Index-Stratified Types*

Rohan Jacob-Rao[1] and Brigitte Pientka[2]

1 **McGill University, Montreal, Canada**
`rjacob18@cs.mcgill.ca`
2 **McGill University, Montreal, Canada**
`bpientka@cs.mcgill.ca`

Abstract

We present an index-typed core language, called Tores, that supports writing inductive proofs as total recursive programs. It allows us to reason directly by induction on index terms and by Mendler-style recursion on indexed recursive types. In addition, Tores offers *stratified* types that are defined by well-founded recursion on index terms. Unlike indexed recursive types, stratified types can only be unfolded based on their index arguments. Both forms of recursively defined types circumvent any positivity restriction on type variables. This is particularly convenient for specifying recursive definitions such as reducibility candidates and logical relations. Our design pushes the expressiveness of indexed type systems while maintaining modularity and a simple metatheory. Our main technical result is a termination proof using a logical predicate semantics for our language.

1998 ACM Subject Classification D.3.1 Formal Definitions and Theory, D.3.3 Language Constructs and Features, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Indexed types, recursive types, logical relations

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.23

1 Introduction

Over the past two decades type systems for ordinary functional programming have grown closer to full dependently typed systems with the goal of bringing the benefits of dependent types to the main stream [Chen and Xi, 2005, Casinghino et al., 2014, Sjöberg, 2015]. The general mantra is to empower programmers and allow them to specify invariants and proofs about the runtime behaviour of their programs using types.

Indexed type systems [Zenger, 1997, Xi and Pfenning, 1999, Cave and Pientka, 2012, Thibodeau et al., 2016] balance the power of expressing rich runtime guarantees with decidable type checking. From a Curry-Howard isomorphism point of view, total programs using indexed recursive types correspond to inductive proofs within a first-order logic with equality and least fixed points [Momigliano and Tiu, 2004, Tiu and Momigliano, 2012, Baelde and Nadathur, 2012], hence also providing a logical foundation for reasoning by induction. This is directly exploited in the programming and proof environment Beluga [Pientka and Dunfield, 2010, Pientka and Cave, 2015] where we chose as an index language an extension of the logical framework LF [Harper et al., 1993, Nanevski et al., 2008] to specify formal systems and then implement proofs about them as total functions using pattern matching.

In this paper, we describe a core language, called Tores, that supports writing inductive proofs as total recursive programs about an index domain. It allows us to reason directly

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

by induction on index terms and by Mendler-style recursion on indexed recursive types. In addition, Tores offers stratified types that are defined by well-founded recursion on terms in the index language. This guarantees their well-foundedness while avoiding any positivity restriction. Unlike indexed recursive types, stratified types can only be unfolded according to their index arguments. Stratified types are particularly convenient when working with recursive but nonpositive definitions such as reducibility candidates in logical relations arguments.

Stratified types offer similar advantages to an indexed type theory as large eliminations do in a fully fledged dependently typed language, or as the use of rewriting to express type-level computation does in the proof theory described in [Baelde and Nadathur, 2012]. In contrast to these approaches, we avoid general type-level computation by offering only restricted unfolding of stratified types. This results in a simpler metatheoretic development. Type checking remains decidable and proving subject reduction is straightforward. Our main result in this paper is a termination proof for Tores using a logical predicate semantics. The combination of indexed recursive and stratified types in our language seems to provide a sweet-spot in leveraging the expressiveness of our logic with the simplicity of our metatheory.

We already use the combination of indexed recursive types and stratified types 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 inductive proofs.

2 Tores: A Language with Indexed Recursive and Stratified Types

2.1 Index Language

Our core language Tores is parametric over an index language. We follow the model given for example by Thibodeau et al. [2016], in that we provide conditions under which a language can be used as the index language for Tores, and thereafter ease the presentation using a simple example index language. For most of this paper we use the index language of natural numbers, which allows us to easily illustrate each feature which is required. In practice, we can consider using other index languages such as those of strings, types [Cheney and Hinze, 2003, Xi et al., 2003], or (contextual) LF [Cave and Pientka, 2012]. The conditions we require of the index language are the following: decidable type checking, decidable equality, decidable matching, and a notion of generating a covering set of index terms for a given index type [Pientka and Abel, 2015]. The last feature is needed to define an induction principle on index terms, which is used in the induction terms and stratified types in Tores.

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.

Index types	U	$:= \mathsf{nat}$
Index terms	M	$:= 0 \mid \mathsf{suc}\, M \mid u$
Index contexts	Δ	$:= \cdot \mid \Delta, u{:}U$
Index substitutions	Θ	$:= \cdot \mid \Theta, M/u$

Tores relies on typing for index terms which we give below for natural numbers (see 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. Note that the definitions of well-formed contexts and well-typed index substitutions are generic to the particular index terms and types.

$\boxed{\vdash \Delta\ \mathsf{ictx}}$ Index context Δ is well-formed

$$\frac{}{\vdash \cdot\ \mathsf{ictx}} \qquad \frac{\vdash \Delta\ \mathsf{ictx} \quad \Delta \vdash U\ \mathsf{itype}}{\vdash \Delta, u{:}U\ \mathsf{ictx}}$$

$\boxed{\Delta \vdash U\ \mathsf{itype}}$ Index type U is well-kinded

$$\frac{}{\Delta \vdash \mathsf{nat}\ \ \mathsf{itype}}$$

$\boxed{\Delta \vdash M : U}$ Index term M has index type U in index context Δ

$$\frac{u{:}U \in \Delta}{\Delta \vdash u : U} \qquad \frac{}{\Delta \vdash 0 : \mathsf{nat}} \qquad \frac{\Delta \vdash M : \mathsf{nat}}{\Delta \vdash \mathsf{suc}\, M : \mathsf{nat}}$$

$\boxed{\Delta \vdash M = N}$ Index term M is equal to N

$$\frac{}{\Delta \vdash M = M}$$

$\boxed{\Delta' \vdash \Theta : \Delta}$ Index substitution Θ maps index variables from Δ to Δ'

$$\frac{}{\Delta' \vdash \cdot : \cdot} \qquad \frac{\Delta' \vdash \Theta : \Delta \quad \Delta' \vdash M : U[\Theta]}{\Delta' \vdash \Theta, M/u : \Delta, u{:}U}$$

Figure 1 Index Language Structure

We support both a single index substitution operation, written $N[M/u]$, and the simultaneous substitution operation, written $M[\Theta]$. We omit their definitions here, and simply state the substitution principles we require.

▶ **Theorem 1** (Index substitution principles).

1.1. *If* $\Delta_1 \vdash M : U$ *and* $\Delta_1, u{:}U, \Delta_2 \vdash N : U'$ *then* $\Delta_1, \Delta_2[M/u] \vdash N[M/u] : U'[M/u]$.
1.2. *If* $\Delta' \vdash \Theta : \Delta$ *and* $\Delta \vdash M : U$ *then* $\Delta' \vdash M[\Theta] : U[\Theta]$.
1.3. *If* $\Delta' \vdash \Theta : \Delta$ *and* $\Delta \vdash M = N$ *then* $\Delta' \vdash M[\Theta] = N[\Theta]$.
1.4. *If* $\Delta \vdash M : U$ *and* $\Delta_1 \vdash \Theta_1 : \Delta$ *and* $\Delta_2 \vdash \Theta_2 : \Delta_1$ *then* $M[\Theta_1][\Theta_2] = M[\Theta_1[\Theta_2]]$.

Important to our treatment of equality elimination is the notion of *matching* index terms. During type checking we rely on a most general unifier (mgu) Θ for M and N, which is provided in the source term. During evaluation, the matching algorithm allows us to generate a correct environment of index values given the substitution Θ. Given two index terms M_1 and M_2, matching attempts to discover a substitution Θ such that $M_1[\Theta] = M_2$. This is written using a separate judgment $\Delta \vdash M \doteq N \searrow (\Delta' \mid \Theta)$. Index matching can be lifted generically to match index substitutions. We write $\Delta \vdash \Theta_1 \doteq \Theta_2 \searrow (\Delta' \mid \Theta)$ to say that matching discovered a substitution Θ such that $\Delta' \vdash \Theta_1[\Theta] = \Theta_2$. We omit the rules for matching due to space constraints, to be found in the appendix (A.1). Importantly, we state the theorems we require of index (substitution) matching.

▶ **Theorem 2** (Soundness of index substitution matching). *Suppose* $\Delta_1 \vdash \Theta_1 : \Delta$ *and* $\Delta_1 \vdash \Theta_1 \doteq \Theta_2 \searrow (\Delta_2 \mid \Theta)$. *Then* $\Delta_2 \vdash \Theta : \Delta_1$ *and* $\Delta_2 \vdash \Theta_1[\Theta] = \Theta_2$.

▶ **Theorem 3** (Completeness of index matching). *Suppose* $\vdash \theta : \Delta$ *and* $\vdash M[\theta] = N[\theta]$ *and* $\Delta' \vdash \Theta : \Delta$ *and* Θ *mgu.* $\Delta' \vdash M[\Theta] = N[\Theta]$. *Then* $\Delta' \vdash \Theta \doteq \theta \searrow (\cdot \mid \theta')$.

Finally, 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 a term and $\overrightarrow{M}$ is a spine, then $M_0, \overrightarrow{M}$ is a spine. Similarly if $u_0{:}U_0$ is a type declaration and $(\overrightarrow{u{:}U})$ is a spine of type declarations, then $(u_0{:}U_0, \overrightarrow{u{:}U})$ is a spine of type declarations. The use of spines is convenient in setting up the types and terms of Tores. We leave the full definition of spines to the appendix (A.1).

2.2 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. This will be convenient in defining our Mendler-style recursion operator. If the spine of type declarations is empty, then $(\cdot); T_1 \to T_2$ degenerates to the simple function space. Note that we can abstract over the index variables $\overrightarrow{u{:}U}$ in any given type T by $(\overrightarrow{u{:}U}); 1 \to T$. We also include indexed existential type $\Sigma u{:}U.\, T$ and importantly, a type of index equality $M = N$, which we use to express equality constraints on indices.

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

We model recursive and stratified types as types of kind $\Pi\overrightarrow{u{:}U}.\, *$. Both forms introduce type variables X, which we track in the type variable context Ξ. The indexed recursive type $\mu X{:}K.\, \Lambda\, \vec{u}.\, T$ binds multiple variables. We do not enforce a positivity condition on recursive types, as the typing rules for Mendler-recursion enforce termination without it. A stratified type is defined by well-founded recursion on an index term. In essence, it is a total recursive type-level function that maps index objects to types. Here we rely on the specifics of the index language. For the index type nat, we can define a stratified type $T_{\mathtt{Rec}} = \mathtt{Rec}_K\, (0 \mapsto T_0 \mid \mathsf{suc}\, u,\, X \mapsto T_s)$. 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 $T_s[M/u, T_{\mathtt{Rec}}\, M/X]$, where we substitute the smaller index and type for the index variable and type variable respectively.

▶ **Example 4.** To illustrate indexed recursive types and stratified types, we consider vectors, i.e. lists that are indexed by their length, where we use A for the type of elements. Vectors are of kind $\Pi n{:}\,\mathsf{nat}\,.*$. We omit the kind annotation for better readability in the subsequent type definitions. We first define vectors using an indexed recursive type, explicit equality and an existential type: $\mathsf{Vec}_\mu \equiv \mu V.\, \Lambda\, n.\, n = 0 + \Sigma m{:}\,\mathsf{nat}\,.\, n = \mathsf{suc}\, m \times (A \times V\, m)$

Vectors can also 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, there are of course types that have to be defined either as a recursive or a stratified type, as not all type definitions directly recurse on an index.

▶ **Example 5.** To illustrate the usefulness of stratified types, we give here the definition of reducibility candidates for simply-typed lambda terms. In this example, our index domain consists of index objects modelling simple types, i.e `base` and $\texttt{arr}\ A\ B$, of index type `tp` and index objects modelling lambda terms, i.e. `c`, $\texttt{lam}\ x.M$, and $\texttt{app}\ M\ N$, of index type `tm`.

We can then define reducibility candidates as a stratified type of kind $\Pi a{:}\texttt{tp}.\Pi m{:}\texttt{tm}.*$. We rely on a recursive type definition of $\texttt{Halt}\ m$ (omitted for space) to describe that the term m steps to a value.

$$\begin{array}{l}\texttt{Rec}\ (\ \texttt{base} \qquad\qquad \mapsto \Lambda m.\texttt{Halt}\ m \\ \quad \mid \texttt{arr}\ a\ b, R_a, R_b \mapsto \Lambda m.\texttt{Halt}\ m \times (n{:}\texttt{tm});\ R_a\ n \to R_b\ (\texttt{app}\ m\ n)\ )\end{array}$$

2.3 Terms

Tores contains many common constructs found in functional programming languages, such as unit, pairs, case expressions, injections and fold operators to construct data. We focus here on the less standard constructs: our definition and use of functions, witnesses for equality, support for recursion on data structures defined by a recursive type and support for induction on index objects.

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

Recall that in our type language, we combine the dependent function space with the simple function type in $(\overrightarrow{u{:}U});\ T_1 \to T_2$. Similarly, we combine abstraction over index variables $\vec{u}$ and term variable x in our function definition, written as $\lambda\, \vec{u}, x.\, t$. The corresponding application form is written $t\, \vec{M}\, s$. 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 witness $\texttt{eq}\, s\, \texttt{with}\, (\Delta.\Theta \mapsto t)$ uses an equality proof s for $M = N$ together with a substitution Θ, which is the most general substitution for index variables in M and N such that $\Delta \vdash M[\Theta] = N[\Theta]$, and continues with the body t in the new index context Δ. It may also be the case that the equality witness s stands for false, in which case we have reached a contradiction and abort using the term $\texttt{eq\_abort}\, s$. Both forms are necessary to make use of equality constraints that arise from indexed type definitions and show that some cases are impossible.

Let us explain the introduction and elimination forms for recursive and stratified types. The "fold" syntax $\texttt{in}_\mu$ introduces terms of recursive types, and $\texttt{in}_l$ introduces terms of stratified types. Here l ranges over constructors in the index language, for example 0 and `suc`. An important difference is in how we eliminate recursive and stratified types.

We can analyze data defined by a recursive type using Mendler-style recursive programs $\texttt{rec}\, f.\, t$. This gives a powerful means of recursion in our language while still ensuring termination. Stratified types can only be unfolded using $\texttt{out}_l$ according to the index. To take full advantage of stratified types, we also allow programmers to use well-founded recursion over index objects, writing $\texttt{ind}\, t_0\, (u,\ f.\, t_s)$. Intuitively, if the index object is 0, then we pick the first branch and execute t_0; if the index object is $\texttt{suc}\, M$ then we pick the second branch instantiating u with M and allowing recursive calls f inside t_s.

▶ **Example 6.** As mentioned before, vectors can be defined using either indexed recursive types or stratified types. Which definition we choose will impact how we write programs that analyze vectors. To highlight the difference in the structure of programs, we will write a simple function that recursively copies a vector using Mendler-style recursion for Vec_μ (the indexed recursive type definition for vectors), and using induction on natural numbers to copy a vector Vec_S (the stratified type definition of vectors).

$$
\begin{array}{l}
\mathit{copy} : (n\colon \mathsf{nat});\ \mathsf{Vec}_\mu\, n \to \mathsf{Vec}_\mu\, n \equiv \mathtt{rec}\, f.\, \lambda\, n, v.\, \mathtt{case}\, v\, \mathtt{of} \\
\quad \mid \mathtt{in}_1\, z \ \mapsto\ \mathtt{in}_\mu\, (\mathtt{in}_1\, z) \\
\quad \mid \mathtt{in}_2\, s \ \mapsto\ \mathtt{unpack}\, s\, \mathtt{as}\, (m, p)\, \mathtt{in} \\
\qquad\qquad\ \ \mathtt{split}\, p\, \mathtt{as}\, \langle q, p' \rangle\, \mathtt{in} \\
\qquad\qquad\ \ \mathtt{split}\, p'\, \mathtt{as}\, \langle e, v' \rangle\, \mathtt{in} \\
\qquad\qquad\ \ \mathtt{in}_\mu\, (\mathtt{in}_2\, (\mathtt{pack}\, (m, \langle q, \langle e, f\, m\, v' \rangle \rangle))))
\end{array}
$$

Here we use recursion and case analysis of the input vector to reconstruct the vector as output. In the case where 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.

To contrast we show the same program that works directly by induction on natural numbers and then unfolding the stratified types definition of Vec_S. Here we first split on the natural number argument and then unfold the vector itself. Equality reasoning is silent.

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

2.4 Typing Rules

Our bidirectional typing rules in Fig. 2 are mostly standard. We check functions against $(\overrightarrow{u{:}U});\ S \to T$ by simply moving the index variables $\overrightarrow{u{:}U}$ into Δ and the term variable x of type S into Γ.

The introduction rule for equality simply falls back to showing that two index objects are equal in the index domain. The equality elimination rules are more interesting. We can use an equality constraint $M = N$ to establish T if, for the unique mgu Θ (for which we have that $M[\Theta] = N[\Theta]$), we can show $T[\Theta]$. The equality elimination term $\mathtt{eq}\, s\, \mathtt{with}\, (\Delta'.\Theta \mapsto t)$ captures the proof s that $M = N$. The index substitution Θ guarantees that $\Delta' \vdash M[\Theta] = N[\Theta]$ and the term t checks against $T[\Theta]$ in the new context Δ'. If M does not unify with N, then we simply conclude T as we have arrived at a contradiction, using $\mathtt{eq\_abort}\, s$ as a witness.

The equality elimination term $\mathtt{eq}\, s\, \mathtt{with}\, (\Delta'.\Theta \mapsto t)$ effectively switches the index context as in Pientka and Dunfield [2008] and Cave and Pientka [2012]. It also resembles the treatment of equality in proof theory [Tiu and Momigliano, 2012, McDowell and Miller, 2002, Schroeder-Heister, 1993]. In that line of work, equality elimination uses complete sets of unifiers instead of a most general unifier. In practice we can usually generate a unique mgu so generalizing in this direction does not seem crucial.

Both indexed recursive types and stratified types are introduced using injections ($\mathtt{in}_\mu$ and $\mathtt{in}_l$). However, as mentioned earlier, they use different elimination forms. For recursive types, we employ a Mendler-style typing rule for recursion $\mathtt{rec}\, f.\, t$. We generalize Mendler's original formulation [Mendler, 1988] to an indexed type system. The idea is to constrain the type of the function variable f so that it can only be applied to structurally smaller

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

$$\frac{x{:}T \in \Gamma}{\Delta; \Xi; \Gamma \vdash x \Rightarrow T} \quad \frac{}{\Delta; \Xi; \Gamma \vdash \langle\rangle \Leftarrow 1} \quad \frac{\Delta; \Xi; \Gamma \vdash t \Rightarrow T}{\Delta; \Xi; \Gamma \vdash t \Leftarrow T} \quad \frac{\Delta, \overrightarrow{u{:}U}; \Xi; \Gamma, x{:}S \vdash t \Leftarrow T}{\Delta; \Xi; \Gamma \vdash \lambda\, \vec{u}, x.\, t \Leftarrow (\overrightarrow{u{:}U});\ S \to T}$$

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

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

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

$$\frac{\Delta \vdash M = N}{\Delta; \Xi; \Gamma \vdash \mathtt{refl} \Leftarrow M = N} \quad \frac{\Delta; \Xi; \Gamma \vdash s \Rightarrow M = N \quad \Delta \vdash M \neq N}{\Delta; \Xi; \Gamma \vdash \mathtt{eq\_abort}\, s \Leftarrow T}$$

$$\frac{\Delta; \Xi; \Gamma \vdash s \Rightarrow M = N \quad \Delta' \vdash \Theta : \Delta \quad \Theta \text{ mgu. } \Delta' \vdash M[\Theta] = N[\Theta] \quad \Delta'; \Xi[\Theta]; \Gamma[\Theta] \vdash t \Leftarrow T[\Theta]}{\Delta; \Xi; \Gamma \vdash \mathtt{eq}\, s\, \mathtt{with}\, (\Delta'.\Theta \mapsto t) \Leftarrow T}$$

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

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

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

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

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

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

Figure 2 Typing Rules for Tores

data. This is achieved by declaring f of type $(\overrightarrow{u{:}U});\ X\ \vec{u} \to T$ in the corresponding typing rule. Here X can only be used to construct types exactly one constructor smaller than the recursive type, so the use of f is guaranteed to be well-founded.

In the induction rule we check $\mathtt{ind}\, t_0\, (u,\ f.\, t_s)$ against $(u{:}\,\mathsf{nat});\ 1 \to T$ by checking t_0 against $T[0/u]$ and checking t_s against $T[\mathsf{suc}\, u/u]$ given a function f of type T. Morally f stands for a function that can be applied to data constrained by the predecessor u. The unfolding rules for stratified types mirror the corresponding folding rules.

Finally, we can synthesize the type of the application $t\ \vec{M}\ s$ by first synthesizing the type $(\overrightarrow{u{:}U});\ S \to T$ for t, and then checking the arguments $\vec{M}$ and s against their corresponding types $(\overrightarrow{u{:}U})$ and $S[\overrightarrow{M/u}]$.

▶ **Theorem 7** (Decidability of type checking). *Type checking of terms is decidable.*

Proof. The typing rules are syntax directed, except for the conversion rule from checking to synthesis. The bidirectional system therefore specifies an algorithm: apply the rule

corresponding to the term syntax, and if no rule applies while checking then use the conversion rule and continue. ◀

2.5 Operational Semantics

We define a big-step operational semantics using index environment θ and term environment σ. Both environments provide closed values for the free variables that occur in a term. This approach aligns closely with both a practical implementation and our mathematical semantics.

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

Values consist of unit, pairs, injections, reflexivity, and closures. The typing rules for values and environments are straight-forward and provided in the appendix (A.4).

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 (lambda, recursion or induction) uniformly.

The evaluation rules for pairs, splits, injections, case expressions, pack and unpack terms are mostly straight-forward. Values evaluate to themselves. To evaluate a variable x in the environment $\theta; \sigma$, we look up the value x is bound to in σ. To evaluate an application $t\, \vec{M}\, s$ under environments θ and σ, we evaluate t in the environment $\theta; \sigma$ to a closure c, evaluate s to a value v, and then compute the result of applying the closure c to the index terms $\vec{M[\theta]}$ and the value v. If c stands for a function, we simply extend both environments and continue to evaluate the body. If c stands for a recursive function $\mathtt{rec}\, f.\, t$, we continue evaluating the body t in the extended environment $\sigma, (\mathtt{rec}\, f.\, t)[\theta; \sigma]/f$. If c stands for an induction on an index object and c is applied to 0, we evaluate the the first branch of the induction term. If c is applied to $\mathsf{suc}\, N$, we recursively compute the result of c applied to N before evaluating the body of the successor branch.

Last, we discuss the evaluation rules for $(\mathtt{eq}\, s\, \mathtt{with}\, (\Delta.\Theta \mapsto t))$ in the environments $\theta; \sigma$. We first evaluate s in the given environment to the value $\mathtt{refl}$, ensuring that during runtime $M[\theta]$ is indeed equal to $N[\theta]$. Note that statically, Θ already guaranteed that M and N are equal in Δ. We further note that both Θ and θ provide instantiations for index variables in the prior index context Δ_0, and there exists a grounding substitution θ' s.t. $\Theta[\theta'] = \theta$ which can be computed by matching. We then continue to evaluate the body t in the environments $\theta'; \sigma$. Note that index objects are computationally relevant in our core language, since we support induction on them. Finally, we can prove subject reduction for our language.

▶ **Theorem 8** (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* $w : T[\theta, \overrightarrow{N/u}]$*.*

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

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

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

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

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

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

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

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

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

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

Figure 3 Big-step Evaluation Rules

Proof. By mutual induction on the evaluation judgments. We include a few key cases in the appendix (A.4). ◀

3 Termination Proof

We now describe our main technical result: termination of evaluation. Our proof of termination uses the technique of logical relations, following the style of Tait [1967] and Girard [1972]. Most proofs left to the appendix (A.5) due to space limitation.

3.1 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 for example. Hence our interpretation for index types U must take into account an environment θ containing instantiations for index variables u. An index environment θ for an index context Δ is simply a grounding substitution $\vdash \theta : \Delta$.

▶ **Definition 9** (Interpretation of index types $[\![U]\!]$). $[\![U]\!](\theta) = \{M \mid \cdot \vdash M : U[\theta]\}$

The interpretation of an index type U under environment θ is the set of closed terms of type $U[\theta]$. Unlike for term-level types, we do not restrict our interpretation to contain only normal forms. This is because for index languages, any reducible expressions are reduced either during equality checking or, in the case of Contextual LF, during hereditary

substitutions [Nanevski et al., 2008]. We leave those details abstract, requiring only the properties in Section 2.1 such as decidable equality of index terms and type-preserving substitution. The interpretation lifts to index spines $(\overrightarrow{u{:}U})$ in a straight-forward manner.

With these definitions, the following lemma follows from the substitution principles of index terms (Theorem 1).

▶ **Lemma 10** (Interpretation of index substitution)**.**

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

3.2 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 2^Ω. The interpretation is defined inductively on the structure of kinds.

▶ **Definition 11** (Interpretation of kinds $[\![K]\!]$)**.**

$$\begin{array}{lcl} [\![*]\!](\theta) & = & 2^\Omega \\ [\![\Pi u{:}U.\, K]\!](\theta) & = & \{\mathcal{C} \mid \forall M \in [\![U]\!](\theta).\, \mathcal{C}(M) \in [\![K]\!](\theta, M/u)\} \end{array}$$

A key observation in our metatheory is that each $[\![K]\!](\theta)$ forms a *complete lattice*. Recall that a lattice is a partially ordered set with additional meet $\bigwedge$ and join $\bigvee$ operations. A lattice is complete if every subset has a meet (greatest lower bound) and a join (least upper bound). In our case, $[\![*]\!](\theta) = 2^\Omega$ is a complete lattice under the subset ordering, with meet and join given by intersection and union respectively. Further, we can induce a complete lattice structure on interpretations of kinds $K = \Pi u{:}U.\, K'$ by lifting the lattice operations pointwise. Precisely, we define

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

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. Our definition relies on the existence of arbitrary meets, as it uses the meet over an impredicatively defined subset of $\mathcal{L}$.

▶ **Definition 12** (Mendler-style pre-fixed point)**.** 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 $\boldsymbol{\mu}_{\mathcal{L}}\mathcal{F} = \bigwedge\{\mathcal{C} \in \mathcal{L} \mid \forall \mathcal{X} \in \mathcal{L}.\, \mathcal{X} \leq \mathcal{C} \implies \mathcal{F}(\mathcal{X}) \leq \mathcal{C}\}$.

3.3 Interpretation of Types

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

▶ **Definition 13** (Semantic function space)**.** For a spine interpretation $\vec{\mathcal{U}}$ and functions $\mathcal{A}, \mathcal{B} : \vec{\mathcal{U}} \to 2^\Omega$, define $\vec{\mathcal{U}}, \mathcal{A} \to \mathcal{B} = \{c \mid \forall \vec{M} \in \vec{\mathcal{U}}.\, \forall v \in \mathcal{A}(\vec{M}).\, c \cdot \vec{M}\, v \Downarrow w \in \mathcal{B}(\vec{M})\}$.

It will also be convenient to lift term-level $\mathtt{in}$ tags to the level of sets and functions in the lattice $[\![K]\!](\theta)$. We define the lifted tags $\mathtt{in}^* : [\![K]\!](\theta) \to [\![K]\!](\theta)$ inductively on K. If $\mathcal{V} \in [\![*]\!](\theta) = 2^\Omega$ 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 $(\mathtt{in}^*\,\mathcal{C})(M) = \mathtt{in}^*\,(\mathcal{C}(M))$ for all $M \in [\![U]\!](\theta)$. Essentially, the $\mathtt{in}^*$ function attaches a tag to every element in the set produced after the index arguments are received.

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

▶ **Definition 14** (Interpretation of type variable contexts $[\![\Xi]\!]$).

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

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

▶ **Definition 15** (Interpretation of types and constructors).

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

where

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

The interpretation of the indexed function type $[\![(\overrightarrow{u{:}U});\, T_1 \to T_2]\!](\theta;\eta)$ contains closures which, when applied to values in the appropriate input sets, evaluate to values in the appropriate output set. The interpretation of the equality type $[\![M = N]\!](\theta;\eta)$ is the set $\{\mathtt{refl}\}$ if $\vdash M[\theta] = N[\theta]$ and the empty set otherwise. The interpretation of a recursive type is the pre-fixed point of the function obtained from the underlying type expression. Finally, interpretation of a stratified type built from $\mathtt{Rec}$ relies on an analogous semantic operator **Rec**. It is defined by primitive recursion on the index argument, returning the first argument in the base case and calling itself recursively in the step case. Note that the definition of **Rec** is specific to the index type it recurses over.

The last form of interpretation we need is for typing contexts Γ, describing well-formed term-level environments σ.

▶ **Definition 16** (Interpretation of typing contexts).

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

3.4 Proof

We now sketch our proof using some key lemmas. The next two lemmas are key for reasoning about recursive types and Mendler-style recursion. The proofs generalize the set-theoretic analogues shown by Jacob-Rao et al. [2016] to the setting of a complete lattice.

▶ **Lemma 17** (Soundness of pre-fixed point). *Suppose $\mathcal{L}$ is a complete lattice, $\mathcal{F} : \mathcal{L} \to \mathcal{L}$ and $\boldsymbol{\mu}$ is as in Def. 12. Then $\mathcal{F}(\boldsymbol{\mu}\mathcal{F}) \leq \boldsymbol{\mu}\mathcal{F}$.*

▶ **Lemma 18** (Function space from a pre-fixed point). *Let $\mathcal{L} = \vec{\mathcal{U}} \to 2^{\Omega}$ and $\mathcal{B} \in \mathcal{L}$ and $\mathcal{F} : \mathcal{L} \to \mathcal{L}$. 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 $c \in \vec{\mathcal{U}}, \boldsymbol{\mu}\mathcal{F} \to \mathcal{B}$.*

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

▶ **Lemma 19** (Type-level substitution associates with interpretation).
Suppose $\Delta; \Xi \vdash T \Leftarrow K$ or $\Delta; \Xi \vdash T \Rightarrow K$, and $\vdash \theta : \Delta'$ and $\eta \in [\![\Xi']\!](\theta)$.

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

Proof. By induction on the structure of T. ◀

▶ **Theorem 20** (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 $v \in [\![T]\!](\theta;\eta)$.*

Proof. By induction on the typing derivation. Note that the cases involving stratified types and induction over indices are specific to the particular index language (and assume an induction principle over closed index types). The rest of the proof is generic, only assuming the properties in Section 2.1. ◀

4 Related Work

From a proof-theoretic point of view, our core language is closely related to [Momigliano and Tiu, 2004, Tiu and Momigliano, 2012] where the authors also consider a first-order logic with equality on index objects and least and greatest fix points, but omit stratified definitions. Least and greatest fixed point definitions must satisfy a functoriality condition. The authors prove consistency of their logic using a cut-elimination argument that is fairly intricate. This work does not support stratified definitions nor induction over index objects. Tiu [2012] also proposes a logic with stratified definitions, but lacks general support for least and greatest fix points. He again proves the consistency of the logic using a cut-elimination argument.

Baelde and Nadathur [2012] present a first-order logic with least and greatest fixed points with equality that corresponds to a core language with (co)iteration. To support computation

on indices the authors support general rewriting on propositions and index terms following the approach in deduction modulo [Dowek and Werner, 2003]. While rewriting can be used to model stratified types, this is not necessarily desirable, as it leads to a more complex metatheory. From a programming point of view, making the unfolding of stratified types explicit leads to a simple decidable type checking procedure.

In languages with full dependent types such as Coq [Bertot and Castéran, 2004] or Agda [Norell, 2007], stratified types can be easily emulated using *large eliminations*, which are functions returning types. However, the metatheory of full dependent types is significantly more complicated than indexed types. Stratified types achieve the same goal as large eliminations for the examples we are interested in, but within an indexed type system with a much simpler metatheory.

In index-typed setting, Mendler-style recursion schemes have been investigated by Ahn [2014], where he describes an extension of System Fω with indexed equirecursive types. In his presentation, the recursion schemes are introduced via Church encodings. We believe our treatment of Mendler-style recursion for indexed recursive types is more direct, and we emphasize the combination with stratified types.

Finally, we point out a relationship to work on intensional/flexible type analysis, which aims to support runtime analysis of types. The original system by Harper and Morrisett [1995] includes a `Typerec` operator that defines types by recursion on a type argument. The LX system by Crary and Weirich [1999] also has a rich type constructor language including means for defining types by primitive (in fact, Mendler-style) recursion. Unlike in those systems, our `Rec` operator defines types by recursion on index terms instead of types. Fundamentally, the goals of intensional type analysis are vastly different from our own. We focus on providing a type-theoretic foundation for programming inductive proofs, whereas the above authors are concerned with efficient compilation of polymorphic programs.

5 Conclusion and Future Work

We presented a core language Tores extending an indexed type system with recursive types and stratified types. We argued that Tores provides a sound and powerful foundation for programming inductive proofs, in particular those involving logical relations. This power comes from the induction principles on recursive types given by Mendler-style 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).

A natural next step is to extend our language with Mendler-style corecursion. This would enable coinductive proofs and provide a core language into which we can compile proofs using corecursion and copatterns [Thibodeau et al., 2016].

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

References

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.

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

Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.

Chris Casinghino, Vilhelm Sjöberg, and Stephanie Weirich. Combining proofs and programs in a dependently typed language. In *41st Symp. on Principles of Programming Languages (POPL '14)*, pages 33–46. ACM Press, 2014.

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

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

Chiyan Chen and Hongwei Xi. Combining programming with theorem proving. In *10th Int'l Conf. on Functional Programming*, pages 66–77, 2005.

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

Karl Crary and Stephanie Weirich. Flexible type analysis. In *4th Int'l Conf. on Functional Programming (ICFP'99)*, pages 233–248. ACM, 1999.

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

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

Robert Harper and Greg Morrisett. Compiling polymorphism using intensional type analysis. In *22nd Symp. on Principles of Programming Languages (POPL'95)*, pages 130–141. ACM, 1995.

Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, January 1993.

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, 1988. AAI8804634.

Alberto Momigliano and Alwen Fernanto Tiu. Types for proofs and programs (TYPES'03). In *Induction and Co-induction in Sequent Calculus*, 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.

Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, September 2007. Technical Report 33D.

Brigitte Pientka. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *35th Symp. on Principles of Programming Languages (POPL'08)*, pages 371–382. ACM Press, 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 Andrew Cave. Inductive Beluga: Programming Proofs (System Description). In *25th Int'l Conf. on Automated Deduction (CADE-25)*, Lecture Notes in Computer Science (LNCS 9195), pages 272–281. Springer, 2015.

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

Brigitte Pientka and Joshua Dunfield. Beluga: a framework for programming and reasoning with deductive systems (System Description). In *5th Int'l Joint Conf. on Automated Reasoning (IJCAR'10)*, Lecture Notes in Artificial Intelligence (LNAI 6173), pages 15–21. Springer, 2010.

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

Vilhem Sjöberg. *A Dependently Typed Language with Nontermination*. PhD thesis, University of Pennsylvania, 2015.

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 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.

Hongwei Xi and Frank Pfenning. Dependent types in practical programming. In *26th Symp. on Principles of Programming Languages (POPL'99)*, pages 214–227. ACM Press, 1999.

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

Christoph Zenger. Indexed types. *Theoretical Computer Science*, 187(1-2):147–165, 1997.

A Appendix

A.1 Index Spines and Substitutions

This section contains the definitions omitted from Section 2.1. We start with the definition of well-kinded spines of index types and well-typed spines of index terms, 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

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

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

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

Next we illustrate single index substitutions using the definition for natural numbers below.

▶ **Definition 21** (Index substitution for natural numbers)**.**

$$\begin{aligned} 0[\Theta] &= 0 \\ (\mathsf{suc}\, M)[\theta] &= \mathsf{suc}\, M[\Theta] \\ u[\Theta] &= \Theta(u) \end{aligned}$$

Unlike single index substitutions, simultaneous substitutions can be developed generically. Typing for simultaneous substitutions is given in Fig. 1. Composition of substitutions is defined below.

▶ **Definition 22** (Composition of index substitutions)**.** Suppose $\Delta_1 \vdash \Theta_1 : \Delta$ and $\Delta_2 \vdash \Theta_2 : \Delta_1$. Then $\Delta_2 \vdash \Theta_1[\Theta_2] : \Delta$ where

$$\begin{aligned} (\cdot)[\Theta_2] &= \Theta_2 \\ (\Theta_1', M/u)[\Theta_2] &= \Theta_1'[\Theta_2], M[\Theta_2]/u \end{aligned}$$

Next we provide the rules for index matching and index substitution matching. Again we use the domain of natural numbers in the single index version and leave the substitution version generic.

$\boxed{\Delta \vdash M \doteq N \searrow (\Delta' \mid \Theta)}$ 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

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

Finally we give the soundness property of single index matching, which was omitted from Section 2.1.

▶ **Theorem 23** (Soundness of index matching). *If* $\Delta \vdash M : U$ *and* $\Delta \vdash M \doteq N \searrow (\Delta' \mid \Theta)$ *then* $\Delta' \vdash \Theta : \Delta$ *and* $\Delta' \vdash M[\Theta] = N$.

A.2 Realistic Index Language

In this section we describe an index language of simple types and intrinsically typed lambda terms. This exemplifies a practical choice of index language for writing proofs such as those involving logical relations. While we do not attempt to model the full logical framework LF, we use the LF terminology. Specifically, we refer to the types that make up the syntactic categories in this example as (LF) types, refer to the objects that describe simple types and intrinsically well-typed lambda-terms as (LF) objects. Further, we refer to the the bound variable context as (LF) context.

(LF) Types	T	$:= \mathsf{tp} \mid \mathsf{tm}\ C$
(LF) Objects	A, B, M, N	$:= u \mid x \mid \mathsf{top} \mid \mathsf{arr}\ A\ B \mid \mathsf{unit} \mid \mathsf{lam}^{A,B}\ x.M \mid \mathsf{app}^{A,B}\ M\ N$
(LF) Contexts	Φ	$:= \cdot \mid \Phi, x{:}T$
Index types	U	$:= (\Phi.T)$
Index terms	C	$:= (\vec{x}.M)$
Index contexts	Δ	$:= \cdot \mid \Delta, u{:}U$
Index substitutions	Θ	$:= \cdot \mid \Theta, C/u$

$\boxed{\Delta; \Phi \vdash T : \mathsf{itype}}$ (LF) Type T is well-formed

$$\frac{}{\Delta; \Phi \vdash \mathsf{tp} : \mathsf{itype}} \qquad \frac{\Delta; \Phi \vdash A : \mathsf{itype}}{\Delta; \Phi \vdash \mathsf{tm}\ A : \mathsf{itype}}$$

$\boxed{\Delta; \Phi \vdash M : T}$ (LF) Term M has LF Type T

$$\frac{x{:}T \in \Phi}{\Delta; \Phi \vdash x : T} \qquad \frac{u : (\Phi.T) \in \Delta}{\Delta; \Phi \vdash u : T} \qquad \frac{}{\Delta; \Phi \vdash \mathsf{unit} : \mathsf{tm}\ \mathsf{top}}$$

$$\frac{\Delta; \Phi \vdash A : \mathsf{tp} \quad \Delta; \Phi \vdash B : \mathsf{tp} \quad \Delta; \Phi, x : \mathsf{tm}\ A \vdash M : \mathsf{tm}\ B}{\Delta; \Phi \vdash \mathsf{lam}^{A,B}\ x.M : \mathsf{tm}\ (\mathsf{arr}\ A\ B)}$$

$$\frac{\Delta; \Phi \vdash A : \mathsf{tp} \quad \Delta; \Phi \vdash B : \mathsf{tp} \quad \Delta; \Phi \vdash M : \mathsf{tm}\ (\mathsf{arr}\ A\ B) \quad \Delta; \Phi \vdash N : \mathsf{tm}\ A}{\Delta; \Phi \vdash \mathsf{app}^{A,B}\ M\ N : \mathsf{tm}\ B}$$

$\boxed{\Delta \vdash C : U}$ Index Term C has Index Type U in the Index Context Δ

$$\frac{\Delta; \Phi \vdash M : T}{\Delta \vdash (\vec{x}.M) : (\Phi.T)}$$

$\boxed{\Delta \vdash U\ \mathsf{itype}}$ Index Type U is well-kinded in the Index Context Δ

$$\frac{\Delta; \Phi \vdash T : \mathsf{itype}}{\Delta \vdash (\Phi.T) : \mathsf{itype}}$$

Given coverage and a way to generate a covering set for the index type $\Phi.T$ [Pientka and Abel, 2015], we can generate two recursors:

$$
\begin{array}{lll}
\mathtt{Rec}\,(\ \vec{x}.x_i & \mapsto & M_x \\
\quad \mid \vec{x}.\mathsf{unit} & \mapsto & M_u \\
\quad \mid \vec{x}.\mathsf{app}^{a,b}\ u\ v,\ X_u,\ X_v & \mapsto & M_a \\
\quad \mid \vec{x}.\mathsf{lam}^{a,b}\ y.u, X_u & \mapsto & M_l\ )
\end{array}
$$

$$
\begin{array}{lll}
\mathtt{Rec}\,(\ \mathsf{top} & \mapsto & M_u \\
\quad \mid \mathsf{arr}\ u\ v,\ X_u,\ X_v & \mapsto & M_a\ )
\end{array}
$$

Next, we define equality and matching of (LF) terms.

$\boxed{\Delta; \vec{x} \vdash M = N}$ (LF) Object M is equal to (LF) Object N

$$\frac{}{\Delta; \vec{x} \vdash M = M}$$

$\boxed{\Delta; \vec{x} \vdash M = N \searrow (\Delta' \mid \theta)}$ (LF) Object M matches (LF) Object N under θ

$$\frac{\Delta = \Delta_1, u : U, \Delta_2 \quad \Delta' = \Delta_1, \Delta_2[\vec{x}.M/u]}{\Delta; \vec{x} \vdash M = u \searrow (\Delta' \mid \mathsf{id}_{\Delta_1}, \vec{x}.M/u,\ \mathsf{id}_{\Delta_2})} \qquad \frac{}{\Delta; \vec{x} \vdash x_i = x_i \searrow (\Delta \mid \mathsf{id}_\Delta)}$$

$$\frac{\begin{array}{llllll}
\Delta\ ; \vec{x} & \vdash & A_1 & = & B_1 & \searrow (\Delta_1 \mid \theta_1) \\
\Delta_1; \vec{x} & \vdash & A_2[\theta_1] & = & B_2 & \searrow (\Delta_2 \mid \theta_2) \\
\Delta_2; \vec{x} & \vdash & M_1[\theta_1[\theta_2]] & = & N_2 & \searrow (\Delta_3 \mid \theta_3) \\
\Delta_3; \vec{x} & \vdash & M_2[\theta_1[\theta_2][\theta_3]] & = & N_2 & \searrow (\Delta_4 \mid \theta_4) \qquad \theta' = \theta_1[\theta_2][\theta_3][\theta_4]
\end{array}}{\Delta; \vec{x} \vdash \mathsf{app}^{A_1,A_2}\ M_1\ M_2 = \mathsf{app}^{B_1,B_2}\ N_1\ N_2 \searrow (\Delta_4 \mid \theta')}$$

$$\frac{\begin{array}{llllll}
\Delta\ ; \vec{x} & \vdash & A_1 & = & B_1 & \searrow (\Delta_1 \mid \theta_1) \\
\Delta_1; \vec{x} & \vdash & A_2[\theta_1] & = & B_2 & \searrow (\Delta_2 \mid \theta_2) \\
\Delta_2; \vec{x}, y & \vdash & M[\theta_1[\theta_1]] & = & N & \searrow (\Delta_3 \mid \theta_3) \qquad \theta' = \theta_1[\theta_2][\theta_3]
\end{array}}{\Delta; \vec{x} \vdash \mathsf{lam}^{A_1,A_2}\ y.M = \mathsf{lam}^{B_1,B_2}\ y.N \searrow (\Delta_3 \mid \theta')}$$

$$\frac{}{\Delta; \vec{x} \vdash \mathsf{top} = \mathsf{top} \searrow (\Delta \mid \mathsf{id}_\Delta)} \qquad \frac{\Delta\ ; \vec{x} \vdash A_1 = B_1 \searrow (\Delta_1 \mid \theta_1) \quad \Delta_1; \vec{x} \vdash A_2[\theta_1 = B_2 \searrow (\Delta_2 \mid \theta_2)}{\Delta; \vec{x} \vdash \mathsf{arr}\ A_1\ A_2 = \mathsf{arr}\ B_1\ B_2 \searrow (\Delta_2 \mid \theta_1[\theta_2])}$$

A.3 Kinding Rules

The kinding rules are shown below. We employ a bidirectional kinding system so that it is evident when kinds can be inferred and when kinding annotations are necessary. For example, we annotate `Rec` types with the result kind K so that we can infer it and check the constituent types against it.

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

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

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

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

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

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

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

A.4 Value Typing and Subject Reduction

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

$$\frac{\cdot \vdash \theta : \Delta \quad \sigma : \Gamma[\theta] \quad \Delta;\Gamma \vdash g \Leftarrow T}{(g)[\theta;\sigma] : T[\theta]} \qquad \frac{}{\langle\rangle : 1} \qquad \frac{\cdot \vdash M = N}{\mathtt{refl} : M = N}$$

$$\frac{v_1 : T_1 \quad v_2 : T_2}{\langle v_1, v_2\rangle : T_1 \times T_2} \qquad \frac{v : T_i}{\mathtt{in}_i\, v : T_1 + T_2} \qquad \frac{\cdot \vdash M : U \quad v : T[M/u]}{\mathtt{pack}\,(M, v) : \Sigma u{:}U.\, T} \qquad \frac{v : T[\overrightarrow{M/u}; \mu X{:}K.\, \Lambda\, \vec{u}.\, T/X]}{\mathtt{in}_\mu\, v : (\mu X{:}K.\, \Lambda\, \vec{u}.\, T)\, \vec{M}}$$

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

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

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

▶ **Theorem 24** (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* $w : T[\theta, \overrightarrow{N/u}]$.

Proof. By mutual induction on the evaluation judgment. We provide only a few key cases here.

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

$\vdash \theta : \Delta$ and $\vdash \sigma : \Gamma[\theta]$ by assumption
$\Delta; \cdot; \Gamma \vdash t\, \vec{M}\, s \Rightarrow t[\overrightarrow{M/u}]$ by assumption
$\Delta; \cdot; \Gamma \vdash t \Rightarrow (\overrightarrow{u{:}U});\ S \to T$
$\Delta \vdash \vec{M} : (\overrightarrow{u{:}U})$
$\Delta; \cdot; \Gamma \vdash s \Leftarrow S[\overrightarrow{M/u}]$ by rule `t-app`
$\vdash \vec{M}\theta : (\overrightarrow{u{:}U[\theta]})$
$\vdash c : ((\overrightarrow{u{:}U});\ S \to T)[\theta]$ by IH
$\vdash v : S[\theta, \overrightarrow{M[\theta]/u}]$ by IH
We observe that $c = g[\theta'; \sigma']$ where g is either $\lambda\, \vec{u}, x.\, t$, $\mathtt{rec}\, f.\, t$, or $\mathtt{ind}\, t_0\, (u,\ f.\, t_s)$
where $\Delta'; \Gamma' \vdash g : G$ s.t. $\vdash g[\theta'; \sigma'] : G[\theta']$ by def. of closures and canonical forms
$G[\theta'] = ((\overrightarrow{u{:}U});\ S \to T)[\theta]$ by previous lines
$G = (\overrightarrow{u{:}U'});\ S' \to T'$ where $\overrightarrow{U'[\theta']} = \overrightarrow{U[\theta]}$ by equality of types
and $S'[\theta', \overrightarrow{u/u}] = S[\theta, \overrightarrow{u/u}]$, and $T'[\theta', \overrightarrow{u/u}] = T[\theta, \overrightarrow{u/u}]$.
$\vdash v : S'[\theta', \overrightarrow{M/u}]$ by previous lines using equality
$\vdash \vec{M[\theta]} : \overrightarrow{u{:}U'[\theta']}$ by previous lines using equality
$\vdash w : T'[\theta', \overrightarrow{M[\theta]/u}]$ by IH 2
$\vdash w : T[\theta, \overrightarrow{M[\theta]/u}]$ by previous lines using equality

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

$\Delta; \cdot; \Gamma \vdash \lambda\, \vec{u}, x.\, t \Leftarrow (\overrightarrow{u{:}U});\ S \to T$ by assumption
$\Delta, \overrightarrow{u{:}U}; \cdot; \Gamma, x{:}S \vdash t \Leftarrow T$ by typing rule
$\vdash \theta : \Delta$ by assumption
$\vdash \theta, \overrightarrow{N/u} : \Delta, \overrightarrow{u{:}U}$ by typing rules for substitutions
$\vdash \sigma : \Gamma[\theta]$ by assumption
$\vdash \sigma, v : \Gamma[\theta, \overrightarrow{\overrightarrow{N}u}]$ since u's do not occur in Γ by weakening
$\vdash \sigma, v : \Gamma[\theta, \overrightarrow{\overrightarrow{N}u}], x : S[\theta, \overrightarrow{N/u}]$ by typing rules for substitutions
$\vdash \sigma, v : (\Gamma, x : S)[\theta, \overrightarrow{N/u}]$ by rules for substitutions
$\vdash w : T[\theta, \overrightarrow{N/u}]$ by IH

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

$\Delta; \cdot; \Gamma \vdash \mathtt{rec}\, f.\, t \Leftarrow (\overrightarrow{u{:}U});\ (\mu X{:}K.\, \Lambda\, \overrightarrow{v}.\, S)\, \vec{u} \to T$ by assumption
$\Delta; X{:}K; \Gamma, f{:}((\overrightarrow{u{:}U});\ X\vec{u} \to T) \vdash t \Leftarrow (\overrightarrow{u{:}U});\ S[\overrightarrow{u/v}] \to T$ by typing rule
$\Delta; \Gamma, f{:}((\overrightarrow{u{:}U});\ (\mu X{:}K.\, \Lambda\, \overrightarrow{v}.\, S)\ \vec{u} \to T) \vdash t \Leftarrow (\overrightarrow{u{:}U});\ S[\overrightarrow{u/v}; \mu X{:}K.\, \Lambda\, \overrightarrow{v}.\, S/X] \to T$ by substitution
$\vdash \mathtt{in}_\mu\, v' : ((\mu X{:}K.\, \Lambda\, \overrightarrow{v}.\, S)\ \vec{u}))[\theta, \overrightarrow{N/u}]$ by assumption
$\vdash v' : S[\theta, \overrightarrow{N/u};\ \mu X{:}K.\, \Lambda\, \overrightarrow{v}.\, S/X]$ by typing rule
$\vdash \theta : \Delta$ by assumption
$\vdash \sigma, (\mathtt{rec}\, f.\, t)[\theta; \sigma]/f : \Gamma[\theta], f{:}((\overrightarrow{u{:}U});\ (\mu X{:}K.\, \Lambda\, \overrightarrow{v}.\, S)\ \vec{u} \to T)[\theta]$ by substitution typing
$\vdash \sigma, (\mathtt{rec}\, f.\, t)[\theta; \sigma]/f : (\Gamma, f{:}((\overrightarrow{u{:}U});\ (\mu X{:}K.\, \Lambda\, \overrightarrow{v}.\, S)\ \vec{u} \to T))[\theta]$ by substitution def.
$\vdash c : ((\overrightarrow{u{:}U});\ S[\overrightarrow{u/v}; \mu X{:}K.\, \Lambda\, \overrightarrow{v}.\, S/X] \to T)[\theta]$ by IH

We observe that $c = g[\theta'; \sigma']$ where $\Delta'; \Gamma' \vdash g : G$ s.t. $\vdash g[\theta'; \sigma'] : G[\theta']$ by def. of closures
$\vdash \theta' : \Delta'$ and $\vdash \sigma' : \Gamma'[\theta[']$ by def. of closure
$G[\theta'] = ((\overrightarrow{u{:}U}); S \to T)[\theta]$ by previous lines
$G = (\overrightarrow{u{:}U'}); S' \to T'$ where $\overrightarrow{U'[\theta']} = \overrightarrow{U[\theta]}$ by equality of types
and $S'[\theta', \overrightarrow{u/u}] = S[\theta, \overrightarrow{u/u}\ ;\ \mu X{:}K.\, \Lambda\, \overrightarrow{v}.\, S/X]$, and $T'[\theta', \overrightarrow{u/u}] = T[\theta, \overrightarrow{u/u}]$.
$\Delta'; \Gamma' \vdash g : (\overrightarrow{u{:}U'}); S' \to T'$ by previous lines
$\vdash \vec{N} : \overrightarrow{U'[\theta']}$ by equality and assumption
$\vdash v' : S'[\theta', \overrightarrow{N/u}]$ by equality and previous lines
$\vdash w : T'[\theta', \overrightarrow{N/u}]$ by IH
$\vdash w : T[\theta, \overrightarrow{N/u}]$ by equality and previous lines
◀

A.5 Termination Proof

▶ **Lemma 25** (Soundness of pre-fixed point). *Suppose $\mathcal{L}$ is a complete lattice, $\mathcal{F} : \mathcal{L} \to \mathcal{L}$ and $\boldsymbol{\mu}$ is as in Def. 12. Then $\mathcal{F}(\boldsymbol{\mu}\mathcal{F}) \leq \boldsymbol{\mu}\mathcal{F}$.*

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 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 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 required. (If this argument appears to be circular, that's because it is! It cleverly exploits our impredicative definition of $\boldsymbol{\mu}$.) ◀

▶ **Lemma 26** (Function space from a pre-fixed point). *Let $\mathcal{L} = \vec{\mathcal{U}} \to 2^{\Omega}$ and $\mathcal{B} \in \mathcal{L}$ and $\mathcal{F} : \mathcal{L} \to \mathcal{L}$. 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 $c \in \vec{\mathcal{U}}, \boldsymbol{\mu}\mathcal{F} \to \mathcal{B}$.*

Proof. We will reframe the lemma statement using a new piece of notation. For a closure $c \in \Omega$ and $\mathcal{B} \in \vec{\mathcal{U}} \to 2^{\Omega} = \mathcal{L}$, define $\mathcal{E}_c(\mathcal{B}) \in \mathcal{L}$ by $\mathcal{E}_c(\mathcal{B})(\vec{M}) = \{v \in \Omega \mid c \cdot \vec{M}\, v \Downarrow w \in \mathcal{B}(\vec{M})\}$. 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 can now rewrite the lemma as the following: 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 $\boldsymbol{\mu}\mathcal{F} \leq \mathcal{E}_c(\mathcal{B})$.

To prove this, assume the premise. 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 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 $\mathcal{C} \in \mathcal{S}$ 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 $\mathcal{E}_c(\mathcal{B}) \leq \mathcal{E}_c(\mathcal{B})$, so we are done. (This proof again makes use of impredicativity in the definition of $\boldsymbol{\mu}$.) ◀

▶ **Lemma 27** (Recursive type contains unfolding).
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 $\vdash \theta : \Delta$ and $\eta \in [\![\Xi]\!](\theta)$. Then $\mathtt{in}_{\mu}\, [\![S[\overrightarrow{M/u}; R/X]]\!](\theta; \eta) \subseteq [\![R\, \vec{M}]\!](\theta; \eta)$.

Proof. Let $\mathcal{L} = [\![K]\!](\theta)$.
Define $\mathcal{F} : \mathcal{L} \to \mathcal{L}$ by $\mathcal{F}(\mathcal{X}) = \vec{N} \mapsto \mathtt{in}_{\mu}\, [\![S]\!](\theta, \overrightarrow{N/u}; \eta, \mathcal{X}/X)$.
Then $[\![R]\!](\theta; \eta)$
$= \boldsymbol{\mu}(\mathcal{X} \mapsto \mathtt{in}^*_{\mu}\, [\![\Lambda\, \vec{u}.\, S]\!](\theta; \eta, \mathcal{X}/X))$ by $[\![\mu X.\, T]\!]$ def.
$= \boldsymbol{\mu}(\mathcal{X} \mapsto \vec{N} \mapsto \mathtt{in}_{\mu}\, [\![S]\!](\theta, \overrightarrow{N/u}; \eta, \mathcal{X}/X))$ by $[\![\Lambda\, \vec{u}.\, T]\!]$ def.
$= \boldsymbol{\mu}\mathcal{F}$ by $\mathcal{F}$ def.
$\mathcal{F}([\![R]\!](\theta; \eta)) \leq_{\mathcal{L}} [\![R]\!](\theta; \eta)$ by Lemma 25
Now $\mathtt{in}_{\mu}\, [\![S[\overrightarrow{M/u}; R/X]]\!](\theta; \eta)$

$= \mathtt{in}_\mu\, [\![S]\!]((\mathsf{id}_\Delta, \overrightarrow{M/u})[\theta]; \eta, [\![R]\!](\eta)/X)$ by Lemma 19
$= \mathtt{in}_\mu\, [\![S]\!](\theta, \overrightarrow{M[\theta]/u}; \eta, [\![R]\!](\theta;\eta)/X)$ by Def. 22
$= \mathcal{F}([\![R]\!](\theta;\eta))(\overrightarrow{M[\theta]})$ by $\mathcal{F}$ def.
$\subseteq [\![R]\!](\theta;\eta)(\overrightarrow{M[\theta]})$ since $\mathcal{F}([\![R]\!](\theta;\eta)) \leq_{\mathcal{L}} [\![R]\!](\theta;\eta)$
$= [\![R\,\vec{M}]\!](\theta;\eta)$ by $[\![T]\!]$ def.

◀

▶ **Lemma 28** (Backward closure). *Let t be a term, θ and σ environments, and $\mathcal{A}, \mathcal{B} \in \vec{\mathcal{U}} \to 2^\Omega$. If $t[\theta;\sigma, (\mathtt{rec}\, f.\, t)[\theta;\sigma]/f] \Downarrow c' \in \vec{\mathcal{U}}, \mathcal{A} \to \mathcal{B}$, then $(\mathtt{rec}\, f.\, t)[\theta;\sigma] \in \vec{\mathcal{U}}, \mathit{in}^*_\mu\, \mathcal{A} \to \mathcal{B}$.*

Proof. Let $c = (\mathtt{rec}\, f.\, t)[\theta;\sigma]$.
Suppose $\vec{M} \in \vec{\mathcal{U}}$ and $v' \in (\mathtt{in}^*_\mu\, \mathcal{A})(\vec{M})$.
$v \in \mathtt{in}_\mu\, \mathcal{A}(\vec{M})$ by $\mathtt{in}^*$ def.
$v' = \mathtt{in}_\mu\, v$ where $v \in \mathcal{A}(\vec{M})$
$t[\theta;\sigma,c/f] \Downarrow c' \in \vec{\mathcal{U}}, \mathcal{A} \to \mathcal{B}$ assumption of lemma
$c' \cdot \vec{M}\, v \Downarrow w \in \mathcal{B}(\vec{M})$ by $\vec{\mathcal{U}}, \mathcal{A} \to \mathcal{B}$ def.
$c \cdot \vec{M}\, (\mathtt{in}_\mu\, v) \Downarrow w \in \mathcal{B}(\vec{M})$ by `e-app-rec`
$c \cdot \vec{M}\, v' \Downarrow w \in \mathcal{B}(\vec{M})$ since $v' = \mathtt{in}_\mu\, v$
$c \in \vec{\mathcal{U}}, \mathtt{in}^*_\mu\, \mathcal{A} \to \mathcal{B}$ by $\vec{\mathcal{U}}, \mathcal{A} \to \mathcal{B}$ def.

◀

▶ **Lemma 29** (Stratified types equivalent to unfolding).
Let $T_{\mathtt{Rec}} \equiv \mathtt{Rec}_K\, (0 \mapsto T_z \mid \mathsf{suc}\, n,\ X \mapsto T_s)$ where $K = \Pi n{:}\,\mathsf{nat}\,.\, \Pi\overrightarrow{u{:}U}.\, *$ and $\Delta; \Xi \vdash T_{\mathtt{Rec}} \Rightarrow K$, 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. $[\![T_{\mathtt{Rec}}\, 0\, \vec{M}]\!](\theta;\eta) = \mathit{in}_0\, ([\![T_z\, \vec{M}]\!](\theta;\eta))$ *and*
2. $[\![T_{\mathtt{Rec}}\, (\mathsf{suc}\, N)\, \vec{M}]\!](\theta;\eta) = \mathit{in}_{\mathsf{suc}}\, ([\![T_s[N/n; (T_{\mathtt{Rec}}\, N)/X]\, \vec{M}]\!](\theta;\eta))$.

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

1. $[\![T_{\mathtt{Rec}}\, 0\, \vec{M}]\!](\theta;\eta)$
   $= [\![T_{\mathtt{Rec}}]\!](\theta;\eta)(0)(\overrightarrow{M[\theta]})$ by $[\![T\, \vec{M}]\!]$ def.
   $= \mathbf{Rec}\, \mathcal{C}\, \mathcal{F}\, 0\, \overrightarrow{M[\theta]}$ by $[\![T_{\mathtt{Rec}}]\!]$ def.
   $= \mathcal{C}\, \overrightarrow{M[\theta]}$ by **Rec** def
   $= (\mathtt{in}^*_0\, [\![T_z]\!](\theta;\eta))(\overrightarrow{M[\theta]})$ by $\mathcal{C}$ def.
   $= \mathtt{in}_0\, ([\![T_z]\!](\theta;\eta)(\overrightarrow{M[\theta]}))$ by $\mathtt{in}^*$ def.
   $= \mathtt{in}_0\, ([\![T_z\, \vec{M}]\!](\theta;\eta))$ by $[\![T\, \vec{M}]\!]$ def.

2. $[\![T_{\mathtt{Rec}}\, (\mathsf{suc}\, N)\, \vec{M}]\!](\theta;\eta)$
   $= [\![T_{\mathtt{Rec}}]\!](\theta;\eta)(\mathsf{suc}\, N[\theta])(\overrightarrow{M[\theta]})$ by $[\![T\, \vec{M}]\!]$ def.
   $= \mathbf{Rec}\, \mathcal{C}\, \mathcal{F}\, (\mathsf{suc}\, N[\theta])\, \overrightarrow{M[\theta]}$ by $[\![T_{\mathtt{Rec}}]\!]$ def.
   $= \mathcal{F}\, N[\theta]\, (\mathbf{Rec}\, \mathcal{C}\, \mathcal{F}\, N[\theta])\, \overrightarrow{M[\theta]}$ by **Rec** def.
   $= (\mathtt{in}^*_{\mathsf{suc}}\, [\![T_s]\!](\theta, N[\theta]/n; \eta, (\mathbf{Rec}\, \mathcal{C}\, \mathcal{F}\, N[\theta])/X))(\overrightarrow{M[\theta]})$ by $\mathcal{F}$ def.
   $= \mathtt{in}_{\mathsf{suc}}\, ([\![T_s]\!](\theta, N[\theta]/n; \eta, (\mathbf{Rec}\, \mathcal{C}\, \mathcal{F}\, N[\theta])/X)(\overrightarrow{M[\theta]}))$ by $\mathtt{in}^*$ def.
   $= \mathtt{in}_{\mathsf{suc}}\, ([\![T_s]\!]((\mathsf{id}_\Delta, N/n)[\theta]; \eta, [\![T_{\mathtt{Rec}}\, N]\!](\theta;\eta)/X)(\overrightarrow{M[\theta]}))$ by Def. 22 and $[\![T_{\mathtt{Rec}}]\!]$ def.
   $= \mathtt{in}_{\mathsf{suc}}\, ([\![T_s[N/n; (T_{\mathtt{Rec}}\, N)/X]]\!](\theta;\eta)(\overrightarrow{M[\theta]}))$ by Lemma 19
   $= \mathtt{in}_{\mathsf{suc}}\, ([\![T_s[N/n; (T_{\mathtt{Rec}}\, N)/X]\, \vec{M}]\!](\theta;\eta))$ by $[\![T\, \vec{M}]\!]$ def.

◀

▶ **Theorem 30** (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* $v \in [\![T]\!](\theta; \eta)$.

Proof. The proof is by induction on the typing derivation. Technically this is a mutual induction on the dual judgments of type checking and synthesis. In each case we introduce the assumptions $\vdash \theta : \Delta$ and $\eta \in [\![\Xi]\!](\theta)$ and $\sigma \in [\![\Gamma]\!](\theta; \eta)$, where Δ, Ξ and Γ appear in the conclusion of the relevant typing rule. We will slightly abuse notation to introduce an existentially quantified variable in the judgment $t[\theta; \sigma] \Downarrow v \in \mathcal{V}$, to mean that $\exists v.\, t[\theta; \sigma] \Downarrow v \wedge v \in \mathcal{V}$.

Case:

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

$\langle\rangle[\theta; \sigma] \Downarrow \langle\rangle$	by `e-unit`
$\langle\rangle \in [\![1]\!](\theta; \eta)$	by $[\![1]\!]$ def.

Case:

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

$\sigma \in [\![\Gamma]\!](\theta; \eta)$	assumption of Thm 30
$x{:}T \in \Gamma$	premise of `t-var`
$\sigma(x) = v \in [\![T]\!](\theta; \eta)$	by Def. 16
$x[\theta; \sigma] \Downarrow v$	by `e-var`

Case:

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

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

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

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

$\forall \vec{M} \in [\![(\overrightarrow{u{:}U})]\!](\theta).\ \forall v \in [\![R]\!](\theta'; \eta).\ c \cdot \vec{M}\, v \Downarrow w \in [\![S]\!](\theta'; \eta)$, where $\theta' = \theta, \overrightarrow{M/u}$.

Suppose $\vec{M} \in [\![(\overrightarrow{u{:}U})]\!](\theta)$ and $v \in [\![R]\!](\theta'; \eta)$ where $\theta' = \theta, \overrightarrow{M/u}$.

$\vdash \theta : \Delta$	assumption of Thm 30
$\vdash \theta' : \Delta, \overrightarrow{u{:}U}$	by index substitution typing

Let $\sigma' = \sigma, v/x$.

$\sigma \in [\![\Gamma]\!](\theta; \eta)$	assumption of Thm 30
$\sigma \in [\![\Gamma]\!](\theta'; \eta)$	since $\vec{u} \notin \mathrm{FV}(\Gamma)$
$\sigma' \in [\![\Gamma, x{:}R]\!](\theta'; \eta)$	by Def. 16
$s[\theta'; \sigma'] \Downarrow w \in [\![S]\!](\theta'; \eta)$	by I.H. with θ', η and σ'
$c \cdot \vec{M}\, v \Downarrow w$	by `e-app-lam`

Case:

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

$q[\theta;\sigma] \Downarrow c \in [\![(\overrightarrow{u{:}U});\ R \to S]\!](\theta;\eta)$	by I.H.
$\overrightarrow{M[\theta]} \in [\![(\overrightarrow{u{:}U})]\!](\theta)$	by Lemma 10.2
$r[\theta;\sigma] \Downarrow v \in [\![R[\overrightarrow{M/u}]]\!](\theta;\eta)$	by I.H.
$v \in [\![R]\!]((\mathsf{id}_\Delta, \overrightarrow{M/u})[\theta];\eta)$	by Lemma 19
$v \in [\![R]\!](\theta, \overrightarrow{M[\theta]/u};\eta)$	by Def. 22
$c \cdot \overrightarrow{M[\theta]}\ v \Downarrow w \in [\![S]\!](\theta, \overrightarrow{M[\theta]/u};\eta)$	by $[\![(\overrightarrow{u{:}U});\ R \to S]\!](\theta;\eta)$ def.
$w \in [\![S]\!]((\mathsf{id}_\Delta, \overrightarrow{M/u})[\theta];\eta)$	by Def. 22
$w \in [\![S[\overrightarrow{M/u}]]\!](\theta;\eta)$	by Lemma 19
$(q\,\vec{M}\,r)[\theta;\sigma] \Downarrow w$	by `e-app`

Case:

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

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

Case:

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

$p[\theta;\sigma] \Downarrow w \in [\![T_1 \times T_2]\!](\theta;\eta)$	by I.H.
$w = \langle v_1, v_2\rangle$ where $v_i \in [\![T_i]\!](\theta;\eta)$ for $i \in \{1,2\}$	by $[\![T_1 \times T_2]\!]$ def.
$\sigma, v_1/x_1, v_2/x_2 \in [\![\Gamma, x_1{:}T_1, x_2{:}T_2]\!](\theta;\eta)$	by Def. 16
$s[\theta;\sigma, v_1/x_1, v_2/x_2] \Downarrow v \in [\![T]\!](\theta;\eta)$	by I.H.
$t[\theta;\sigma] \Downarrow v$	by `e-split`

Case:

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

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

$t_i[\theta;\sigma] \Downarrow v_i \in [\![T_i]\!](\theta;\eta)$	by I.H.
$(\texttt{in}_i\,t_i)[\theta;\sigma] \Downarrow \texttt{in}_i\,v_i$	by `e-in`$_i$.
$\texttt{in}_i\,v_i \in \texttt{in}_i\,[\![T_i]\!](\theta;\eta) \subseteq [\![T_1 + T_2]\!](\theta;\eta)$	by $[\![T_1 + T_2]\!]$ def.

Case:

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

$s[\theta;\sigma] \Downarrow w \in [\![T_1 + T_2]\!](\theta;\eta)$	by I.H.
$w \in \mathtt{in}_1\, [\![T_1]\!](\theta;\eta)$ or $w \in \mathtt{in}_2\, [\![T_2]\!](\theta;\eta)$	by $[\![T_1 + T_2]\!]$ def.
$w = \mathtt{in}_i\, v_i$ where $v_i \in [\![T_i]\!](\theta;\eta)$, for some $i \in \{1,2\}$.	
$\sigma \in [\![\Gamma]\!](\theta;\eta)$	by assumption of Thm 30
$\sigma, v_i/x_i \in [\![\Gamma, x_i{:}T_i]\!](\theta;\eta)$	by Def. 16
$t_i[\theta;\sigma, v_i/x_i] \Downarrow v \in [\![T]\!](\theta;\eta)$	by I.H. on t_i
$t[\theta;\sigma] \Downarrow v$	by $\mathtt{e\text{-}case\text{-}in}_i$

Case:

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

$s[\theta;\sigma] \Downarrow w \in [\![S[M/u]]\!](\theta;\eta)$	by I.H.
$w \in [\![S]\!]((\mathsf{id}_\Delta, M/u)[\theta];\eta)$	by Lemma 19
$w \in [\![S]\!](\theta, M[\theta]/u;\eta)$	by Def. 22
$(\mathtt{pack}\,(M, s))[\theta;\sigma] \Downarrow \mathtt{pack}\,(M[\theta], w)$	by $\mathtt{e\text{-}pack}$
$M[\theta] \in [\![U]\!](\theta)$	by Lemma 10.1
$\mathtt{pack}\,(M[\theta], w) \in [\![\Sigma u{:}U.\, S]\!](\theta;\eta)$	by $[\![\Sigma u{:}U.\, S]\!]$ def.

Case:

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

$p[\theta;\sigma] \Downarrow w' \in [\![\Sigma u{:}U.\, S]\!](\theta;\eta)$	by I.H.
$w' = \mathtt{pack}\,(M, w)$ where $M \in [\![U]\!](\theta)$ and $w \in [\![S]\!](\theta, M/u;\eta)$	by $[\![\Sigma u{:}U.\, S]\!]$ def.
Let $\theta' = \theta, M/u$.	
$\vdash \theta' : \Delta, u{:}U$	by index substitution typing
$\sigma \in [\![\Gamma]\!](\theta;\eta)$	assumption of Thm 30
$\sigma \in [\![\Gamma]\!](\theta';\eta)$	since $u \notin \mathrm{FV}(\Gamma)$
Let $\sigma' = \sigma, w/x$.	
$\sigma' \in [\![\Gamma, x{:}S]\!](\theta';\eta)$	by Def. 16
$q[\theta';\sigma'] \Downarrow v \in [\![T]\!](\theta';\eta)$	by I.H.
$(\mathtt{unpack}\, p\, \mathtt{as}\, (u, x)\, \mathtt{in}\, q)[\theta;\sigma] \Downarrow v$	by $\mathtt{e\text{-}unpack}$
$v \in [\![T]\!](\theta;\eta)$	since $u \notin \mathrm{FV}(T)$

Case:

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

$\vdash M[\theta] = N[\theta]$	by Thm 1.3
$\mathtt{refl} \in [\![M = N]\!](\theta;\eta)$	by $[\![M = N]\!]$ def.
$\mathtt{refl}[\theta;\sigma] \Downarrow \mathtt{refl}$	by $\mathtt{e\text{-}refl}$

Case:

$$\frac{\Delta;\Xi;\Gamma \vdash q \Rightarrow M = N \quad \Delta' \vdash \Theta : \Delta \quad \Theta \text{ mgu. } \Delta' \vdash M[\Theta] = N[\Theta] \quad \Delta';\Xi[\Theta];\Gamma[\Theta] \vdash s \Leftarrow T[\Theta]}{\Delta;\Xi;\Gamma \vdash \mathtt{eq}\, q\, \mathtt{with}\, (\Delta'.\Theta \mapsto s) \Leftarrow T}\ \mathtt{t\text{-}eq}$$

$q[\theta;\sigma] \Downarrow w \in [\![M = N]\!](\theta;\eta)$	by I.H.
$w = \mathtt{refl}$ and $\vdash M[\theta] = N[\theta]$	by $[\![M = N]\!]$ def.
$\Delta' \vdash \Theta : \Delta$ and Θ mgu s.t. $\Delta' \vdash M[\Theta] = N[\Theta]$	premises of `t-eq`
$\vdash \theta : \Delta$	assumption of Thm 30
$\Delta' \vdash \Theta \doteq \theta \searrow (\cdot \mid \theta')$	by Thm 3
$[\![\Xi[\Theta]]\!](\theta') = [\![\Xi]\!](\Theta[\theta'])$ and $[\![T[\Theta]]\!](\theta';\eta) = [\![T]\!](\Theta[\theta';\eta)$	by Lemma 19
$[\![\Xi[\Theta]]\!](\theta') = [\![\Xi]\!](\theta])$ and $[\![T[\Theta]]\!](\theta';\eta) = [\![T]\!](\theta;\eta)$	by Thm 2
Extending Lemma 19 from types T to typing contexts Γ, $[\![\Gamma[\Theta]]\!](\theta';\eta) = [\![\Gamma]\!](\Theta[\theta';\eta) = [\![\Gamma]\!](\theta;\eta)$.	
$\eta \in [\![\Xi[\Theta]]\!](\theta')$	since $\eta \in [\![\Xi]\!](\theta)$
$\sigma \in [\![\Gamma[\Theta]]\!](\theta';\eta)$	since $\sigma \in [\![\Gamma]\!](\theta;\eta)$
$s[\theta';\sigma] \Downarrow v \in [\![T[\Theta]]\!](\theta';\eta)$	by I.H.
$v \in [\![T]\!](\theta;\eta)$	
$(\mathtt{eq}\, q\, \mathtt{with}\, (\Delta'.\Theta \mapsto s))[\theta;\sigma] \Downarrow v$	by `e-eq`

Case:

$$\frac{\Delta;\Xi;\Gamma \vdash s \Rightarrow M = N \quad \Delta \vdash M \neq\!\!\!\!\!\!\cdot\ \ N}{\Delta;\Xi;\Gamma \vdash \mathtt{eq\_abort}\, s \Leftarrow T}\ \mathtt{t\text{-}eqfalse}$$

$s[\theta;\sigma] \Downarrow w \in [\![M = N]\!](\theta;\eta)$	by I.H.
$w = \mathtt{refl}$ and $\vdash M[\theta] = N[\theta]$	by $[\![M = N]\!]$ def.
$\Delta \vdash M \doteq N$ under θ	since $\vdash \theta : \Delta$
$\Delta \vdash M \neq\!\!\!\!\!\!\cdot\ \ N$	premise of `t-eqfalse`
Contradiction	
Derive $(\mathtt{eq\_abort}\, s)[\theta;\sigma] \Downarrow v \in [\![T]\!](\theta;\eta)$	

Case:

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

Let $R = \mu X{:}K.\, \Lambda\, \vec{u}.\, S$.

$s[\theta;\sigma] \Downarrow w \in [\![S[\overrightarrow{M/u}; R/X]]\!](\theta;\eta)$	by I.H.
$(\mathtt{in}_\mu\, s)[\theta;\sigma] \Downarrow \mathtt{in}_\mu\, w$	by $\mathtt{e\text{-}in}_\mu$
$\mathtt{in}_\mu\, w \in [\![R\, \vec{M}]\!](\theta;\eta)$	by Lemma 27

Case:

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

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 to show that $c \in [\![(\overrightarrow{u{:}U});\ C \to S]\!](\theta;\eta)$,

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 rules we know that $K = \Pi\overrightarrow{u'{:}U}.*$ so $[\![K]\!](\theta) = \vec{\mathcal{U}} \to 2^\Omega$. Let $\mathcal{L} = [\![K]\!](\theta)$ and define $\mathcal{F} \in \mathcal{L} \to \mathcal{L}$ by $\mathcal{F}(\mathcal{X}) = \mathtt{in}^*_\mu\, [\![\Lambda\, \overrightarrow{u'}.\, R]\!](\theta; \eta, \mathcal{X}/X)$.

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

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

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

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

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 Lemma 26. To this end, suppose $\mathcal{X} \in \vec{\mathcal{U}} \to 2^\Omega = \mathcal{L}$ and $c \in \vec{\mathcal{U}}, \mathcal{X} \to \mathcal{B}$. The goal is now to show $c \in \vec{\mathcal{U}}, \mathcal{F}(\mathcal{X}) \to \mathcal{B}$.

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 28, 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 $(\overrightarrow{u{:}U});\, X\, \vec{u} \to S$ and $(\overrightarrow{u{:}U});\, R[\overrightarrow{u/u'}] \to S$ appearing in the premise of `t-rec`. 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 $\eta, \mathcal{X}/X \in [\![\Xi, X{:}K]\!](\theta)$.

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

$$
\begin{aligned}
&[\![(\overrightarrow{u{:}U});\, R[\overrightarrow{u/u'}] \to S]\!](\theta; \eta, \mathcal{X}/X) \\
&= \vec{\mathcal{U}}, (\vec{M} \mapsto [\![R[\overrightarrow{u/u'}]]\!](\theta, \overrightarrow{M/u}; \eta, \mathcal{X}/X)) \to (\vec{M} \mapsto [\![S]\!](\theta, \overrightarrow{M/u}; \eta, \mathcal{X}/X)) \\
&= \vec{\mathcal{U}}, (\vec{M} \mapsto [\![R]\!]((\mathsf{id}_\Delta, \overrightarrow{u/u'})[\theta, \overrightarrow{M/u'}]; \eta, \mathcal{X}/X)) \to (\vec{M} \mapsto [\![S]\!](\theta, \overrightarrow{M/u}; \eta)) \\
&\qquad \text{by Lemma 19 and again dropping a mapping for } X \\
&= \vec{\mathcal{U}}, (\vec{M} \mapsto [\![R]\!](\theta, \overrightarrow{M/u'}; \eta, \mathcal{X}/X)) \to (\vec{M} \mapsto [\![S]\!](\theta, \overrightarrow{M/u}; \eta)) \\
&= \vec{\mathcal{U}}, \mathcal{A} \to \mathcal{B}.
\end{aligned}
$$

Our assumption from Lemma 26 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:

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

$s[\theta;\sigma] \Downarrow w \in [\![T_z\, \vec{M}]\!](\theta;\eta)$ by I.H.
$(\mathtt{in}_0\, s)[\theta;\sigma] \Downarrow \mathtt{in}_0\, w$ by $\mathtt{e\text{-}in}_0$
$\mathtt{in}_0\, w \in \mathtt{in}_0\, [\![T_z\, \vec{M}]\!](\theta;\eta)$
$\mathtt{in}_0\, w \in [\![T_{\mathtt{Rec}}\, 0\, \vec{M}]\!](\theta;\eta)$ by Lemma 29

Case:

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

$s[\theta;\sigma] \Downarrow w \in [\![T_s[N/u; (T_{\mathtt{Rec}}\, N)/X]\, \vec{M}]\!](\theta;\eta)$ by I.H.
$(\mathtt{in}_{\mathsf{suc}}\, s)[\theta;\sigma] \Downarrow \mathtt{in}_{\mathsf{suc}}\, w$ by $\mathtt{e\text{-}in}_{\mathsf{suc}}$
$\mathtt{in}_{\mathsf{suc}}\, w \in \mathtt{in}_{\mathsf{suc}}\, ([\![T_s[N/u; (T_{\mathtt{Rec}}\, N)/X]\, \vec{M}]\!](\theta;\eta))$
$\mathtt{in}_{\mathsf{suc}}\, w \in [\![T_{\mathtt{Rec}}\, (\mathsf{suc}\, N)\, \vec{M}]\!](\theta;\eta)$ by Lemma 29

Case:

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

$s[\theta;\sigma] \Downarrow w \in [\![T_{\mathtt{Rec}}\, 0\, \vec{M}]\!](\theta;\eta)$ by I.H.
$w = \mathtt{in}_0\, v$ for some $v \in ([\![T_z\, \vec{M}]\!](\theta;\eta))$ by Lemma 29
$(\mathtt{out}_0\, s)[\theta;\sigma] \Downarrow v$ by $\mathtt{e\text{-}out}_0$

Case:

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

$s[\theta;\sigma] \Downarrow w \in [\![T_{\mathtt{Rec}}\, (\mathsf{suc}\, N)\, \vec{M}]\!](\theta;\eta)$ by I.H.
$w = \mathtt{in}_{\mathsf{suc}}\, v$ for some $v \in [\![T_s[N/u; (T_{\mathtt{Rec}}\, N)/X]\, \vec{M}]\!](\theta;\eta)$ by Lemma 29
$(\mathtt{out}_{\mathsf{suc}}\, s)[\theta;\sigma] \Downarrow v$ by $\mathtt{e\text{-}out}_{\mathsf{suc}}$

Case:

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

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

$t_z[\theta;\sigma] \Downarrow w \in [\![S[0/u]]\!](\theta;\eta)$	by I.H.
$w \in [\![S]\!]((\mathsf{id}_\Delta, 0/u)[\theta];\eta)$	by Lemma 19
$w \in [\![S]\!](\theta, 0/u;\eta)$	by Def. 22
$c \cdot 0\,\langle\rangle \Downarrow w$	by $\texttt{e-app-ind}_0$
Step case: $N = \mathsf{suc}\,N'$ for some $N' \in [\![\mathsf{nat}]\!]$.	
$c \cdot N'\,\langle\rangle \Downarrow v \in [\![S]\!](\theta, N'/u;\eta)$	by inner I.H.
Let $\theta' = \theta, N'/u$, so $\vdash \theta' : \Delta, u{:}\,\mathsf{nat}$.	
$\sigma \in [\![\Gamma]\!](\theta';\eta)$	since $u \notin \mathrm{FV}(\Gamma)$
$\sigma, v/x \in [\![\Gamma, x{:}S]\!](\theta';\eta)$	by Def. 16
$t_s[\theta';\sigma, v/x] \Downarrow w \in [\![S[\mathsf{suc}\,u/u]]\!](\theta';\eta)$	by I.H.
$w \in [\![S]\!]((\mathsf{id}_\Delta, \mathsf{suc}\,u/u)[\theta'];\eta)$	by Lemma 19
$w \in [\![S]\!](\theta', (\mathsf{suc}\,u)[\theta']/u;\eta)$	by Def. 22
$w \in [\![S]\!](\theta', \mathsf{suc}\,N'/u;\eta)$	by Def. 21
$w \in [\![S]\!](\theta, N/u;\eta)$	overwriting u in θ'
$c \cdot N\,\langle\rangle \Downarrow w$	by $\texttt{e-app-ind}_{\mathsf{suc}}$

◀