}{\langle v_1, v_2 \rangle : T_1 \times T_2}$$

$$\frac{v : T_i}{\texttt{in}_i\, v : T_1 + T_2} \qquad \frac{\cdot \vdash M : U \quad v : T[M/u]}{\texttt{pack}\,(M, v) : \Sigma u{:}U.\, T} \qquad \frac{v : T[\overrightarrow{M/u}; \mu X{:}K.\, \Lambda\, \vec{u}.\, T/X]}{\texttt{in}_\mu\, v : (\mu X{:}K.\, \Lambda\, \vec{u}.\, T)\, \vec{M}}$$

$$\frac{v : T_0\, \vec{M}}{\texttt{in}_0\, v : T_{\texttt{Rec}}\, 0\, \vec{M}} \qquad \frac{v : T_s[N/u; T_{\texttt{Rec}}\, N/X]\, \vec{M}}{\texttt{in}_{\texttt{suc}}\, v : T_{\texttt{Rec}}\, (\texttt{suc}\, N)\vec{M}}$$

$$\frac{\cdot \vdash \theta : \Delta \quad \sigma : \Gamma[\theta] \quad \cdot \vdash \vec{N} : \vec{U} \quad v : S[\theta, \overrightarrow{N/u}] \qquad \Delta; \cdot, X{:}K; \Gamma, f{:}((\overrightarrow{u{:}U});\, S \to X\, \vec{u}) \vdash t \Leftarrow (\overrightarrow{u{:}U});\, S \to T[\overrightarrow{u/u'}]}{(\texttt{corec}\, f.\, t)[\theta; \sigma] \cdot \vec{N}\ v : (\nu X{:}K.\, \Lambda\, \vec{u'}.\, T)[\theta]\ \vec{N}}$$

$\boxed{\sigma : \Gamma}$ Environment σ has domain Γ

$$\frac{}{\cdot : \cdot} \qquad \frac{\sigma : \Gamma \qquad v : T}{(\sigma, v/x) : \Gamma, x{:}T}$$

Figure 6 Value and environment typing

A.3 Proofs

▶ **Theorem 26** (Subject Reduction).

1. *If $t[\theta; \sigma] \Downarrow v$ where $\Delta; \cdot; \Gamma \vdash t \Leftarrow T$ or $\Delta; \cdot; \Gamma \vdash t \Rightarrow T$, and $\vdash \theta : \Delta$ and $\sigma : \Gamma[\theta]$, then $v : T[\theta]$.*
2. *If $g[\theta; \sigma] \cdot \vec{N}\, v \Downarrow w$ where $\Delta; \cdot; \Gamma \vdash g \Leftarrow (\overrightarrow{u{:}U});\, S \to T$ and $\vdash \theta : \Delta$ and $\sigma : \Gamma[\theta]$ and $\vdash \vec{N} : (\overrightarrow{u{:}U})[\theta]$ and $v : S[\theta, \overrightarrow{N/u}]$, then $w : T[\theta, \overrightarrow{N/u}]$.*
3. *If $c \cdot_{\mathit{out}_\nu} \Downarrow w$ where $c : (\nu X{:}K.\, \Lambda\, \vec{u}.\, T)\ \vec{M}$ then $w : T[\overrightarrow{M/u}; (\nu X{:}K.\, \Lambda\, \vec{u}.\, T)/X]$.*

Proof. By mutual induction on the evaluation judgments. We include a few key cases here.

Case $$\frac{s[\theta; \sigma] \Downarrow \texttt{refl} \qquad \Delta' \vdash \Theta \doteq \theta \searrow (\cdot \mid \theta') \qquad t[\theta'; \sigma] \Downarrow v}{(\texttt{eq}\, s\, \texttt{with}\, (\Delta'.\Theta \ \mapsto\ t)\,)[\theta; \sigma] \Downarrow v}$$

$\Delta; \cdot; \Gamma \vdash s \Rightarrow M = N$ and $\Delta'; \cdot; \Gamma[\Theta] \vdash t \Leftarrow T[\Theta]$ — by inversion of typing
$\texttt{refl} : (M = N)[\theta]$ — by I.H.
$\texttt{refl} : M[\theta] = N[\theta]$ — by type substitution
$\vdash M[\theta] = N[\theta]$ — by inversion of value typing
$\vdash \theta' : \Delta'$ and $\vdash \Theta[\theta'] = \theta$ — by soundness of matching (Req. 5.2)

686 $T[\Theta][\theta'] = T[\Theta[\theta']] = T[\theta]$ by associativity of type substitution
687 $\Gamma[\Theta][\theta'] = \Gamma[\Theta[\theta']] = \Gamma[\theta]$ similarly for contexts
688 $\sigma : \Gamma[\theta]$ by assumption
689 $\sigma : \Gamma[\Theta][\theta']$ by context equality
690 $v : T[\Theta][\theta']$ by I.H.
691 $v : T[\theta]$ by type equality

692 **Case** $$\frac{t[\theta;\sigma] \Downarrow c \qquad s[\theta;\sigma] \Downarrow v \qquad c \cdot \overrightarrow{M[\theta]}\, v \Downarrow w}{(t\, \vec{M}\, s)[\theta;\sigma] \Downarrow w}$$

693 $\vdash \theta : \Delta$ and $\sigma : \Gamma[\theta]$ by assumption
694 $\Delta; \cdot; \Gamma \vdash t\, \vec{M}\, s \Rightarrow T[\overrightarrow{M/u}]$ by assumption
695 $\Delta; \cdot; \Gamma \vdash t \Rightarrow (\overrightarrow{u{:}U});\, S \to T$
696 $\Delta \vdash \vec{M} : (\overrightarrow{u{:}U})$
697 $\Delta; \cdot; \Gamma \vdash s \Leftarrow S[\overrightarrow{M/u}]$ by inversion of typing
698 $\vdash \overrightarrow{M[\theta]} : (\overrightarrow{u{:}U})[\theta]$ extending Req. 3.2 to index spines
699 $c : ((\overrightarrow{u{:}U});\, S \to T)[\theta]$ by I.H.
700 $v : S[\overrightarrow{M/u}][\theta]$ by I.H.
701 $v : S[\theta, \overrightarrow{M[\theta]/u}]$ by associativity of type substitution
702 $c = g[\theta';\sigma']$ where g is a function value by closure grammar and typing
703 $\Delta'; \cdot; \Gamma' \vdash g \Leftarrow G$ and $g[\theta';\sigma'] : G[\theta']$ by closure typing
704 $G[\theta'] = ((\overrightarrow{u{:}U});\, S \to T)[\theta]$ by previous lines
705 $G = (\overrightarrow{u{:}U'});\, S' \to T'$ where $\overrightarrow{U'[\theta']} = \overrightarrow{U[\theta]}$
706 and $S'[\theta', \overrightarrow{u/u}] = S[\theta, \overrightarrow{u/u}]$ and $T'[\theta', \overrightarrow{u/u}] = T[\theta, \overrightarrow{u/u}]$ by equality of types
707 $v : S'[\theta', \overrightarrow{M[\theta]/u}]$ by type equality
708 $\vdash \overrightarrow{M[\theta]} : (\overrightarrow{u{:}U'[\theta']})$ by type equality
709 $w : T'[\theta', \overrightarrow{M[\theta]/u}]$ by I.H. 2
710 $w : T[\theta, \overrightarrow{M[\theta]/u}]$ by type equality

711 **Case** $$\frac{t[\theta, \overrightarrow{N/u}; \sigma, v/x] \Downarrow w}{(\lambda\, \vec{u}, x.\, t)[\theta;\sigma] \cdot \vec{N}\, v \Downarrow w}$$

712 $\Delta; \cdot; \Gamma \vdash \lambda\, \vec{u}, x.\, t \Leftarrow (\overrightarrow{u{:}U});\, S \to T$ by assumption
713 $\Delta, \overrightarrow{u{:}U}; \cdot; \Gamma, x{:}S \vdash t \Leftarrow T$ by inversion of typing
714 $\vdash \theta : \Delta$ by assumption
715 $\vdash \theta, \overrightarrow{N/u} : \Delta, \overrightarrow{u{:}U}$ by substitution typing
716 $\sigma : \Gamma[\theta]$ by assumption
717 $\sigma : \Gamma[\theta, \overrightarrow{N/u}]$ by weakening since $\vec{u}$ do not occur in Γ
718 $\sigma, v/x : \Gamma[\theta, \overrightarrow{N/u}], x : S[\theta, \overrightarrow{N/u}]$ by environment typing
719 $\sigma, v/x : (\Gamma, x : S)[\theta, \overrightarrow{N/u}]$ by def. of context substitution
720 $w : T[\theta, \overrightarrow{N/u}]$ by I.H.

721 **Case** $$\frac{t[\theta; \sigma, (\texttt{rec}\, f.\, t)[\theta;\sigma]/f] \Downarrow c \qquad c \cdot \vec{N}\, v' \Downarrow w}{(\texttt{rec}\, f.\, t)[\theta;\sigma] \cdot \vec{N}\, (\texttt{in}_\mu\, v') \Downarrow w}$$

722 $\Delta; \cdot; \Gamma \vdash \mathtt{rec}\, f.\, t \Leftarrow (\overrightarrow{u{:}U});\ (\mu X{:}K.\, \Lambda\, \vec{v}.\, S)\, \vec{u} \to T$ by assumption
723 $\Delta; X{:}K; \Gamma, f{:}((\overrightarrow{u{:}U});\ X\vec{u} \to T) \vdash t \Leftarrow (\overrightarrow{u{:}U});\ S[\overrightarrow{u/v}] \to T$ by inversion of typing
724 $\Delta; \cdot; \Gamma, f{:}((\overrightarrow{u{:}U});\ (\mu X{:}K.\, \Lambda\, \vec{v}.\, S)\ \vec{u} \to T) \vdash t \Leftarrow (\overrightarrow{u{:}U});\ S[\overrightarrow{u/v}; \mu X{:}K.\, \Lambda\, \vec{v}.\, S/X] \to T$
725 by substitution property of type variables
726 $\mathtt{in}_\mu\, v' : ((\mu X{:}K.\, \Lambda\, \vec{v}.\, S)\ \vec{u}))[\theta, \overrightarrow{N/u}]$ by assumption
727 $v' : S[\theta, \overrightarrow{N/u};\ \mu X{:}K.\, \Lambda\, \vec{v}.\, S/X]$ by value typing
728 $\vdash \theta : \Delta$ by assumption
729 $\sigma, (\mathtt{rec}\, f.\, t)[\theta; \sigma]/f : \Gamma[\theta], f{:}((\overrightarrow{u{:}U});\ (\mu X{:}K.\, \Lambda\, \vec{v}.\, S)\ \vec{u} \to T)[\theta]$ by environment typing
730 $\sigma, (\mathtt{rec}\, f.\, t)[\theta; \sigma]/f : (\Gamma, f{:}((\overrightarrow{u{:}U});\ (\mu X{:}K.\, \Lambda\, \vec{v}.\, S)\ \vec{u} \to T))[\theta]$ by def. of context substitution
731 $c : ((\overrightarrow{u{:}U});\ S[\overrightarrow{u/v}; \mu X{:}K.\, \Lambda\, \vec{v}.\, S/X] \to T)[\theta]$ by I.H.
732 $c = g[\theta'; \sigma']$ where $\Delta'; \cdot; \Gamma' \vdash g \Leftarrow G$ and $g[\theta'; \sigma'] : G[\theta']$
733 and $\vdash \theta' : \Delta'$ and $\sigma' : \Gamma'[\theta']$ by closure grammar and typing
734 $G[\theta'] = ((\overrightarrow{u{:}U});\ S \to T)[\theta]$ by previous lines
735 $G = (\overrightarrow{u{:}U'});\ S' \to T'$ where $\overrightarrow{U'[\theta']} = \overrightarrow{U[\theta]}$
736 and $S'[\theta', \overrightarrow{u/u}] = S[\theta, \overrightarrow{u/u}\ ;\ \mu X{:}K.\, \Lambda\, \vec{v}.\, S/X]$ and $T'[\theta', \overrightarrow{u/u}] = T[\theta, \overrightarrow{u/u}]$ by type equality
737 $\Delta'; \cdot; \Gamma' \vdash g \Leftarrow (\overrightarrow{u{:}U'});\ S' \to T'$ by previous lines
738 $\vdash \vec{N} : (\overrightarrow{u : U[\theta]})$ by assumption
739 $\vdash \vec{N} : (\overrightarrow{u : U'[\theta']})$ by type equality
740 $v' : S'[\theta', \overrightarrow{N/u}]$ by type equality
741 $w : T'[\theta', \overrightarrow{N/u}]$ by I.H.
742 $w : T[\theta, \overrightarrow{N/u}]$ by type equality
743

744 **Case** $\dfrac{t[\theta; \sigma, (\mathtt{corec}\, f.\, t)[\theta; \sigma]/f] \Downarrow c \quad c \cdot \vec{N}\, v \Downarrow w}{((\mathtt{corec}\, f.\, t)[\theta; \sigma] \cdot \vec{N}\ v)\ \cdot_{\mathtt{out}_\nu} \Downarrow w}$

745 $(\mathtt{corec}\, f.\, t)[\theta; \sigma] \cdot \vec{N}\ v : (\nu X{:}K.\, \Lambda\, \vec{u'}.\, T)\, \vec{M}$ by assumption
746 $(\mathtt{corec}\, f.\, t)[\theta; \sigma] \cdot \vec{N}\ v : (\nu X{:}K.\, \Lambda\, \vec{u'}.\, T')[\theta]\ \vec{N}$ by value typing
747 $(\nu X{:}K.\, \Lambda\, \vec{u'}.\, T) = (\nu X{:}K.\, \Lambda\, \vec{u'}.\, T')[\theta]$ and $\vec{N} = \vec{M}$ by type equality
748 $\Delta; \cdot, X{:}K; \Gamma, f{:}((\overrightarrow{u{:}U});\ S \to X\vec{u}) \vdash t \Leftarrow (\overrightarrow{u{:}U});\ S \to T'[\overrightarrow{u/u'}]$
749 and $\cdot \vdash \theta : \Delta$
750 and $\sigma : \Gamma[\theta]$
751 and $\cdot \vdash \vec{N} : \vec{U}$
752 and $v : S[\theta, \overrightarrow{N/u}]$ by inversion on value typing
753 $\Delta; \cdot; \Gamma, f{:}((\overrightarrow{u{:}U});\ S \to (\nu X{:}K.\, \Lambda\, \vec{u'}.\, T')\ \vec{u}) \vdash t' \Leftarrow (\overrightarrow{u{:}U});\ S \to T'[\overrightarrow{u/u'}; (\nu X{:}K.\, \Lambda\, \vec{u'}.\, T')/X]$
754 by substitution property of type variables
755 $\sigma, (\mathtt{corec}\, f.\, t)[\theta; \sigma]/f : \Gamma[\theta], f{:}((\overrightarrow{u{:}U});\ S \to (\nu X{:}K.\, \Lambda\, \vec{u'}.\, T')\ \vec{u})[\theta]$ by environment typing
756 $\sigma, (\mathtt{corec}\, f.\, t)[\theta; \sigma]/f : (\Gamma, f{:}((\overrightarrow{u{:}U});\ S \to (\nu X{:}K.\, \Lambda\, \vec{u'}.\, T')\ \vec{u}))[\theta]$ by def. of ctx subst.
757 $c : ((\overrightarrow{u{:}U});\ S \to T'[\overrightarrow{u/u'}; (\nu X{:}K.\, \Lambda\, \vec{u'}.\, T')/X])[\theta]$ by I.H.
758 $c = g[\theta'; \sigma']$ where $\Delta'; \cdot; \Gamma' \vdash g \Leftarrow G$ and $g[\theta'; \sigma'] : G[\theta']$ by closure grammar and typing
759 $G[\theta'] = ((\overrightarrow{u{:}U});\ S \to T'[\overrightarrow{u/u'}; (\nu X{:}K.\, \Lambda\, \vec{u'}.\, T')/X])[\theta]$ by type uniqueness
760 $G = ((\overrightarrow{u{:}U'});\ S' \to T'')$ where $\overrightarrow{U'[\theta']} = \overrightarrow{U[\theta]}$ and $S'[\theta', \overrightarrow{u/u}] = S[\theta, \overrightarrow{u/u}]$
761 and $T''[\theta', \overrightarrow{u/u}] = T'[\theta, \overrightarrow{u/u'}\ ;\ (\nu X{:}K.\, \Lambda\, \vec{u'}.\, T')[\theta]/X]$ by type equality
762 $\Delta'; \cdot; \Gamma' \vdash g \Leftarrow (\overrightarrow{u{:}U'});\ S' \to T''$ by previous lines
763 $\vdash \vec{N} : \overrightarrow{U'[\theta']}$ by type equality

764 $v : S'[\theta', \overrightarrow{N/u}]$ by type equality
765 $w : T''[\theta', \overrightarrow{N/u}]$ by I.H.
766 $w : T'[\theta, \overrightarrow{M/u'} \; ; \; (\nu X{:}K.\, \Lambda\, \overrightarrow{u'}.\, T')[\theta]/X]$ by type equality
767 $w : T[\overrightarrow{M/u'} \; ; \; (\nu X{:}K.\, \Lambda\, \overrightarrow{u'}.\, T)/X]$ by type equality
768 ◀

769 ▶ **Lemma 27** (Soundness of pre-fixed point). *Suppose $\mathcal{L}$ is a complete lattice, $\mathcal{F} : \mathcal{L} \to \mathcal{L}$ and*
770 *$\boldsymbol{\mu}$ is as in Def. 14. Then $\mathcal{F}(\boldsymbol{\mu}\mathcal{F}) \leq \boldsymbol{\mu}\mathcal{F}$.*

771 **Proof.** Recall that $\boldsymbol{\mu}\mathcal{F} = \bigwedge \mathcal{S}$ where $\mathcal{S} = \{\mathcal{C} \in \mathcal{L} \mid \forall \mathcal{X} \in \mathcal{L}.\, \mathcal{X} \leq \mathcal{C} \implies \mathcal{F}(\mathcal{X}) \leq \mathcal{C}\}$. To
772 show $\mathcal{F}(\boldsymbol{\mu}\mathcal{F}) \leq \bigwedge \mathcal{S}$, it suffices to show $\mathcal{F}(\boldsymbol{\mu}\mathcal{F}) \leq \mathcal{C}$ for every $\mathcal{C} \in \mathcal{S}$. By definition of the
773 meet, $\boldsymbol{\mu}\mathcal{F} \leq \mathcal{C}$ for every $\mathcal{C} \in \mathcal{S}$. But by definition of $\mathcal{S}$, this implies that $\mathcal{F}(\boldsymbol{\mu}\mathcal{F}) \leq \mathcal{C}$ as
774 required. (This argument exploits our impredicative definition of $\boldsymbol{\mu}$.) ◀

775 ▶ **Lemma 28** (Function space from pre-fixed and post-fixed points). *Let $\mathcal{L} = \vec{\mathcal{U}} \to \mathcal{P}(\Omega)$ and*
776 *$\mathcal{B} \in \mathcal{L}$ and $\mathcal{F} : \mathcal{L} \to \mathcal{L}$.*

777 **1.** *If $\forall \mathcal{X} \in \mathcal{L}.\, c \in \vec{\mathcal{U}}, \mathcal{X} \to \mathcal{B} \implies c \in \vec{\mathcal{U}}, \mathcal{F}\mathcal{X} \to \mathcal{B}$, then $c \in \vec{\mathcal{U}}, \boldsymbol{\mu}\mathcal{F} \to \mathcal{B}$.*
778 **2.** *If $\forall \mathcal{X} \in \mathcal{L}.\, c \in \vec{\mathcal{U}}, \mathcal{B} \to \mathcal{X} \implies c \in \vec{\mathcal{U}}, \mathcal{B} \to \mathcal{F}\mathcal{X}$, then $c \in \vec{\mathcal{U}}, \mathcal{B} \to \boldsymbol{\nu}\mathcal{F}$.*

779 **Proof.** We will reframe the lemma statement using a new piece of notation. For a closure
780 $c \in \Omega$ and $\mathcal{B} \in \vec{\mathcal{U}} \to \mathcal{P}(\Omega) = \mathcal{L}$, define $\mathcal{E}_c(\mathcal{B}) \in \mathcal{L}$ by $\mathcal{E}_c(\mathcal{B})(\vec{M}) = \{v \in \Omega \mid c \cdot \vec{M}\, v \Downarrow w \in$
781 $\mathcal{B}(\vec{M})\}$. One can see that $c \in \vec{\mathcal{U}}, \mathcal{A} \to \mathcal{B} \iff \mathcal{A} \leq \mathcal{E}_c(\mathcal{B})$ (using the ordering on $\mathcal{L}$). We
782 can now rewrite the lemma as the following:

783 **1.** if $\forall \mathcal{X} \in \mathcal{L}.\, \mathcal{X} \leq \mathcal{E}_c(\mathcal{B}) \implies \mathcal{F}\mathcal{X} \leq \mathcal{E}_c(\mathcal{B})$ then $\boldsymbol{\mu}\mathcal{F} \leq \mathcal{E}_c(\mathcal{B})$;
784 **2.** if $\forall \mathcal{X} \in \mathcal{L}.\, \mathcal{E}_c(\mathcal{B}) \leq \mathcal{X} \implies \mathcal{E}_c(\mathcal{B}) \leq \mathcal{F}\mathcal{X}$ then $\mathcal{E}_c(\mathcal{B}) \leq \boldsymbol{\nu}\mathcal{F}$.

785 To prove each of them, we first assume the premise. Let us do the first statement now.
786 Recall that $\boldsymbol{\mu}\mathcal{F} = \bigwedge \mathcal{S}$ where $\mathcal{S} = \{\mathcal{C} \in \mathcal{L} \mid \forall \mathcal{X} \in \mathcal{L}.\, \mathcal{X} \leq \mathcal{C} \implies \mathcal{F}(\mathcal{X}) \leq \mathcal{C}\}$. By definition
787 of the meet, $\boldsymbol{\mu}\mathcal{F} \leq \mathcal{C}$ for every $\mathcal{C} \in \mathcal{S}$. Therefore it suffices to show that there is just one
788 $\mathcal{C} \in \mathcal{S}$ for which $\mathcal{C} \leq \mathcal{E}_c(\mathcal{B})$. However, our assumption is exactly that $\mathcal{E}_c(\mathcal{B}) \in \mathcal{S}$, and clearly
789 $\mathcal{E}_c(\mathcal{B}) \leq \mathcal{E}_c(\mathcal{B})$, so we are done. (This proof again makes use of impredicativity in the
790 definition of $\boldsymbol{\mu}$.)
791 The second case follows the same idea. $\boldsymbol{\nu}\mathcal{F} = \bigvee \mathcal{S}$ where $\mathcal{S} = \{\mathcal{C} \in \mathcal{L} \mid \forall \mathcal{X} \in \mathcal{L}.\, \mathcal{C} \leq$
792 $\mathcal{X} \implies \mathcal{C} \leq \mathcal{F}(\mathcal{X})\}$. By definition of join, $\mathcal{C} \leq \boldsymbol{\nu}F$ for every $\mathcal{C} \in \mathcal{S}$. It thus suffices to show
793 that there is one $\mathcal{C} \in \mathcal{S}$ for which $\mathcal{E}_c(\mathcal{B}) \leq \mathcal{C}$. Again, $\mathcal{E}_c(\mathcal{B}) \in \mathcal{S}$ and $\mathcal{E}_c(\mathcal{B}) \leq \mathcal{E}_c(\mathcal{B})$ so we are
794 done. ◀

795 ▶ **Lemma 29** (Recursive type contains unfolding).
796 *Let $R = \mu X{:}K.\, \Lambda\, \vec{u}.\, S$ where $K = \Pi\overrightarrow{u{:}U}.\, *$ and $\Delta; \Xi \vdash R \Rightarrow K$, and $\Delta \vdash \vec{M} : (\overrightarrow{u{:}U})$ and*
797 *$\vdash \theta : \Delta$ and $\eta \in [\![\Xi]\!](\theta)$. Then $\mathtt{in}_\mu\, [\![S[\overrightarrow{M/u}; R/X]]\!](\theta; \eta) \subseteq [\![R\, \vec{M}]\!](\theta; \eta)$.*

798 **Proof.** Let $\mathcal{L} = [\![K]\!](\theta)$.
799 Define $\mathcal{F} : \mathcal{L} \to \mathcal{L}$ by $\mathcal{F}(\mathcal{X}) = \vec{N} \mapsto \mathtt{in}_\mu\, [\![S]\!](\theta, \overrightarrow{N/u}; \eta, \mathcal{X}/X)$.
800 Then $[\![R]\!](\theta; \eta)$
801 $= \boldsymbol{\mu}(\mathcal{X} \mapsto \mathtt{in}^*_\mu\, [\![\Lambda\, \vec{u}.\, S]\!](\theta; \eta, \mathcal{X}/X))$ by $[\![\mu X.\, T]\!]$ def.
802 $= \boldsymbol{\mu}(\mathcal{X} \mapsto \vec{N} \mapsto \mathtt{in}_\mu\, [\![S]\!](\theta, \overrightarrow{N/u}; \eta, \mathcal{X}/X))$ by $[\![\Lambda\, \vec{u}.\, T]\!]$ def.
803 $= \boldsymbol{\mu}\mathcal{F}$ by $\mathcal{F}$ def.
804 $\mathcal{F}([\![R]\!](\theta; \eta)) \leq_{\mathcal{L}} [\![R]\!](\theta; \eta)$ by Lemma 27

805 Now $\mathtt{in}_\mu \llbracket S[\overrightarrow{M/u}; R/X]\rrbracket(\theta;\eta)$
806 $= \mathtt{in}_\mu \llbracket S\rrbracket((\mathsf{id}_\Delta, \overrightarrow{M/u})[\theta];\eta, \llbracket R\rrbracket(\theta;\eta)/X)$ by Lemma 21
807 $= \mathtt{in}_\mu \llbracket S\rrbracket(\theta, \overrightarrow{M[\theta]/u};\eta, \llbracket R\rrbracket(\theta;\eta)/X)$ by Def. 2
808 $= \mathcal{F}(\llbracket R\rrbracket(\theta;\eta))(\overrightarrow{M[\theta]})$ by $\mathcal{F}$ def.
809 $\subseteq \llbracket R\rrbracket(\theta;\eta)(\overrightarrow{M[\theta]})$ since $\mathcal{F}(\llbracket R\rrbracket(\theta;\eta)) \leq_{\mathcal{L}} \llbracket R\rrbracket(\theta;\eta)$
810 $= \llbracket R\,\vec{M}\rrbracket(\theta;\eta)$ by $\llbracket R\,\vec{M}\rrbracket$ def.
811 ◀

812 ▶ **Lemma 30** (Backward closure).
813 *Let t be a term, θ and σ environments, and $\mathcal{A},\mathcal{B} \in \vec{\mathcal{U}} \to \mathcal{P}(\Omega)$.*

814 **1.** *If $t[\theta;\sigma,(\mathtt{rec}\,f.\,t)[\theta;\sigma]/f] \Downarrow c' \in \vec{\mathcal{U}}, \mathcal{A} \to \mathcal{B}$, then $(\mathtt{rec}\,f.\,t)[\theta;\sigma] \in \vec{\mathcal{U}}, \mathtt{in}^*_\mu\,\mathcal{A} \to \mathcal{B}$.*
815 **2.** *If $t[\theta;\sigma,(\mathtt{corec}\,f.\,t)[\theta;\sigma]/f] \Downarrow c' \in \vec{\mathcal{U}}, \mathcal{A} \to \mathcal{B}$, then $(\mathtt{corec}\,f.\,t)[\theta;\sigma] \in \vec{\mathcal{U}}, \mathcal{A} \to \mathtt{out}^*_\nu\,\mathcal{B}$.*

816 **Proof.** **1.** Let $c = (\mathtt{rec}\,f.\,t)[\theta;\sigma]$.
817 Suppose $\vec{M} \in \vec{\mathcal{U}}$ and $v' \in (\mathtt{in}^*_\mu\,\mathcal{A})(\vec{M})$.
818 $v \in \mathtt{in}_\mu\,\mathcal{A}(\vec{M})$ by $\mathtt{in}^*$ def.
819 $v' = \mathtt{in}_\mu\,v$ where $v \in \mathcal{A}(\vec{M})$
820 $t[\theta;\sigma,c/f] \Downarrow c' \in \vec{\mathcal{U}}, \mathcal{A} \to \mathcal{B}$ assumption of lemma
821 $c' \cdot \vec{M}\,v \Downarrow w \in \mathcal{B}(\vec{M})$ by $\vec{\mathcal{U}}, \mathcal{A} \to \mathcal{B}$ def.
822 $c \cdot \vec{M}\,(\mathtt{in}_\mu\,v) \Downarrow w \in \mathcal{B}(\vec{M})$ by $\mathtt{e\text{-}app\text{-}rec}$
823 $c \cdot \vec{M}\,v' \Downarrow w \in \mathcal{B}(\vec{M})$ since $v' = \mathtt{in}_\mu\,v$
824 $c \in \vec{\mathcal{U}}, \mathtt{in}^*_\mu\,\mathcal{A} \to \mathcal{B}$ by $\vec{\mathcal{U}}, \mathcal{A} \to \mathcal{B}$ def.
825
826 **2.** Let $c = (\mathtt{corec}\,f.\,t)[\theta;\sigma]$.
827 Suppose $\vec{M} \in \vec{\mathcal{U}}$ and $v \in \mathcal{A}(\vec{M})$.
828 $t[\theta;\sigma,c/f] \Downarrow c' \in \vec{\mathcal{U}}, \mathcal{A} \to \mathcal{B}$ assumption
829 $c' \cdot \vec{M}\,v \Downarrow w \in \mathcal{B}(\vec{M})$ by $\vec{\mathcal{U}}, \mathcal{A} \to \mathcal{B}$ def.
830 $(c \cdot \vec{M}\ v) \cdot_{\mathtt{out}_\nu} \Downarrow w \in \mathcal{B}(\vec{M})$ by $\mathtt{e\text{-}out}_\nu$
831 $c \cdot \vec{M}\ v \in \mathtt{out}^*_\nu\,(\mathcal{B}(\vec{M}))$ by $\mathtt{out}^*_\nu$ def.
832 $c \in \vec{\mathcal{U}}, \mathcal{A} \to \mathtt{out}^*_\nu\,\mathcal{B}$ by $\vec{\mathcal{U}}, \mathcal{A} \to \mathcal{B}$ def.
833 ◀

834 ▶ **Lemma 31** (Stratified types equivalent to unfolding).
835 *Let $T_{\mathsf{Rec}} \equiv \mathsf{Rec}_K\,(0 \mapsto T_z \mid \mathsf{suc}\,n,\ X \mapsto T_s)$ where $K = \Pi n{:}\,\mathsf{nat}\,.\,\Pi\overrightarrow{u{:}U}.\,*$ and $\Delta;\Xi \vdash T_{\mathsf{Rec}} \Rightarrow$*
836 *K, and $\Delta \vdash \vec{M} : (\overrightarrow{u{:}U})$ and $\Delta \vdash N : \mathsf{nat}$ and $\vdash \theta : \Delta$ and $\eta \in \llbracket\Xi\rrbracket(\theta)$. Then*

837 **1.** $\llbracket T_{\mathsf{Rec}}\,0\,\vec{M}\rrbracket(\theta;\eta) = \mathtt{in}_0\,(\llbracket T_z\,\vec{M}\rrbracket(\theta;\eta))$ *and*
838 **2.** $\llbracket T_{\mathsf{Rec}}\,(\mathsf{suc}\,N)\,\vec{M}\rrbracket(\theta;\eta) = \mathtt{in}_{\mathsf{suc}}\,(\llbracket T_s[N/n;(T_{\mathsf{Rec}}\,N)/X]\,\vec{M}\rrbracket(\theta;\eta))$.

839 **Proof.** Let $\mathcal{C} = \mathtt{in}^*_0\,\llbracket T_z\rrbracket(\theta;\eta)$ and $\mathcal{F} = (N \mapsto \mathcal{X} \mapsto \mathtt{in}^*_{\mathsf{suc}}\,\llbracket T_s\rrbracket(\theta, N/n;\eta,\mathcal{X}/X))$.

840 **1.** $\llbracket T_{\mathsf{Rec}}\,0\,\vec{M}\rrbracket(\theta;\eta)$
841 $= \llbracket T_{\mathsf{Rec}}\rrbracket(\theta;\eta)(0)(\overrightarrow{M[\theta]})$ by $\llbracket T\,\vec{M}\rrbracket$ def.
842 $= \mathbf{Rec}\ \mathcal{C}\,\mathcal{F}\,0\,\overrightarrow{M[\theta]}$ by $\llbracket T_{\mathsf{Rec}}\rrbracket$ def.
843 $= \mathcal{C}\,\overrightarrow{M[\theta]}$ by $\mathbf{Rec}$ def
844 $= (\mathtt{in}^*_0\,\llbracket T_z\rrbracket(\theta;\eta))(\overrightarrow{M[\theta]})$ by $\mathcal{C}$ def.
845 $= \mathtt{in}_0\,(\llbracket T_z\rrbracket(\theta;\eta)(\overrightarrow{M[\theta]}))$ by $\mathtt{in}^*$ def.
846 $= \mathtt{in}_0\,(\llbracket T_z\,\vec{M}\rrbracket(\theta;\eta))$ by $\llbracket T\,\vec{M}\rrbracket$ def.
847

848 **2.** $[\![T_{\texttt{Rec}}\,(\mathsf{suc}\,N)\,\vec{M}]\!](\theta;\eta)$

849 $= [\![T_{\texttt{Rec}}]\!](\theta;\eta)(\mathsf{suc}\,N[\theta])(\overrightarrow{M[\theta]})$ by $[\![T\,\vec{M}]\!]$ def.

850 $= \mathbf{Rec}\,\mathcal{C}\,\mathcal{F}\,(\mathsf{suc}\,N[\theta])\,\overrightarrow{M[\theta]}$ by $[\![T_{\texttt{Rec}}]\!]$ def.

851 $= \mathcal{F}\,N[\theta]\,(\mathbf{Rec}\,\mathcal{C}\,\mathcal{F}\,N[\theta])\,\overrightarrow{M[\theta]}$ by **Rec** def.

852 $= (\mathtt{in}^*_{\mathsf{suc}}\,[\![T_s]\!](\theta, N[\theta]/n;\eta,(\mathbf{Rec}\,\mathcal{C}\,\mathcal{F}\,N[\theta])/X))(\overrightarrow{M[\theta]})$ by $\mathcal{F}$ def.

853 $= \mathtt{in}_{\mathsf{suc}}\,([\![T_s]\!](\theta, N[\theta]/n;\eta,(\mathbf{Rec}\,\mathcal{C}\,\mathcal{F}\,N[\theta])/X)(\overrightarrow{M[\theta]}))$ by $\mathtt{in}^*$ def.

854 $= \mathtt{in}_{\mathsf{suc}}\,([\![T_s]\!]((\mathsf{id}_\Delta, N/n)[\theta];\eta,[\![T_{\texttt{Rec}}\,N]\!](\theta;\eta)/X)(\overrightarrow{M[\theta]}))$ by Def. 2 and $[\![T_{\texttt{Rec}}]\!]$ def.

855 $= \mathtt{in}_{\mathsf{suc}}\,([\![T_s[N/n;(T_{\texttt{Rec}}\,N)/X]]\!](\theta;\eta)(\overrightarrow{M[\theta]}))$ by Lemma 21

856 $= \mathtt{in}_{\mathsf{suc}}\,([\![T_s[N/n;(T_{\texttt{Rec}}\,N)/X]\,\vec{M}]\!](\theta;\eta))$ by $[\![T\,\vec{M}]\!]$ def.

857

858 ◀

859 ▶ **Theorem 32** (Termination of evaluation)**.** *If $\Delta;\Xi;\Gamma \vdash t \Leftarrow T$ or $\Delta;\Xi;\Gamma \vdash t \Rightarrow T$, and*
860 *$\vdash \theta : \Delta$ and $\eta \in [\![\Xi]\!](\theta)$ and $\sigma \in [\![\Gamma]\!](\theta;\eta)$, then $t[\theta;\sigma] \Downarrow v$ for some $v \in [\![T]\!](\theta;\eta)$.*

861 **Proof.** The proof is by induction on the typing derivation. Technically this is a mutual
862 induction on the dual judgments of type checking and synthesis. In each case we introduce
863 the assumptions $\vdash \theta : \Delta$ and $\eta \in [\![\Xi]\!](\theta)$ and $\sigma \in [\![\Gamma]\!](\theta;\eta)$, where Δ, Ξ and Γ appear in
864 the conclusion of the relevant typing rule. We will slightly abuse notation to introduce an
865 existentially quantified variable in the judgment $t[\theta;\sigma] \Downarrow v \in \mathcal{V}$, to mean that $\exists v.\, t[\theta;\sigma] \Downarrow$
866 $v \wedge v \in \mathcal{V}$. Note that the cases involving stratified types and induction over indices are
867 specific to the particular index language (and assume an induction principle over closed index
868 types). The rest of the proof is generic, only assuming the properties in Section 2.

869 **Case:**

870 $$\frac{}{\Delta;\Xi;\Gamma \vdash \langle\rangle \Leftarrow 1}\ \texttt{t-unit}$$

871 $\langle\rangle[\theta;\sigma] \Downarrow \langle\rangle$ by `e-unit`

872 $\langle\rangle \in [\![1]\!](\theta;\eta)$ by $[\![1]\!]$ def.

873

874 **Case:**

875 $$\frac{x{:}T \in \Gamma}{\Delta;\Xi;\Gamma \vdash x \Rightarrow T}\ \texttt{t-var}$$

876 $\sigma \in [\![\Gamma]\!](\theta;\eta)$ assumption of Thm 32

877 $x{:}T \in \Gamma$ premise of `t-var`

878 $\sigma(x) = v \in [\![T]\!](\theta;\eta)$ by Def. 18

879 $x[\theta;\sigma] \Downarrow v$ by `e-var`

880

881 **Case:**

882 $$\frac{\Delta,\overrightarrow{u{:}U};\Xi;\Gamma,x{:}R \vdash s \Leftarrow S}{\Delta;\Xi;\Gamma \vdash \lambda\,\vec{u},x.\,s \Leftarrow (\overrightarrow{u{:}U});\,R \to S}\ \texttt{t-lam}$$

883 Let c be the closure $(\lambda\,\vec{u},x.\,s)[\theta;\sigma]$.

884 $(\lambda\,\vec{u},x.\,s)[\theta;\sigma] \Downarrow c$ by `e-lam`

885 Suffices to show $c \in [\![(\overrightarrow{u{:}U});\,R \to S]\!](\theta;\eta)$, i.e.

886 $\forall \vec{M} \in [\![(\overrightarrow{u{:}U})]\!](\theta).\ \forall v \in [\![R]\!](\theta';\eta).\ c \cdot \vec{M}\ v \Downarrow w \in [\![S]\!](\theta';\eta)$, where $\theta' = \theta, \overrightarrow{M/u}$.
887 Suppose $\vec{M} \in [\![(\overrightarrow{u{:}U})]\!](\theta)$ and $v \in [\![R]\!](\theta';\eta)$ where $\theta' = \theta, \overrightarrow{M/u}$.
888 $\vdash \theta : \Delta$ assumption of Thm 32
889 $\vdash \theta' : \Delta, \overrightarrow{u{:}U}$ by index substitution typing
890 Let $\sigma' = \sigma, v/x$.
891 $\sigma \in [\![\Gamma]\!](\theta;\eta)$ assumption of Thm 32
892 $\sigma \in [\![\Gamma]\!](\theta';\eta)$ since $\vec{u} \notin \mathrm{FV}(\Gamma)$
893 $\sigma' \in [\![\Gamma, x{:}R]\!](\theta';\eta)$ by Def. 18
894 $s[\theta';\sigma'] \Downarrow w \in [\![S]\!](\theta';\eta)$ by I.H. with θ', η and σ'
895 $c \cdot \vec{M}\ v \Downarrow w$ by `e-app-lam`
896

897 **Case:**

898 $$\frac{\Delta;\Xi;\Gamma \vdash q \Rightarrow (\overrightarrow{u{:}U});\ R \to S \quad \Delta \vdash \vec{M} : (\overrightarrow{u{:}U}) \quad \Delta;\Xi;\Gamma \vdash r \Leftarrow R[\overrightarrow{M/u}]}{\Delta;\Xi;\Gamma \vdash q\ \vec{M}\ r \Rightarrow S[\overrightarrow{M/u}]}\ \texttt{t-app}$$

899 $q[\theta;\sigma] \Downarrow c \in [\![(\overrightarrow{u{:}U});\ R \to S]\!](\theta;\eta)$ by I.H.
900 $c \in [\![(\overrightarrow{u{:}U})]\!](\theta)$, $(\vec{N} \mapsto [\![R]\!](\theta, \overrightarrow{M/u};\eta)) \longrightarrow (\vec{N} \mapsto [\![S]\!](\theta, \overrightarrow{M/u};\eta))$ by $[\![(\overrightarrow{u{:}U});\ R \to S]\!](\theta;\eta)$
901 def.
902 $\overrightarrow{M[\theta]} \in [\![(\overrightarrow{u{:}U})]\!](\theta)$ by Lemma 12.2
903 $r[\theta;\sigma] \Downarrow v \in [\![R[\overrightarrow{M/u}]]\!](\theta;\eta)$ by I.H.
904 $v \in [\![R]\!]((\mathsf{id}_\Delta, \overrightarrow{M/u})[\theta];\eta)$ by Lemma 21
905 $v \in [\![R]\!](\theta, \overrightarrow{M[\theta]/u};\eta)$ by Def. 2
906 $c \cdot \overrightarrow{M[\theta]}\ v \Downarrow w \in [\![S]\!](\theta, \overrightarrow{M[\theta]/u};\eta)$ by Thm 15
907 $w \in [\![S]\!]((\mathsf{id}_\Delta, \overrightarrow{M/u})[\theta];\eta)$ by Def. 2
908 $w \in [\![S[\overrightarrow{M/u}]]\!](\theta;\eta)$ by Lemma 21
909 $(q\ \vec{M}\ r)[\theta;\sigma] \Downarrow w$ by `e-app`
910

911 **Case:**

912 $$\frac{\Delta;\Xi;\Gamma \vdash t_1 \Leftarrow T_1 \quad \Delta;\Xi;\Gamma \vdash t_2 \Leftarrow T_2}{\Delta;\Xi;\Gamma \vdash \langle t_1, t_2\rangle \Leftarrow T_1 \times T_2}\ \texttt{t-pair}$$

913 $t_i[\theta;\sigma] \Downarrow v_i \in [\![T_i]\!](\theta;\eta)$ for $i \in \{1,2\}$ by I.H.
914 $\langle t_1, t_2\rangle[\theta;\sigma] \Downarrow \langle v_1, v_2\rangle$ by `e-pair`
915 $\langle v_1, v_2\rangle \in [\![T_1 \times T_2]\!](\theta;\eta)$ by $[\![T_1 \times T_2]\!]$ def.
916

917 **Case:**

918 $$\frac{\Delta;\Xi;\Gamma \vdash p \Rightarrow T_1 \times T_2 \quad \Delta;\Xi;\Gamma, x_1{:}T_1, x_2{:}T_2 \vdash s \Leftarrow T}{\Delta;\Xi;\Gamma \vdash \texttt{split}\ p\ \texttt{as}\ \langle x_1, x_2\rangle\ \texttt{in}\ s \Leftarrow T}\ \texttt{t-split}$$

919 $p[\theta;\sigma] \Downarrow w \in [\![T_1 \times T_2]\!](\theta;\eta)$ by I.H.
920 $w = \langle v_1, v_2\rangle$ where $v_i \in [\![T_i]\!](\theta;\eta)$ for $i \in \{1,2\}$ by $[\![T_1 \times T_2]\!]$ def.
921 $\sigma, v_1/x_1, v_2/x_2 \in [\![\Gamma, x_1{:}T_1, x_2{:}T_2]\!](\theta;\eta)$ by Def. 18
922 $s[\theta;\sigma, v_1/x_1, v_2/x_2] \Downarrow v \in [\![T]\!](\theta;\eta)$ by I.H.
923 $t[\theta;\sigma] \Downarrow v$ by `e-split`
924

925 **Case:**

926 $$\frac{\Delta; \Xi; \Gamma \vdash t_i \Leftarrow T_i}{\Delta; \Xi; \Gamma \vdash \mathtt{in}_i\, t_i \Leftarrow T_1 + T_2}\ \mathtt{t\text{-}in}_i \qquad \text{for } i \in \{1,2\}$$

927 This is really two cases. Fix $i \in \{1,2\}$.

928 $t_i[\theta;\sigma] \Downarrow v_i \in [\![T_i]\!](\theta;\eta)$ by I.H.

929 $(\mathtt{in}_i\, t_i)[\theta;\sigma] \Downarrow \mathtt{in}_i\, v_i$ by $\mathtt{e\text{-}in}_i$.

930 $\mathtt{in}_i\, v_i \in \mathtt{in}_i\, [\![T_i]\!](\theta;\eta) \subseteq [\![T_1 + T_2]\!](\theta;\eta)$ by $[\![T_1 + T_2]\!]$ def.

931

932 **Case:**

933 $$\frac{\Delta; \Xi; \Gamma \vdash s \Rightarrow T_1 + T_2 \quad \Delta; \Xi; \Gamma, x_1{:}T_1 \vdash t_1 \Leftarrow T \quad \Delta; \Xi; \Gamma, x_2{:}T_2 \vdash t_2 \Leftarrow T}{\Delta; \Xi; \Gamma \vdash (\mathtt{case}\, s\, \mathtt{of}\, \mathtt{in}_1\, x_1 \mapsto t_1 \mid \mathtt{in}_2\, x_2 \mapsto t_2) \Leftarrow T}\ \mathtt{t\text{-}case}$$

934 $s[\theta;\sigma] \Downarrow w \in [\![T_1 + T_2]\!](\theta;\eta)$ by I.H.

935 $w \in \mathtt{in}_1\, [\![T_1]\!](\theta;\eta)$ or $w \in \mathtt{in}_2\, [\![T_2]\!](\theta;\eta)$ by $[\![T_1 + T_2]\!]$ def.

936 $w = \mathtt{in}_i\, v_i$ where $v_i \in [\![T_i]\!](\theta;\eta)$, for some $i \in \{1,2\}$.

937 $\sigma \in [\![\Gamma]\!](\theta;\eta)$ by assumption of Thm 32

938 $\sigma, v_i/x_i \in [\![\Gamma, x_i{:}T_i]\!](\theta;\eta)$ by Def. 18

939 $t_i[\theta;\sigma, v_i/x_i] \Downarrow v \in [\![T]\!](\theta;\eta)$ by I.H. on t_i

940 $t[\theta;\sigma] \Downarrow v$ by $\mathtt{e\text{-}case\text{-}in}_i$

941

942 **Case:**

943 $$\frac{\Delta \vdash M : U \quad \Delta; \Xi; \Gamma \vdash s \Leftarrow S[M/u]}{\Delta; \Xi; \Gamma \vdash \mathtt{pack}\,(M, s) \Leftarrow \Sigma u{:}U.\, S}\ \mathtt{t\text{-}pack}$$

944 $s[\theta;\sigma] \Downarrow w \in [\![S[M/u]]\!](\theta;\eta)$ by I.H.

945 $w \in [\![S]\!]((\mathsf{id}_\Delta, M/u)[\theta];\eta)$ by Lemma 21

946 $w \in [\![S]\!](\theta, M[\theta]/u;\eta)$ by Def. 2

947 $(\mathtt{pack}\,(M, s))[\theta;\sigma] \Downarrow \mathtt{pack}\,(M[\theta], w)$ by $\mathtt{e\text{-}pack}$

948 $M[\theta] \in [\![U]\!](\theta)$ by Lemma 12.1

949 $\mathtt{pack}\,(M[\theta], w) \in [\![\Sigma u{:}U.\, S]\!](\theta;\eta)$ by $[\![\Sigma u{:}U.\, S]\!]$ def.

950

951 **Case:**

952 $$\frac{\Delta; \Xi; \Gamma \vdash p \Rightarrow \Sigma u{:}U.\, S \quad \Delta, u{:}U; \Xi; \Gamma, x{:}S \vdash q \Leftarrow T}{\Delta; \Xi; \Gamma \vdash \mathtt{unpack}\, p\, \mathtt{as}\, (u, x)\, \mathtt{in}\, q \Leftarrow T}\ \mathtt{t\text{-}unpack}$$

953 $p[\theta;\sigma] \Downarrow w' \in [\![\Sigma u{:}U.\, S]\!](\theta;\eta)$ by I.H.

954 $w' = \mathtt{pack}\,(M, w)$ where $M \in [\![U]\!](\theta)$ and $w \in [\![S]\!](\theta, M/u;\eta)$ by $[\![\Sigma u{:}U.\, S]\!]$ def.

955 Let $\theta' = \theta, M/u$.

956 $\vdash \theta' : \Delta, u{:}U$ by index substitution typing

957 $\sigma \in [\![\Gamma]\!](\theta;\eta)$ assumption of Thm 32

958 $\sigma \in [\![\Gamma]\!](\theta';\eta)$ since $u \notin \mathrm{FV}(\Gamma)$

959 Let $\sigma' = \sigma, w/x$.

960 $\sigma' \in [\![\Gamma, x{:}S]\!](\theta';\eta)$ by Def. 18

961 $q[\theta';\sigma'] \Downarrow v \in [\![T]\!](\theta';\eta)$ by I.H.

962 $(\mathtt{unpack}\, p\, \mathtt{as}\, (u, x)\, \mathtt{in}\, q)[\theta;\sigma] \Downarrow v$ by $\mathtt{e\text{-}unpack}$

963 $v \in [\![T]\!](\theta;\eta)$ since $u \notin \mathrm{FV}(T)$

964

965 **Case:**

966 $$\frac{\Delta \vdash M = N}{\Delta; \Xi; \Gamma \vdash \texttt{refl} \Leftarrow M = N}\ \texttt{t-refl}$$

967 $\vdash M[\theta] = N[\theta]$ by Req. 3.3
968 $\texttt{refl} \in [\![M = N]\!](\theta; \eta)$ by $[\![M = N]\!]$ def.
969 $\texttt{refl}[\theta; \sigma] \Downarrow \texttt{refl}$ by `e-refl`
970

971 **Case:**

972 $$\frac{\Delta; \Xi; \Gamma \vdash q \Rightarrow M = N \quad \Delta \vdash M \doteq N \searrow (\Delta' \mid \Theta) \quad \Delta'; \Xi[\Theta]; \Gamma[\Theta] \vdash s \Leftarrow T[\Theta]}{\Delta; \Xi; \Gamma \vdash \texttt{eq}\, q\, \texttt{with}\, (\Delta'.\Theta \mapsto s) \Leftarrow T}\ \texttt{t-eq}$$

973 $q[\theta; \sigma] \Downarrow w \in [\![M = N]\!](\theta; \eta)$ by I.H.
974 $w = \texttt{refl}$ and $\vdash M[\theta] = N[\theta]$ by $[\![M = N]\!]$ def.
975 $\Delta \vdash M \doteq N \searrow (\Delta' \mid \Theta)$ premise of `t-eq`
976 $\vdash \theta : \Delta$ assumption of Thm 32
977 $\Delta' \vdash \Theta \doteq \theta \searrow (\cdot \mid \theta')$ by Req. 6
978 $[\![\Xi[\Theta]]\!](\theta') = [\![\Xi]\!](\Theta[\theta'])$ and $[\![T[\Theta]]\!](\theta'; \eta) = [\![T]\!](\Theta[\theta']; \eta)$ by Lemma 21
979 $[\![\Xi[\Theta]]\!](\theta') = [\![\Xi]\!](\theta])$ and $[\![T[\Theta]]\!](\theta'; \eta) = [\![T]\!](\theta; \eta)$ by Thm 5.2
980 Extending Lemma 21 from types T to typing contexts Γ,
981 $[\![\Gamma[\Theta]]\!](\theta'; \eta) = [\![\Gamma]\!](\Theta[\theta']; \eta) = [\![\Gamma]\!](\theta; \eta)$.
982 $\eta \in [\![\Xi[\Theta]]\!](\theta')$ since $\eta \in [\![\Xi]\!](\theta)$
983 $\sigma \in [\![\Gamma[\Theta]]\!](\theta'; \eta)$ since $\sigma \in [\![\Gamma]\!](\theta; \eta)$
984 $s[\theta'; \sigma] \Downarrow v \in [\![T[\Theta]]\!](\theta'; \eta)$ by I.H.
985 $v \in [\![T]\!](\theta; \eta)$
986 $(\texttt{eq}\, q\, \texttt{with}\, (\Delta'.\Theta \mapsto s))[\theta; \sigma] \Downarrow v$ by `e-eq`
987

988 **Case:**

989 $$\frac{\Delta; \Xi; \Gamma \vdash s \Rightarrow M = N \quad \Delta \vdash M \doteq N \searrow \#}{\Delta; \Xi; \Gamma \vdash \texttt{eq_abort}\, s \Leftarrow T}\ \texttt{t-eqfalse}$$

990 $s[\theta; \sigma] \Downarrow w \in [\![M = N]\!](\theta; \eta)$ by I.H.
991 $w = \texttt{refl}$ and $\vdash M[\theta] = N[\theta]$ by $[\![M = N]\!]$ def.
992 θ unifies M and N in Δ by def. of a unifier
993 $\Delta \vdash M \doteq N \searrow \#$ premise of `t-eqfalse`
994 There is no unifier for M and N consequence of Req. 4
995 Contradiction: derive $(\texttt{eq_abort}\, s)[\theta; \sigma] \Downarrow v \in [\![T]\!](\theta; \eta)$
996

997 **Case:**

998 $$\frac{\Delta; \Xi; \Gamma \vdash s \Leftarrow S[\overrightarrow{M/u}; \mu X{:}K.\, \Lambda \vec{u}.\, S/X]}{\Delta; \Xi; \Gamma \vdash \texttt{in}_\mu\, s \Leftarrow (\mu X{:}K.\, \Lambda \vec{u}.\, S)\, \vec{M}}\ \texttt{t-in}_\mu$$

999 Let $R = \mu X{:}K.\, \Lambda \vec{u}.\, S$.
1000 $s[\theta; \sigma] \Downarrow w \in [\![S[\overrightarrow{M/u}; R/X]]\!](\theta; \eta)$ by I.H.
1001 $(\texttt{in}_\mu\, s)[\theta; \sigma] \Downarrow \texttt{in}_\mu\, w$ by $\texttt{e-in}_\mu$
1002 $\texttt{in}_\mu\, w \in [\![R\, \vec{M}]\!](\theta; \eta)$ by Lemma 29
1003

1004 **Case:**

$$\frac{\Delta; \Xi, X{:}K; \Gamma, f{:}(\overrightarrow{u{:}U});\ X\,\vec{u} \to S \vdash s \Leftarrow (\overrightarrow{u{:}U});\ R[\overrightarrow{u/u'}] \to S \quad \vec{u} \notin \mathrm{FV}(R)}{\Delta; \Xi; \Gamma \vdash \mathtt{rec}\, f.\, s \Leftarrow (\overrightarrow{u{:}U});\ (\mu X{:}K.\, \Lambda\, \overrightarrow{u'}.\, R)\,\vec{u} \to S}\ \mathtt{t\text{-}rec} \quad 1005$$

1006 Let $c = (\mathtt{rec}\, f.\, s)[\theta; \sigma]$ and $C = (\mu X{:}K.\, \Lambda\, \overrightarrow{u'}.\, R)\,\vec{u}$. By e-rec, $(\mathtt{rec}\, f.\, s)[\theta; \sigma] \Downarrow c$. We need
1007 to show that $c \in [\![(\overrightarrow{u{:}U});\ C \to S]\!](\theta; \eta)$,

1008 Let $\vec{\mathcal{U}} = [\![(\overrightarrow{u{:}U})]\!](\theta) = [\![(\overrightarrow{u'{:}U})]\!](\theta)$ (both $\vec{u}$ and $\overrightarrow{u'}$ do not appear in θ). From the kinding
1009 rules we know that $K = \Pi\overrightarrow{u'{:}U}.\, *$ so $[\![K]\!](\theta) = \vec{\mathcal{U}} \to \mathcal{P}(\Omega)$. Let $\mathcal{L} = [\![K]\!](\theta)$ and define
1010 $\mathcal{F} \in \mathcal{L} \to \mathcal{L}$ by $\mathcal{F}(\mathcal{X}) = \mathtt{in}^*_\mu\, [\![\Lambda\, \overrightarrow{u'}.\, R]\!](\theta; \eta, \mathcal{X}/X)$.

1011 For $\vec{M} \in \vec{\mathcal{U}}$,

1012 $$[\![C]\!](\theta, \overrightarrow{M/u}; \eta) = [\![\mu X{:}K.\, \Lambda\, \overrightarrow{u'}.\, R]\!](\theta, \overrightarrow{M/u}; \eta)(\vec{u}[\theta, \overrightarrow{M/u}]) \qquad \text{by } [\![T]\!] \text{ def}$$
1013 $$= [\![\mu X{:}K.\, \Lambda\, \overrightarrow{u'}.\, R]\!](\theta; \eta)(\vec{M})$$
1014 dropping mappings for $\vec{u}$ since $\vec{u} \notin \mathrm{FV}(R)$
1015 $$= \boldsymbol{\mu}(\mathcal{X} \mapsto \mathtt{in}^*_\mu\, [\![\Lambda\, \overrightarrow{u'}.\, R]\!](\theta; \eta, \mathcal{X}/X))(\vec{M}) \qquad \text{by } [\![T]\!] \text{ def}$$
1016 $$= (\boldsymbol{\mu}\mathcal{F})(\vec{M}) \qquad \text{by } \mathcal{F} \text{ def.}$$

1017
1018

1019 Define $\mathcal{B} \in \mathcal{L}$ by $\mathcal{B}(\vec{M}) = [\![S]\!](\theta, \overrightarrow{M/u}; \eta)$. Then

1020 $$[\![(\overrightarrow{u{:}U});\ C \to S]\!](\theta; \eta) = \vec{\mathcal{U}},\, \boldsymbol{\mu}\mathcal{F} \to \mathcal{B} \qquad \text{by } [\![T]\!] \text{ def.}$$

1021 We want to show $c \in \vec{\mathcal{U}},\, \boldsymbol{\mu}\mathcal{F} \to \mathcal{B}$. We will instead prove the sufficient condition given in
1022 Lemma 28. To this end, suppose $\mathcal{X} \in \vec{\mathcal{U}} \to \mathcal{P}(\Omega) = \mathcal{L}$ and $c \in \vec{\mathcal{U}},\, \mathcal{X} \to \mathcal{B}$. The goal is now
1023 to show $c \in \vec{\mathcal{U}},\, \mathcal{F}(\mathcal{X}) \to \mathcal{B}$.

1024 Define $\mathcal{A} \in \mathcal{L}$ by $\mathcal{A} = [\![\Lambda\, \overrightarrow{u'}.\, R]\!](\theta; \eta, \mathcal{X}/X)$, so $\mathcal{F}(\mathcal{X}) = \mathtt{in}^*_\mu\, \mathcal{A}$. By Lemma 30, it suffices
1025 to show that

1026 $$s[\theta; \sigma, c/f] \Downarrow c' \in \vec{\mathcal{U}},\, \mathcal{A} \to \mathcal{B}.$$

1027 We need to interpret the types $(\overrightarrow{u{:}U});\ X\,\vec{u} \to S$ and $(\overrightarrow{u{:}U});\ R[\overrightarrow{u/u'}] \to S$ appearing in the
1028 premise of t-rec. Note that these types are well-kinded under the contexts $\Delta; \Xi, X{:}K$. Since
1029 $\mathcal{X} \in \mathcal{L} = [\![K]\!](\theta)$, we interpret them under the environments θ and $\eta, \mathcal{X}/X \in [\![\Xi, X{:}K]\!](\theta)$.

1030 $$[\![(\overrightarrow{u{:}U});\ X\,\vec{u} \to S]\!](\theta; \eta, \mathcal{X}/X)$$
1031 $$= \vec{\mathcal{U}},\, (\vec{M} \mapsto [\![X\,\vec{u}]\!](\theta, \overrightarrow{M/u}; \eta, \mathcal{X}/X)) \to (\vec{M} \mapsto [\![S]\!](\theta, \overrightarrow{M/u}; \eta, \mathcal{X}/X))$$
1032 $$= \vec{\mathcal{U}},\, (\vec{M} \mapsto \mathcal{X}(\vec{M})) \to (\vec{M} \mapsto [\![S]\!](\theta, \overrightarrow{M/u}; \eta))$$
1033 dropping a mapping for X since $X \notin \mathrm{FV}(S)$
1034 $$= \vec{\mathcal{U}},\, \mathcal{X} \to \mathcal{B}$$
1035

$$
\begin{aligned}
&[\![(\overrightarrow{u{:}U});\ R[\overrightarrow{u/u'}] \to S]\!](\theta;\eta,\mathcal{X}/X)\\
&= \vec{\mathcal{U}},\ (\vec{M} \mapsto [\![R[\overrightarrow{u/u'}]]\!](\theta,\overrightarrow{M/u};\eta,\mathcal{X}/X)) \to (\vec{M} \mapsto [\![S]\!](\theta,\overrightarrow{M/u};\eta,\mathcal{X}/X))\\
&= \vec{\mathcal{U}},\ (\vec{M} \mapsto [\![R]\!]((\mathsf{id}_\Delta,\overrightarrow{u/u'})[\theta,\overrightarrow{M/u'}];\eta,\mathcal{X}/X)) \to (\vec{M} \mapsto [\![S]\!](\theta,\overrightarrow{M/u};\eta))\\
&\qquad \text{by Lemma 21 and again dropping a mapping for } X\\
&= \vec{\mathcal{U}},\ (\vec{M} \mapsto [\![R]\!](\theta,\overrightarrow{M/u'};\eta,\mathcal{X}/X)) \to (\vec{M} \mapsto [\![S]\!](\theta,\overrightarrow{M/u};\eta))\\
&= \vec{\mathcal{U}},\ \mathcal{A} \to \mathcal{B}.
\end{aligned}
$$

Our assumption from Lemma 28 is that $c \in \vec{\mathcal{U}},\ \mathcal{X} \to \mathcal{B}$. Moreover, since $X \notin \mathrm{FV}(\Gamma)$, $[\![\Gamma]\!](\theta;\eta,\mathcal{X}/X) = [\![\Gamma]\!](\theta;\eta) \ni \sigma$. Hence $\sigma, c/f \in [\![\Gamma, f{:}(\overrightarrow{u{:}U});\ X\,\vec{u} \to S]\!](\theta;\eta,\mathcal{X}/X)$. Now we can apply the induction hypothesis with $\eta' = \eta,\mathcal{X}/X$ and $\sigma' = \sigma, c/f$ to learn that $s[\theta;\sigma'] \Downarrow c'$ where $c' \in [\![(\overrightarrow{u{:}U});\ R[\overrightarrow{u/u'}] \to S]\!](\theta;\eta') = \vec{\mathcal{U}},\ \mathcal{A} \to \mathcal{B}$.

Case:

$$
\frac{\Delta;\Xi,X{:}K;\Gamma,f{:}(\overrightarrow{u{:}U});\ S \to X\,\vec{u} \vdash s \Leftarrow (\overrightarrow{u{:}U});\ S \to R[\overrightarrow{u/u'}] \quad \vec{u} \notin \mathrm{FV}(R)}{\Delta;\Xi;\Gamma \vdash \mathtt{corec}\,f.\,s \Leftarrow (\overrightarrow{u{:}U});\ S \to (\nu X{:}K.\,\Lambda\,\vec{u'}.\,R)\,\vec{u}}\ \texttt{t-corec}
$$

Let $c = (\mathtt{corec}\,f.\,s)[\theta;\sigma]$ and $C = (\nu X{:}K.\,\Lambda\,\vec{u'}.\,R)\,\vec{u}$. By e-corec, $(\mathtt{corec}\,f.\,s)[\theta;\sigma] \Downarrow c$. We need to show that $c \in [\![(\overrightarrow{u{:}U});\ S \to C]\!](\theta;\eta)$,

Let $\vec{\mathcal{U}} = [\![(\overrightarrow{u{:}U})]\!](\theta) = [\![(\overrightarrow{u'{:}U})]\!](\theta)$ (both $\vec{u}$ and $\vec{u'}$ do not appear in θ). From the kinding rules we know that $K = \Pi\overrightarrow{u'{:}U}.*$ so $[\![K]\!](\theta) = \vec{\mathcal{U}} \to \mathcal{P}(\Omega)$. Let $\mathcal{L} = [\![K]\!](\theta)$ and define $\mathcal{F} \in \mathcal{L} \to \mathcal{L}$ by $\mathcal{F}(\mathcal{X}) = \mathtt{out}^*_\nu\,([\![\Lambda\,\vec{u'}.\,R]\!](\theta;\eta,\mathcal{X}/X))$.

For $\vec{M} \in \vec{\mathcal{U}}$,

$$
\begin{aligned}
[\![C]\!](\theta,\overrightarrow{M/u};\eta) &= [\![\nu X{:}K.\,\Lambda\,\vec{u'}.\,R]\!](\theta,\overrightarrow{M/u};\eta)(\vec{u}[\theta,\overrightarrow{M/u}]) && \text{by } [\![T]\!] \text{ def}\\
&= [\![\nu X{:}K.\,\Lambda\,\vec{u'}.\,R]\!](\theta;\eta)(\vec{M})\\
&\qquad \text{dropping mappings for } \vec{u} \text{ since } \vec{u} \notin \mathrm{FV}(R)\\
&= \boldsymbol{\nu}(\mathcal{X} \mapsto \mathtt{out}^*_\nu\,([\![\Lambda\,\vec{u'}.\,R]\!](\theta;\eta,\mathcal{X}/X)))(\vec{M}) && \text{by } [\![T]\!] \text{ def}\\
&= (\boldsymbol{\nu}\mathcal{F})(\vec{M}) && \text{by } \mathcal{F} \text{ def.}
\end{aligned}
$$

Define $\mathcal{A} \in \mathcal{L}$ by $\mathcal{A}(\vec{M}) = [\![S]\!](\theta,\overrightarrow{M/u};\eta)$. Then

$$
[\![(\overrightarrow{u{:}U});\ S \to C]\!](\theta;\eta) = \vec{\mathcal{U}},\ A \to \boldsymbol{\nu}\mathcal{F} \qquad \text{by } [\![T]\!] \text{ def.}
$$

We want to show $c \in \vec{\mathcal{U}},\ \mathcal{A} \to \boldsymbol{\nu}\mathcal{F}$. We will instead prove the sufficient condition given in Lemma 28. To this end, suppose $\mathcal{X} \in \vec{\mathcal{U}} \to \mathcal{P}(\Omega) = \mathcal{L}$ and $c \in \vec{\mathcal{U}},\ \mathcal{A} \to \mathcal{X}$. The goal is now to show $c \in \vec{\mathcal{U}},\ \mathcal{A} \to \mathcal{F}(\mathcal{X})$.

Define $\mathcal{B} \in \mathcal{L}$ by $\mathcal{B} = [\![\Lambda\,\vec{u'}.\,R]\!](\theta;\eta,\mathcal{X}/X)$, so $\mathcal{F}(\mathcal{X}) = \mathtt{out}^*_\nu\,\mathcal{B}$. By Lemma 30, it suffices to show that

$$
s[\theta;\sigma,c/f] \Downarrow c' \in \vec{\mathcal{U}},\ \mathcal{A} \to \mathcal{B}.
$$

We need to interpret the types $(\overrightarrow{u{:}U});\ S \to X\,\vec{u}$ and $(\overrightarrow{u{:}U});\ S \to R[\overrightarrow{u/u'}]$ appearing in the premise of t-corec. Note that these types are well-kinded under the contexts

$\Delta;\Xi, X{:}K$. Since $\mathcal{X} \in \mathcal{L} = [\![K]\!](\theta)$, we interpret them under the environments θ and $\eta, \mathcal{X}/X \in [\![\Xi, X{:}K]\!](\theta)$.

$$
\begin{aligned}
& [\![(\overrightarrow{u{:}U});\, S \to X\,\vec{u}]\!](\theta;\eta,\mathcal{X}/X) \\
&= \vec{\mathcal{U}},\, (\vec{M} \mapsto [\![S]\!](\theta, \overrightarrow{M/u}; \eta, \mathcal{X}/X)) \to (\vec{M} \mapsto [\![X\,\vec{u}]\!](\theta, \overrightarrow{M/u}; \eta, \mathcal{X}/X)) \\
&= \vec{\mathcal{U}},\, (\vec{M} \mapsto [\![S]\!](\theta, \overrightarrow{M/u}; \eta)) \to (\vec{M} \mapsto \mathcal{X}(\vec{M})) \\
&\qquad \text{dropping a mapping for } X \text{ since } X \notin \mathrm{FV}(S) \\
&= \vec{\mathcal{U}},\, \mathcal{A} \to \mathcal{X}
\end{aligned}
$$

$$
\begin{aligned}
& [\![(\overrightarrow{u{:}U});\, S \to R[\overrightarrow{u/u'}]]\!](\theta;\eta,\mathcal{X}/X) \\
&= \vec{\mathcal{U}},\, (\vec{M} \mapsto [\![S]\!](\theta, \overrightarrow{M/u}; \eta, \mathcal{X}/X)) \to (\vec{M} \mapsto [\![R[\overrightarrow{u/u'}]]\!](\theta, \overrightarrow{M/u}; \eta, \mathcal{X}/X)) \\
&= \vec{\mathcal{U}},\, (\vec{M} \mapsto [\![S]\!](\theta, \overrightarrow{M/u}; \eta)) \to (\vec{M} \mapsto [\![R]\!]((\mathsf{id}_\Delta, \overrightarrow{u/u'})[\theta, \overrightarrow{M/u'}]; \eta, \mathcal{X}/X)) \\
&\qquad \text{by Lemma 21 and again dropping a mapping for } X \\
&= \vec{\mathcal{U}},\, (\vec{M} \mapsto [\![S]\!](\theta, \overrightarrow{M/u}; \eta)) \to (\vec{M} \mapsto [\![R]\!](\theta, \overrightarrow{M/u'}; \eta, \mathcal{X}/X)) \\
&= \vec{\mathcal{U}},\, \mathcal{A} \to \mathcal{B}.
\end{aligned}
$$

Our assumption from Lemma 28 is that $c \in \vec{\mathcal{U}}, \mathcal{A} \to \mathcal{X}$. Moreover, since $X \notin \mathrm{FV}(\Gamma)$, $[\![\Gamma]\!](\theta;\eta,\mathcal{X}/X) = [\![\Gamma]\!](\theta;\eta) \ni \sigma$. Hence $\sigma, c/f \in [\![\Gamma, f{:}(\overrightarrow{u{:}U});\, S \to X\,\vec{u}]\!](\theta;\eta,\mathcal{X}/X)$. Now we can apply the induction hypothesis with $\eta' = \eta, \mathcal{X}/X$ and $\sigma' = \sigma, c/f$ to learn that $s[\theta;\sigma'] \Downarrow c'$ where $c' \in [\![(\overrightarrow{u{:}U});\, S \to R[\overrightarrow{u/u'}]]\!](\theta;\eta') = \vec{\mathcal{U}}, \mathcal{A} \to \mathcal{B}$.

Case:

$$
\frac{\Delta;\Xi;\Gamma \vdash t \Rightarrow (\nu X{:}K.\,\Lambda\,\vec{u}.\,T)\,\vec{M}}{\Delta;\Xi;\Gamma \vdash \mathtt{out}_\nu\, t \Rightarrow T[\overrightarrow{M/u}; \nu X{:}K.\,\Lambda\,\vec{u}.\,T/X]}
$$

$t[\theta;\sigma] \Downarrow v \in [\![(\nu X{:}K.\,\Lambda\,\vec{u}.\,T)\,\vec{M}]\!](\theta;\eta)$ for some v — by I.H.
$v \in [\![\nu X{:}K.\,\Lambda\,\vec{u}.\,T]\!](\theta;\eta)\ (\vec{M}[\theta])$ — by $[\![T]\!]$ def.
$v \in \boldsymbol{\nu}(\mathcal{X} \mapsto \mathtt{out}^*_\nu\,([\![\Lambda\,\vec{u}.\,T]\!](\theta;\eta,\mathcal{X}/X)\}))\ (\vec{M}[\theta])$ — by $[\![T]\!]$ def.
$v \in \bigvee\{\mathcal{C} \in \mathcal{L} \mid \forall \mathcal{X} \in \mathcal{L}.\, \mathcal{C} \le_{\mathcal{L}} \mathcal{X} \implies \mathcal{C} \le_{\mathcal{L}} \mathtt{out}^*_\nu\,([\![\Lambda\,\vec{u}.\,T]\!](\theta;\eta,\mathcal{X}/X))\}\ (\vec{M}[\theta])$
by definition of $\boldsymbol{\nu}$
$v \in \mathcal{C}(\vec{M}[\theta])$ such that $\forall \mathcal{X}.\, \mathcal{C}(\vec{M}[\theta]) \le \mathcal{X}(\vec{M}[\theta])$
$\quad \implies \mathcal{C}(\vec{M}[\theta]) \le (\mathtt{out}^*_\nu\,([\![\Lambda\,\vec{u}.\,T]\!](\theta;\eta,\mathcal{X}/X)))(\vec{M}[\theta])$ — by def of $\bigvee$
The right-hand side can be simplified as $\mathcal{C}(M[\theta]) \le (\mathtt{out}^*_\nu\,([\![T]\!](\theta, \overrightarrow{(M[\theta])/u}; \eta, \mathcal{X}/X)))$ by def of $[\![T]\!]$
Choosing $[\![\nu X{:}K.\,\Lambda\,\vec{u}.\,T]\!](\theta;\eta)$ for $\mathcal{X}$, the left-hand side holds trivially by def. of $[\![\nu X{:}K.\,\Lambda\,\vec{u}.\,T]\!]$ and of $\bigvee$
Hence, $v \in (\mathtt{out}^*_\nu\,([\![T]\!](\theta, \overrightarrow{(M[\theta])/u}; \eta, [\![\nu X{:}K.\,\Lambda\,\vec{u}.\,T]\!](\theta;\eta)/X)))$
$v \cdot_{\mathtt{out}_\nu} \Downarrow w \in [\![T]\!](\theta, \overrightarrow{(M[\theta])/u}; \eta, [\![\nu X{:}K.\,\Lambda\,\vec{u}.\,T]\!](\theta;\eta)/X)$
$v = (\mathtt{corec}\, f.\, t)[\theta', \sigma'] \cdot \vec{N}\ v'$ — by inversion on $\mathtt{e\text{-}corec\text{-}out}_\nu$
$(\mathtt{out}_\nu\, t)[\sigma;\theta] \Downarrow w \in [\![T]\!](\theta, \overrightarrow{(M[\theta])/u}; \eta, [\![\nu X{:}K.\,\Lambda\,\vec{u}.\,T]\!](\theta;\eta)/X)$ — by $\mathtt{e\text{-}out}_\nu$
$\mathtt{out}_\nu\, t \Downarrow w \in [\![T[\overrightarrow{M/u}; (\nu X{:}K.\,\Lambda\,\vec{u}.\,T)/X]]\!](\theta;\eta)$ — by Lemma 21

1109 **Case:**

$$\frac{\Delta; \Xi; \Gamma \vdash s \Leftarrow T_z\, \vec{M}}{\Delta; \Xi; \Gamma \vdash \mathtt{in}_0\, s \Leftarrow T_{\mathtt{Rec}}\, 0\, \vec{M}}\ \mathtt{t\text{-}in}_0$$ 1110

1111 $s[\theta; \sigma] \Downarrow w \in [\![T_z\, \vec{M}]\!](\theta; \eta)$ by I.H.
1112 $(\mathtt{in}_0\, s)[\theta; \sigma] \Downarrow \mathtt{in}_0\, w$ by $\mathtt{e\text{-}in}_0$
1113 $\mathtt{in}_0\, w \in \mathtt{in}_0\, [\![T_z\, \vec{M}]\!](\theta; \eta)$
1114 $\mathtt{in}_0\, w \in [\![T_{\mathtt{Rec}}\, 0\, \vec{M}]\!](\theta; \eta)$ by Lemma 31
1115

1116 **Case:**

$$\frac{\Delta; \Xi; \Gamma \vdash s \Leftarrow T_s[N/u; (T_{\mathtt{Rec}}\, N)/X]\, \vec{M}}{\Delta; \Xi; \Gamma \vdash \mathtt{in}_{\mathsf{suc}}\, s \Leftarrow T_{\mathtt{Rec}}\, (\mathsf{suc}\, N)\, \vec{M}}\ \mathtt{t\text{-}in}_{\mathsf{suc}}$$ 1117

1118 $s[\theta; \sigma] \Downarrow w \in [\![T_s[N/u; (T_{\mathtt{Rec}}\, N)/X]\, \vec{M}]\!](\theta; \eta)$ by I.H.
1119 $(\mathtt{in}_{\mathsf{suc}}\, s)[\theta; \sigma] \Downarrow \mathtt{in}_{\mathsf{suc}}\, w$ by $\mathtt{e\text{-}in}_{\mathsf{suc}}$
1120 $\mathtt{in}_{\mathsf{suc}}\, w \in \mathtt{in}_{\mathsf{suc}}\, ([\![T_s[N/u; (T_{\mathtt{Rec}}\, N)/X]\, \vec{M}]\!](\theta; \eta))$
1121 $\mathtt{in}_{\mathsf{suc}}\, w \in [\![T_{\mathtt{Rec}}\, (\mathsf{suc}\, N)\, \vec{M}]\!](\theta; \eta)$ by Lemma 31
1122

1123 **Case:**

$$\frac{\Delta; \Xi; \Gamma \vdash s \Rightarrow T_{\mathtt{Rec}}\, 0\, \vec{M}}{\Delta; \Xi; \Gamma \vdash \mathtt{out}_0\, s \Rightarrow T_z\, \vec{M}}\ \mathtt{t\text{-}out}_0$$ 1124

1125 $s[\theta; \sigma] \Downarrow w \in [\![T_{\mathtt{Rec}}\, 0\, \vec{M}]\!](\theta; \eta)$ by I.H.
1126 $w = \mathtt{in}_0\, v$ for some $v \in ([\![T_z\, \vec{M}]\!](\theta; \eta))$ by Lemma 31
1127 $(\mathtt{out}_0\, s)[\theta; \sigma] \Downarrow v$ by $\mathtt{e\text{-}out}_0$
1128

1129 **Case:**

$$\frac{\Delta; \Xi; \Gamma \vdash s \Rightarrow T_{\mathtt{Rec}}\, (\mathsf{suc}\, N)\, \vec{M}}{\Delta; \Xi; \Gamma \vdash \mathtt{out}_{\mathsf{suc}}\, s \Rightarrow T_s[N/u; (T_{\mathtt{Rec}}\, N)/X]\, \vec{M}}\ \mathtt{t\text{-}out}_{\mathsf{suc}}$$ 1130

1131 $s[\theta; \sigma] \Downarrow w \in [\![T_{\mathtt{Rec}}\, (\mathsf{suc}\, N)\, \vec{M}]\!](\theta; \eta)$ by I.H.
1132 $w = \mathtt{in}_{\mathsf{suc}}\, v$ for some $v \in [\![T_s[N/u; (T_{\mathtt{Rec}}\, N)/X]\, \vec{M}]\!](\theta; \eta)$ by Lemma 31
1133 $(\mathtt{out}_{\mathsf{suc}}\, s)[\theta; \sigma] \Downarrow v$ by $\mathtt{e\text{-}out}_{\mathsf{suc}}$
1134

1135 **Case:**

$$\frac{\Delta; \Xi; \Gamma \vdash t_z \Leftarrow S[0/u] \quad \Delta, u{:}\,\mathsf{nat}; \Xi; \Gamma, x{:}S \vdash t_s \Leftarrow S[\mathsf{suc}\, u/u]}{\Delta; \Xi; \Gamma \vdash \mathtt{ind}\, t_z\, (u,\, x.\, t_s) \Leftarrow (u{:}\,\mathsf{nat});\, 1 \to S}\ \mathtt{t\text{-}ind}$$ 1136

1137 Let c be the closure $(\mathtt{ind}\, t_z\, (u,\, x.\, t_s))[\theta; \sigma]$.
1138 $(\lambda\, \vec{u}, x.\, s)[\theta; \sigma] \Downarrow c$ by $\mathtt{e\text{-}ind}$
1139 Suffices to show $c \in [\![(u{:}\,\mathsf{nat});\, 1 \to S]\!](\theta; \eta)$, i.e.
1140 $\forall N \in [\![\mathsf{nat}]\!].\ c \cdot N\, \langle\rangle \Downarrow w \in [\![S]\!](\theta, N/u; \eta)$.
1141 Proceed by induction on N.
1142 Base case: $N = 0$.

1143 $t_z[\theta; \sigma] \Downarrow w \in [\![S[0/u]]\!](\theta; \eta)$ by I.H.
1144 $w \in [\![S]\!]((\mathsf{id}_\Delta, 0/u)[\theta]; \eta)$ by Lemma 21
1145 $w \in [\![S]\!](\theta, 0/u; \eta)$ by Def. 2
1146 $c \cdot 0 \langle\rangle \Downarrow w$ by `e-app-ind`$_0$
1147 Step case: $N = \mathsf{suc}\, N'$ for some $N' \in [\![\mathsf{nat}]\!]$.
1148 $c \cdot N' \langle\rangle \Downarrow v \in [\![S]\!](\theta, N'/u; \eta)$ by inner I.H.
1149 Let $\theta' = \theta, N'/u$, so $\vdash \theta' : \Delta, u{:}\, \mathsf{nat}$.
1150 $\sigma \in [\![\Gamma]\!](\theta'; \eta)$ since $u \notin \mathrm{FV}(\Gamma)$
1151 $\sigma, v/x \in [\![\Gamma, x{:}S]\!](\theta'; \eta)$ by Def. 18
1152 $t_s[\theta'; \sigma, v/x] \Downarrow w \in [\![S[\mathsf{suc}\, u/u]]\!](\theta'; \eta)$ by I.H.
1153 $w \in [\![S]\!]((\mathsf{id}_\Delta, \mathsf{suc}\, u/u)[\theta']; \eta)$ by Lemma 21
1154 $w \in [\![S]\!](\theta', (\mathsf{suc}\, u)[\theta']/u; \eta)$ by Def. 2
1155 $w \in [\![S]\!](\theta', \mathsf{suc}\, N'/u; \eta)$ by θ' def.
1156 $w \in [\![S]\!](\theta, N/u; \eta)$ by N def. and overwriting u in θ'
1157 $c \cdot N \langle\rangle \Downarrow w$ by `e-app-ind`$_{\mathsf{suc}}$
1158
1159