A finite axiomatisation of inductive-inductive definitions

Fredrik Nordvall Forsberg and Anton Setzer *

Swansea University
Singleton Park
Swansea SA2 8PP, UK

January 20, 2012

Abstract

Induction-induction is a principle for mutually defining data types $A : \mathsf{Set}$, $B : A \to \mathsf{Set}$. Both A and B are defined inductively, and the constructors for A can refer to B and vice versa. In addition, the constructor for B can refer to *the constructor* for A. Induction-induction occurs in a natural way when formalising dependent type theory in type theory. We give some examples of inductive-inductive definitions, such as the set of surreal numbers. We then give a new finite axiomatisation of the principle of induction-induction, and prove its consistency by constructing a model.

1 Introduction

Induction is an important principle of definition and reasoning, especially so in constructive mathematics. Martin-Löf's formulation [ML84] of type theory, for instance, includes inductive definitions of among others Cartesian products, disjoint unions, the natural numbers, lists, well-orderings, the identity set, finite sets, and a universe à la Tarski.

These examples can be categorised as different kinds of inductive definitions. The first few (up to well-orderings) are just ordinary inductive definitions, where a single set is defined inductively. A typical example is the type $W(A, B)$ of well-orderings, parameterised by $A : \mathsf{Set}$, $B : A \to \mathsf{Set}$. The introduction rule is:

$$\frac{a : A \qquad f : B(a) \to W(A, B)}{\mathsf{sup}(a, f) : W(A, B)}$$

*Both authors are supported by EPSRC grant EP/G033374/1, Theory and applications of induction-recursion.

Here $a : A$ is a *non-inductive* argument, whereas $f : B(a) \to W(A, B)$ is an *inductive* argument because of the occurrence of $W(A, B)$. Note how the later argument depends on the earlier non-inductive argument.

The identity type and the finite sets are examples of inductive families [Dyb94], where a family $X : I \to \mathsf{Set}$ for some fixed index set I is defined inductively simultaneously. For the family $\mathsf{Fin} : \mathbb{N} \to \mathsf{Set}$ of finite sets, the index set is $\mathbb{N}$, and we have introduction rules

$$\frac{n : \mathbb{N}}{\mathsf{z}_n : \mathsf{Fin}(n+1)} \qquad \frac{n : \mathbb{N} \qquad m : \mathsf{Fin}(n)}{\mathsf{s}_n(m) : \mathsf{Fin}(n+1)}$$

Thus the type $\mathsf{Fin}(n+1)$ has $n+1$ elements z_n, $\mathsf{s}_n(\mathsf{z}_{n-1})$, $\mathsf{s}_n(\mathsf{s}_{n-1}(\mathsf{z}_{n-2}))$ up to $\mathsf{s}_n(\mathsf{s}_{n-1}(\cdots \mathsf{s}_1(\mathsf{z}_0)))$. The type of the inductive argument $m : \mathsf{Fin}(n)$ of the second rule has index n, which is different from the index $n+1$ of the type of the constructed element. Thus the whole family has to be defined simultaneously.

The universe à la Tarski is an example of an inductive-recursive definition [Dyb00], where a set U is defined inductively together with a recursive function $T : U \to \mathsf{Set}$. The constructors for U may depend negatively on T applied to elements of U, as is the case if U for example is closed under dependent function spaces:

$$\frac{a : U \qquad b : T(a) \to U}{\pi(a, b) : U}$$

with $T(\pi(a, b)) = (x : T(a)) \to T(b(x))$.[1]

Here, $T : U \to \mathsf{Set}$ is defined recursively. Sometimes, however, one might not want to give $T(u)$ completely as soon as $u : U$ is introduced, but instead define T inductively as well. This is the principle of *induction-induction*. A set A is inductively defined simultaneously with an A-indexed set B, which is also inductively defined, and the introduction rules for A may also refer to B. Typical introduction rules might take the form

$$\frac{a : A \quad b : B(a) \quad \ldots}{\mathsf{intro}_A(a, b, \ldots) : A} \qquad \frac{a_0 : A \qquad b : B(a_0) \quad a_1 : A \quad \ldots}{\mathsf{intro}_B(a_0, b, a_1, \ldots) : B(a_1)}$$

This is not a simple mutual inductive definition of two sets, as B is indexed by A. It is not an ordinary inductive family, as A may refer to B. Finally, it is not an instance of induction-recursion, as B is constructed inductively, not recursively.

In this article, we give a new finite axiomatisation of a type theory with inductive-inductive definitions. It differs from our earlier axiomatisation [NFS10] in that it is finite, and is hopefully easier to understand. The price we have to pay for this simplicity is that the natural full function space set-theoretical model construction becomes slightly more involved.

[1] The notation for the dependent function space and other type-theoretical constructs is explained in Section 2.

1.1 Examples of inductive-inductive definitions

In this section, we give some examples of inductive-inductive definitions, starting with the perhaps most important one:

Example 1 (Contexts and types)**.** Danielsson [Dan07] and Chapman [Cha09] model the syntax of dependent type theory in the theory itself by inductively defining contexts, types (in a given context) and terms (of a given type). To see the inductive-inductive nature of the construction, it is enough to concentrate on contexts and types.

Informally, we have an empty context ε, and if we have any context Γ and a valid type σ in that context, then we can extend the context with a fresh variable $x : \sigma$ to get a new context $\Gamma, x : \sigma$. This is the only way contexts are formed. We end up with the following inductive definition of the set of contexts (with $\Gamma \triangleright \sigma$ meaning $\Gamma, x : \sigma$ since we are using de Bruijn indices):

$$\frac{}{\varepsilon : \mathsf{Ctxt}} \qquad \frac{\Gamma : \mathsf{Ctxt} \quad \sigma : \mathsf{Ty}(\Gamma)}{\Gamma \triangleright \sigma : \mathsf{Ctxt}}$$

Moving on to types, we have a base type ι (valid in any context) and dependent function types: if σ is a type in context Γ, and τ is a type in $\Gamma, x : \sigma$ (x is the variable from the domain), then $\Pi(\sigma, \tau)$ is a type in the original context. This leads us to the following inductive definition of $\mathsf{Ty} : \mathsf{Ctxt} \to \mathsf{Set}$:

$$\frac{\Gamma : \mathsf{Ctxt}}{\iota_\Gamma : \mathsf{Ty}(\Gamma)} \qquad \frac{\Gamma : \mathsf{Ctxt} \quad \sigma : \mathsf{Ty}(\Gamma) \quad \tau : \mathsf{Ty}(\Gamma \triangleright \sigma)}{\Pi_\Gamma(\sigma, \tau) : \mathsf{Ty}(\Gamma)}$$

Note that the definition of Ctxt refers to Ty, so both sets have to be defined simultaneously. Note also how the introduction rule for Π explicitly focuses on a specific constructor in the index of the type of τ. ■

Often, one wishes to define a set A where all elements of A satisfy some property $P : A \to \mathsf{Set}$. If P is inductively defined, one can define A and P simultaneously and achieve that every element of A satisfies P by construction. One example of such a data type is the type of sorted lists:

Example 2 (Sorted lists)**.** Let us define a data type consisting of sorted lists (of natural numbers, say). With induction-induction, we can simultaneously define the set $\mathsf{SortedList}$ of sorted lists and the predicate $\leq_{\mathrm{L}} : (\mathbb{N} \times \mathsf{SortedList}) \to \mathsf{Set}$ with $n \leq_{\mathrm{L}} \ell$ true if n is less than or equal to every element of ℓ.

The empty list is certainly sorted, and if we have a proof p that n is less than or equal to every element of the list ℓ, we can put n in front of ℓ to get a new sorted list $\mathsf{cons}(n, \ell, p)$. Translated into introduction rules, this becomes:

$$\frac{}{\mathsf{nil} : \mathsf{SortedList}} \qquad \frac{n : \mathbb{N} \qquad \ell : \mathsf{SortedList} \qquad p : n \leq_{\mathrm{L}} \ell}{\mathsf{cons}(n, \ell, p) : \mathsf{SortedList}}$$

For $\leq_{\mathrm{L}}$, we have that every $m : \mathbb{N}$ is trivially smaller than every element of the empty list, and if $m \leq n$ and inductively $m \leq_{\mathrm{L}} \ell$, then $m \leq_{\mathrm{L}} \mathsf{cons}(n, \ell, p)$:

$$\frac{}{\mathsf{triv}_m : m \leq_{\mathrm{L}} \mathsf{nil}} \qquad \frac{q : m \leq n \qquad p_{m,\ell} : m \leq_{\mathrm{L}} \ell}{\ll q, p_{m,\ell} \gg : m \leq_{\mathrm{L}} \mathsf{cons}(n, \ell, p)}$$

This makes sense even if the order $\leq$ is not transitive. If it is (as the standard order on the natural numbers is, for example), the argument $p_{m,\ell} : m \leq_{\mathrm{L}} \ell$ can be dropped from the constructor $\ll \cdot \gg$, since we already have $q : m \leq n$ and $p : n \leq_{\mathrm{L}} \ell$, hence by transitivity we must have $m \leq_{\mathrm{L}} \ell$.

Of course, there are also many alternative ways to define such a data type using ordinary induction (or using e.g. induction-recursion, similarly to C. Coquand's definition of fresh lists as reported by Dybjer [Dyb00]). ■

Example 3 (Conway's surreal numbers)**.** Conway [Con01] informally uses induction-induction (but couched in ZF set theory, not type theory) in order to define his *surreal numbers*. The class [2] of surreal numbers is defined inductively, together with an order relation on surreal numbers which is also defined inductively:

- A surreal number $X = (X_{\mathrm{L}}, X_{\mathrm{R}})$ consists of two sets X_{L} and X_{R} of surreal numbers, such that no element from X_{L} is greater than any element from X_{R}.
- A surreal number $Y = (Y_{\mathrm{L}}, Y_{\mathrm{R}})$ is greater than another surreal number $X = (X_{\mathrm{L}}, X_{\mathrm{R}})$, $X \leq Y$, if and only if
  - there is no $x \in X_{\mathrm{L}}$ such that $Y \leq x$, and
  - there is no $y \in Y_{\mathrm{R}}$ such that $y \leq X$.

Both rules can be understood as inductive definitions. Notice how the second definition only makes sense in the presence of the first definition, and how the first definition already refers to the second.

As an inductive definition, the negative occurrence of $\leq$ in the definition of the class of surreal numbers is problematic. We can get around this by simultaneously defining the class $\mathsf{Surreal} : \mathsf{Set}$ together with two relations $\leq : \mathsf{Surreal} \to \mathsf{Surreal} \to \mathsf{Set}$ and $\not\leq : \mathsf{Surreal} \to \mathsf{Surreal} \to \mathsf{Set}$ as follows:

- If X_{L} and X_{R} are sets of surreal numbers, and for all $x \in X_{\mathrm{L}}$, $y \in X_{\mathrm{R}}$ we have $x \not\leq y$, then $(X_{\mathrm{L}}, X_{\mathrm{R}})$ is a surreal number.
- Assume $X = (X_{\mathrm{L}}, X_{\mathrm{R}})$ and $Y = (Y_{\mathrm{L}}, Y_{\mathrm{R}})$ are surreal numbers. If
  - for all $x \in X_{\mathrm{L}}$ we have $Y \not\leq x$, and
  - for all $y \in Y_{\mathrm{R}}$ we have $y \not\leq X$,

  then $X \leq Y$.
- Assume $X = (X_{\mathrm{L}}, X_{\mathrm{R}})$ and $Y = (Y_{\mathrm{L}}, Y_{\mathrm{R}})$ are surreal numbers.
  - If there exist $x \in X_{\mathrm{L}}$ such that $Y \leq x$, then $X \not\leq Y$.
  - If there exist $y \in Y_{\mathrm{R}}$ such that $y \leq X$, then $X \not\leq Y$.

[2] The surreal numbers form a class, not a set, since they contain the class of ordinals. This can be avoided by referring to a universe.

We see that $\mathsf{Surreal} : \mathsf{Set}$ together with $\leq, \not\leq : \mathsf{Surreal} \to \mathsf{Surreal} \to \mathsf{Set}$ are defined inductive-inductively.

Mamane [Mam06] develops the theory of surreal numbers in the proof assistant Coq, using an encoding to reduce the inductive-inductive definition to an ordinary inductive one. ■

Note that these examples strictly speaking refer to extensions of inductive-inductive definitions as presented in this article. Example 1 in full would be an example of a defining of a telescope $A : \mathsf{Set}$, $B : A \to \mathsf{Set}$, $C : (x : A) \to B(x) \to \mathsf{Set}$, ... inductive-inductively. In Example 2, $A : \mathsf{Set}$ and $B : A \to I \to \mathsf{Set}$ for some previously defined set I is defined, and Example 3 gives an inductive-inductive definition of $A : \mathsf{Set}$, $B, B' : A \to A \to \mathsf{Set}$. In the future, we plan to publish an axiomatisation which captures all these examples in full. For pedagogical reasons, we think it is preferable to first only treat the simpler case $A : \mathsf{Set}$, $B : A \to \mathsf{Set}$ as in the current article.

1.2 Inductive-inductive definitions versus inductive-recursive definitions

In both an inductive-inductive and an inductive-recursive definition, a set U and a family $T : U \to \mathsf{Set}$ are defined simultaneously. The difference between the two principles is how T is defined: inductively or recursively. We discuss in the following first the difference between an inductive and a recursive definition. To exemplify this difference, consider the following two definitions of a data type $\mathsf{Nonempty} : \mathbb{N} \to \mathsf{Set}$ of non-empty lists of a certain length (with elements from a set A):

Inductive definition The singleton list $[a]$ has length 1, and if a is an element, and the list ℓ has length n, then $\mathsf{cons}(a, \ell)$ is a list of length $n+1$. As an inductive definition, this becomes

$$\frac{a : A}{[a] : \mathsf{Nonempty}_{\mathrm{ind}}(1)} \qquad \frac{a : A \qquad \ell : \mathsf{Nonempty}_{\mathrm{ind}}(n)}{\mathsf{cons}(a, \ell) : \mathsf{Nonempty}_{\mathrm{ind}}(n+1)}$$

Notice that there is no constructor which constructs elements in $\mathsf{Nonempty}_{\mathrm{ind}}(0)$.

Recursive definition In the recursive definition, we define the set $\mathsf{Nonempty}_{\mathrm{rec}}(n)$ for every natural number:

$$\begin{aligned} \mathsf{Nonempty}_{\mathrm{rec}}(0) &= \mathbf{0} \\ \mathsf{Nonempty}_{\mathrm{rec}}(1) &= A \\ \mathsf{Nonempty}_{\mathrm{rec}}(n+2) &= A \times \mathsf{Nonempty}_{\mathrm{rec}}(n+1) \end{aligned}$$

In the recursive definition, $\mathsf{Nonempty}_{\mathrm{rec}}(k)$ is defined in one go, whereas the inductively defined $\mathsf{Nonempty}_{\mathrm{ind}}(k)$ is built up from below. In order to prove that $\mathsf{Nonempty}_{\mathrm{ind}}(0)$ is empty, one has to carry out a proof by induction over $\mathsf{Nonempty}_{\mathrm{ind}}$.

This difference is now carried over to an inductive-recursive/inductive-inductive definition of $U : \mathsf{Set}$, $T : U \to \mathsf{Set}$. In an inductive-inductive definition, T is generated inductively, i.e. given by a constructor $\mathsf{intro}_T : (x : F(U,T)) \to T(i(x))$ for some (strictly positive) functor F. In an inductive-recursive definition, on the other hand, T is defined by recursion on the way the elements of U are generated. This means that $T(\mathsf{intro}_U(x))$ must be given completely as soon as the constructor $\mathsf{intro}_U : G(U,T) \to U$ is introduced.

There are some practical differences between the two approaches. An inductive-inductive definition gives more freedom to describe the data type, in the sense that many different constructors for T can contribute to the set $T(\mathsf{intro}_U(x))$. However, because of the inductive generation of T, T can only occur positively in the type of the constructors for U (and T), whereas T can occur also negatively in an inductive-recursive definition.

2 Type-theoretical preliminaries

We work in a type theory with at least two universes Set and Type, with $\mathsf{Set} : \mathsf{Type}$ and Set a subuniverse of Type, i.e. if $A : \mathsf{Set}$ then $A : \mathsf{Type}$. Both Set and Type are closed under dependent function types, written $(x : A) \to B$, where B is a set or type depending on $x : A$. Abstraction is written as $\lambda x : A.e$, where $e : B$ depending on $x : A$, and application as $f(x)$. Repeated abstraction and application are written as $\lambda x_1 : A_1 \dots x_k : A_k.e$ and $f(x_1, \dots, x_k)$. If the type of x can be inferred, we simply write $\lambda x.e$ as an abbreviation. Furthermore, both Set and Type are closed under dependent products, written $(x : A) \times B$, where B is a set or type depending on $x : A$, with pairs $\langle a, b\rangle$, where $a : A$ and $b : B[x := a]$. We also have β- and η-rules for both dependent function types and products.

We add an empty type $\mathbf{0} : \mathsf{Set}$, with elimination $!_A : \mathbf{0} \to A$ for every $A : \mathsf{Set}$ (we will write $!$ for $!_A$ if A can be inferred from the context). We also add a unit type $\mathbf{1} : \mathsf{Set}$, with unique element $\star : \mathbf{1}$ and an η-rule stating that if $x : \mathbf{1}$, then $x = \star : \mathbf{1}$. Moreover, we include a two element set $\mathbf{2} : \mathsf{Set}$, with elements $\mathsf{tt} : \mathbf{2}$, $\mathsf{ff} : \mathbf{2}$ and elimination constant $\mathsf{if}\ \cdot\ \mathsf{then}\ \cdot\ \mathsf{else}\ \cdot : (a : \mathbf{2}) \to A(\mathsf{tt}) \to A(\mathsf{ff}) \to A(a)$ where $i : \mathbf{2} \Rightarrow A(i) : \mathsf{Type}$. It satisfies the obvious computation rules, i.e. $\mathsf{if}\ \mathsf{tt}\ \mathsf{then}\ a\ \mathsf{else}\ b = a$ and $\mathsf{if}\ \mathsf{ff}\ \mathsf{then}\ a\ \mathsf{else}\ b = b$.

With $\mathsf{if}\ \cdot\ \mathsf{then}\ \cdot\ \mathsf{else}\ \cdot$ and dependent products, we can now define the disjoint union of two sets $A + B := (x : \mathbf{2}) \times (\mathsf{if}\ x\ \mathsf{then}\ A\ \mathsf{else}\ B)$ with constructors $\mathsf{inl} = \lambda a : A.\langle \mathsf{tt}, a\rangle$ and $\mathsf{inr} = \lambda b : B.\langle \mathsf{ff}, b\rangle$, and prove the usual formation, introduction, elimination and equality rules. Importantly, we get large elimination for sums, since we have large elimination for $\mathbf{2}$. We can define the eliminator $[f, g] : (c : A + B) \to C(c)$, where $x : A + B \Rightarrow C(x) : \mathsf{Type}$ and $f : (a : A) \to C(\mathsf{inl}(a))$, $g : (b : B) \to C(\mathsf{inr}(b))$, satisfying the definitional equalities

$$[f, g](\mathsf{inl}(a)) = f(a)\ ,$$
$$[f, g](\mathsf{inr}(b)) = g(b)\ .$$

Intensional type theory in Martin-Löf's logical framework extended with dependent products and $\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$ has all the features we need. Thus, our development can be seen as an extension of the logical framework.

3 A finite axiomatisation

In this section, we give a finite axiomatisation of a type theory with inductive-inductive definitions. This axiomatisation differs slightly from our previous axiomatisation [NFS10], and is hopefully easier to understand. However, the definable sets should be the same for both axiomatisations.

We take inspiration from Dybjer and Setzer's axiomatisation of inductive-recursive definitions [DS99]. The main idea is to construct a universe consisting of codes for inductive-inductive definitions, together with a decoding function, which maps a code φ to the domain of the constructor for the inductively defined set represented by φ. Just as the second component $B : A \to \mathsf{Set}$ depends on the first set A, the universe $\mathsf{SP}^0_{\mathrm{B}} : \mathsf{SP}^0_{\mathrm{A}} \to \mathsf{Type}$ of codes for the second component will depend on the universe $\mathsf{SP}^0_{\mathrm{A}}$ of codes for the first set.

3.1 Dissecting an inductive-inductive definition

We want to formalise and internalise an inductive-inductive definition given by constructors

$$\mathsf{intro}_{\mathrm{A}} : \Phi_{\mathrm{A}}(A, B) \to A$$

and

$$\mathsf{intro}_{\mathrm{B}} : (x : \Phi_{\mathrm{B}}(A, B, \mathsf{intro}_{\mathrm{A}})) \to B(\theta(x))$$

for some $\Phi_{\mathrm{A}}(A, B) : \mathsf{Set}$, $\Phi_{\mathrm{B}}(A, B, \mathsf{intro}_{\mathrm{A}}) : \mathsf{Set}$ and $\theta : \Phi_{\mathrm{B}}(A, B, \mathsf{intro}_{\mathrm{A}}) \to A$. Here, $\theta(x)$ is the *index* of $\mathsf{intro}_{\mathrm{B}}(x)$, i.e. the element $a : A$ such that $\mathsf{intro}_{\mathrm{B}}(x) : B(a)$.

Not all expressions Φ_{A} and Φ_{B} give rise to acceptable inductive-inductive definitions. It is well known, for example, that the theory easily becomes inconsistent if A or B occur in negative positions in Φ_{A} or Φ_{B} respectively. Thus, we restrict our attention to a class of strictly positive functors.

These are based on the following analysis of what kind of premises can occur in a definition. A premise is either *inductive* or *non-inductive*. A non-inductive premise consists of a previously constructed set K, on which later premises can depend. An inductive premise is inductive in A or B. If it is inductive in A, it is of the form $K \to A$ for some previously constructed set K. Premises inductive in B are of the form $(x : K) \to B(i(x))$ for some $i : K \to A$.

If $K = \mathbf{1}$, we have the special case of a single inductive premise. In the case of B-inductive arguments, the choice of $i : \mathbf{1} \to A$ is then just a choice of a single element $a = i(\star) : A$ so that the premise is of the form $B(a)$.

3.2 The axiomatisation

We now give the formal rules for an inductive-inductive definition of $A : \mathsf{Set}$, $B : A \to \mathsf{Set}$. These consists of a set of rules for the universe $\mathsf{SP}^0_{\mathrm{A}}$ of descriptions of the set A and its decoding function $\mathsf{Arg}^0_{\mathrm{A}}$, a set of rules for the universe $\mathsf{SP}^0_{\mathrm{B}}$ and its decoding function $\mathsf{Arg}^0_{\mathrm{B}}$, and formation and introduction rules for $A : \mathsf{Set}$, $B : A \to \mathsf{Set}$ defined inductive-inductively by a pair of codes $\gamma_{\mathrm{A}} : \mathsf{SP}^0_{\mathrm{A}}$, $\gamma_{\mathrm{B}} : \mathsf{SP}^0_{\mathrm{B}}(\gamma_{\mathrm{A}})$.

3.2.1 The universe $\mathsf{SP}^0_{\mathrm{A}}$ of descriptions of A

We introduce the universe of codes for the index set with the formation rule

$$\frac{A_{\mathrm{ref}} : \mathsf{Set}}{\mathsf{SP}_{\mathrm{A}}(A_{\mathrm{ref}}) : \mathsf{Type}}$$

The set A_{ref} should be thought of as the elements of A that we can refer to in the code that we are defining. To start with, we cannot refer to any elements in A, and so we define $\mathsf{SP}^0_{\mathrm{A}} \coloneqq \mathsf{SP}_{\mathrm{A}}(\mathbf{0})$. After introducing an inductive argument $a : A$, we can refer to a in later arguments, so that A_{ref} will be extended to include a as well for the construction of the rest of the code.

The introduction rules for SP_{A} reflects the informal discussion in Section 3.1. The rules are as follows (we suppress the global premise $A_{\mathrm{ref}} : \mathsf{Set}$):

$$\frac{}{\mathsf{nil} : \mathsf{SP}_{\mathrm{A}}(A_{\mathrm{ref}})}$$

The code nil represents a trivial constructor $c : \mathbf{1} \to A$ (a base case).

$$\frac{K : \mathsf{Set} \qquad \gamma : K \to \mathsf{SP}_{\mathrm{A}}(A_{\mathrm{ref}})}{\mathsf{non\text{-}ind}(K, \gamma) : \mathsf{SP}_{\mathrm{A}}(A_{\mathrm{ref}})}$$

The code $\mathsf{non\text{-}ind}(K, \gamma)$ represents a non-inductive argument $x : K$, with the rest of the arguments given by $\gamma(x)$.

$$\frac{K : \mathsf{Set} \qquad \gamma : \mathsf{SP}_{\mathrm{A}}(A_{\mathrm{ref}} + K)}{\mathsf{A\text{-}ind}(K, \gamma) : \mathsf{SP}_{\mathrm{A}}(A_{\mathrm{ref}})}$$

The code $\mathsf{A\text{-}ind}(K, \gamma)$ represents an inductive argument with type $K \to A$, with the rest of the arguments given by γ. Notice that $\gamma : \mathsf{SP}_{\mathrm{A}}(A_{\mathrm{ref}} + K)$, so that the remaining arguments can refer to more elements in A (namely those introduced by the inductive argument).

$$\frac{K : \mathsf{Set} \qquad h_{\mathrm{index}} : K \to A_{\mathrm{ref}} \qquad \gamma : \mathsf{SP}_{\mathrm{A}}(A_{\mathrm{ref}})}{\mathsf{B\text{-}ind}(K, h_{\mathrm{index}}, \gamma) : \mathsf{SP}_{\mathrm{A}}(A_{\mathrm{ref}})}$$

Finally, the code $\mathsf{B\text{-}ind}(K, h_{\mathrm{index}}, \gamma)$ represents an inductive argument with type $(x : K) \to B(i(x))$, where the index $i(x)$ is determined by h_{index}, and the rest of the arguments are given by γ.

Example 4. The constructor $\triangleright : ((\Gamma : \mathsf{Ctxt}) \times \mathsf{Ty}(\Gamma)) \to \mathsf{Ctxt}$ is represented by the code

$$\gamma_{\triangleright} = \mathsf{A\text{-}ind}(\mathbf{1}, \mathsf{B\text{-}ind}(\mathbf{1}, \lambda(\star : \mathbf{1}) . \widehat{\Gamma}, \mathsf{nil})) \ ,$$

where $\widehat{\Gamma} = \mathsf{inr}(\star)$ is the representation of Γ in $A_{\mathrm{ref}} = \mathbf{0} + \mathbf{1}$. ■

We now define the decoding function $\mathsf{Arg}_{\mathrm{A}}$, which maps a code to the domain of the constructor it represents. In addition to a set X_{ref} and a code $\gamma : \mathsf{SP}_{\mathrm{A}}(X_{\mathrm{ref}})$, $\mathsf{Arg}_{\mathrm{A}}$ will take a set X and a family $Y : X \to \mathsf{Set}$ as arguments to use as A and B in the inductive arguments. These will later be instantiated by the sets defined inductive-inductively. We also require a function $\mathsf{rep}_{\mathrm{X}} : X_{\mathrm{ref}} \to X$ which we think of as mapping a "referable" element to the element it represents in X. All in all, $\mathsf{Arg}_{\mathrm{A}}$ has the following formation rule:

$$\frac{X_{\mathrm{ref}} : \mathsf{Set} \qquad \gamma : \mathsf{SP}_{\mathrm{A}}(X_{\mathrm{ref}}) \qquad X : \mathsf{Set} \qquad Y : X \to \mathsf{Set} \qquad \mathsf{rep}_{\mathrm{X}} : X_{\mathrm{ref}} \to X}{\mathsf{Arg}_{\mathrm{A}}(X_{\mathrm{ref}}, \gamma, X, Y, \mathsf{rep}_{\mathrm{X}}) : \mathsf{Set}}$$

Notice that if $\gamma : \mathsf{SP}^0_{\mathrm{A}}$, i.e. if $X_{\mathrm{ref}} = \mathbf{0}$, then we can choose $\mathsf{rep}_{\mathrm{X}} = !_X : \mathbf{0} \to X$ (indeed, extensionally, this is the only choice), so that we can define

$$\mathsf{Arg}^0_{\mathrm{A}} : \mathsf{SP}^0_{\mathrm{A}} \to (X : \mathsf{Set}) \to (Y : X \to \mathsf{Set}) \to \mathsf{Set}$$

by $\mathsf{Arg}^0_{\mathrm{A}}(\gamma, X, Y) = \mathsf{Arg}_{\mathrm{A}}(\mathbf{0}, \gamma, X, Y, !_X)$.

The definition of $\mathsf{Arg}_{\mathrm{A}}$ follows the informal description of what the different codes represent above:

$$\begin{aligned}
\mathsf{Arg}_{\mathrm{A}}(X_{\mathrm{ref}}, \mathsf{nil}, X, Y, \mathsf{rep}_{\mathrm{X}}) &= \mathbf{1} \\
\mathsf{Arg}_{\mathrm{A}}(X_{\mathrm{ref}}, \mathsf{non\text{-}ind}(K, \gamma), X, Y, \mathsf{rep}_{\mathrm{X}}) &= (x : K) \times \mathsf{Arg}_{\mathrm{A}}(X_{\mathrm{ref}}, \gamma(x), X, Y, \mathsf{rep}_{\mathrm{X}}) \\
\mathsf{Arg}_{\mathrm{A}}(X_{\mathrm{ref}}, \mathsf{A\text{-}ind}(K, \gamma), X, Y, \mathsf{rep}_{\mathrm{X}}) &= (j : K \to X) \times \mathsf{Arg}_{\mathrm{A}}(X_{\mathrm{ref}} + K, \gamma, X, Y, [\mathsf{rep}_{\mathrm{X}}, j]) \\
\mathsf{Arg}_{\mathrm{A}}(X_{\mathrm{ref}}, \mathsf{B\text{-}ind}(K, h_{\mathrm{index}}, \gamma), X, Y, \mathsf{rep}_{\mathrm{X}}) &= \\
&\quad ((x : K) \to Y((\mathsf{rep}_{\mathrm{X}} \circ h_{\mathrm{index}})(x))) \times \mathsf{Arg}_{\mathrm{A}}(X_{\mathrm{ref}}, \gamma, X, Y, \mathsf{rep}_{\mathrm{X}})
\end{aligned}$$

Example 5. Recall the code $\gamma_{\triangleright} = \mathsf{A\text{-}ind}(\mathbf{1}, \mathsf{B\text{-}ind}(\mathbf{1}, \lambda(\star : \mathbf{1}) . \mathsf{inr}(\star), \mathsf{nil}))$ for the constructor $\triangleright : ((\Gamma : \mathsf{Ctxt}) \times \mathsf{Ty}(\Gamma)) \to \mathsf{Ctxt}$. We have

$$\mathsf{Arg}^0_{\mathrm{A}}(\gamma_{\triangleright}, \mathsf{Ctxt}, \mathsf{Ty}) = (\Gamma : \mathbf{1} \to \mathsf{Ctxt}) \times (\mathbf{1} \to \mathsf{Ty}(\Gamma(\star))) \times \mathbf{1}$$

which, thanks to the η-rules for $\mathbf{1}$, $\times$ and $\to$, is isomorphic to the domain of $\triangleright$. ■

3.2.2 Towards descriptions of B

As we have seen in Example 1, it is important that the constructor $\mathsf{intro}_{\mathrm{B}}$ for the second set $B : A \to \mathsf{Set}$ can refer to the constructor $\mathsf{intro}_{\mathrm{A}}$ for the first set A. This means that inductive arguments might be of type $B(\mathsf{intro}_{\mathrm{A}}(\overline{a}))$ for some $\overline{a} : \mathsf{Arg}^0_{\mathrm{A}}(\gamma_{\mathrm{A}}, A, B)$ or even $B(\mathsf{intro}_{\mathrm{A}}(\ldots \mathsf{intro}_{\mathrm{A}} \ldots (\overline{a})))$ for some $\overline{a} : \mathsf{Arg}^0_{\mathrm{A}}(\gamma_{\mathrm{A}}, \ldots \mathsf{Arg}^0_{\mathrm{A}}(\gamma_{\mathrm{A}}, A, B) \ldots, B')$. Thus, we need to be able to represent such indices in the descriptions of the constructor $\mathsf{intro}_{\mathrm{B}}$.

First, it is no longer enough to only keep track of the referable elements X_{ref} of X – we need to be able to refer to elements of B as well, since they could be used as arguments to $\mathsf{intro}_{\mathrm{A}}$. We will represent the elements of Y we can refer to by a set Y_{ref}, together with functions $\mathsf{rep}_{\mathrm{index}} : Y_{\mathrm{ref}} \to X$ and $\mathsf{rep}_{\mathrm{Y}} : (x : Y_{\mathrm{ref}}) \to Y(\mathsf{rep}_{\mathrm{index}}(x))$; the function $\mathsf{rep}_{\mathrm{index}}$ gives the index of the represented element, and $\mathsf{rep}_{\mathrm{Y}}$ the actual element.

We want to represent elements in $\mathsf{Arg}^0_{\mathrm{A}}(\gamma_{\mathrm{A}}, X, Y)$. We claim that the elements in $\mathsf{Arg}^0_{\mathrm{A}}(\gamma_{\mathrm{A}}, X_{\mathrm{ref}} + Y_{\mathrm{ref}}, [\lambda x\,.\,\mathbf{0}, \lambda x\,.\,\mathbf{1}])$ are suitable for this purpose. To see this, first observe that we can define functions

$$f : X_{\mathrm{ref}} + Y_{\mathrm{ref}} \to X \ ,$$

$$g : (x : X_{\mathrm{ref}} + Y_{\mathrm{ref}}) \to [\lambda x\,.\,\mathbf{0}, \lambda x\,.\,\mathbf{1}](x) \to Y(f(x))$$

by $f = [\mathsf{rep}_{\mathrm{X}}, \mathsf{rep}_{\mathrm{index}}]$ and $g = [\lambda x\,.\,!, \lambda x \star\,.\,\mathsf{rep}_{\mathrm{Y}}(x)]$. Then, we can lift these functions to a function

$$\mathsf{Arg}^0_{\mathrm{A}}(\gamma_{\mathrm{A}}, f, g) : \mathsf{Arg}^0_{\mathrm{A}}(\gamma_{\mathrm{A}}, X_{\mathrm{ref}} + Y_{\mathrm{ref}}, [\lambda x\,.\,\mathbf{0}, \lambda x\,.\,\mathbf{1}]) \to \mathsf{Arg}^0_{\mathrm{A}}(\gamma_{\mathrm{A}}, X, Y)$$

by observing that $\mathsf{Arg}^0_{\mathrm{A}}(\gamma_{\mathrm{A}})$ is functorial:

Lemma 6. *For each $\gamma : \mathsf{SP}^0_{\mathrm{A}}$, $\mathsf{Arg}^0_{\mathrm{A}}(\gamma)$ extends to a functor from families of sets to sets, i.e. given $f : X \to X'$ and $g : (x : X) \to Y(x) \to Y'(f(x))$, one can define $\mathsf{Arg}^0_{\mathrm{A}}(\gamma, f, g) : \mathsf{Arg}^0_{\mathrm{A}}(\gamma, X, Y) \to \mathsf{Arg}^0_{\mathrm{A}}(\gamma, X', Y')$.*

Remark 7. In extensional type theory, one can also prove that $\mathsf{Arg}^0_{\mathrm{A}}(\gamma, f, g) : \mathsf{Arg}^0_{\mathrm{A}}(\gamma, X, Y) \to \mathsf{Arg}^0_{\mathrm{A}}(\gamma, X', Y')$ actually is a functor, i.e. that identities and compositions are preserved, but that will not be needed for the current development.

Proof. This is straightforward in extensional type theory. In intensional type theory without propositional identity types, we have to be more careful. The function $\mathsf{Arg}^0_{\mathrm{A}}(\gamma, f, g)$ is defined by induction over γ. In order to do this, we need to refer inductively to the case when X_{ref} is no longer $\mathbf{0}$. Hence, we need to consider the more general case where X, Y, X', Y', f and g have types as above, and $X_{\mathrm{ref}}, : \mathsf{Set}$, $\mathsf{rep}_{\mathrm{X}} : X_{\mathrm{ref}} \to X$, $\mathsf{rep}'_{\mathrm{X}} : X_{\mathrm{ref}} \to X'$. One expects the equality $f(\mathsf{rep}_{\mathrm{X}}(x)) = \mathsf{rep}'_{\mathrm{X}}(x)$ to hold for all $x : X_{\mathrm{ref}}$. In order to avoid the use of identity types, we state this in a form of Leibniz equality, specialised to the instance we actually need; we require a term

$$p : (x : X_{\mathrm{ref}}) \to Y'(f(\mathsf{rep}_{\mathrm{X}}(x))) \to Y'(\mathsf{rep}'_{\mathrm{X}}(x)) \ .$$

Thus we define

$$\mathsf{Arg}_{\mathrm{A}}(\gamma, f, g, p) : \mathsf{Arg}_{\mathrm{A}}(X_{\mathrm{ref}}, \gamma, X, Y, \mathsf{rep}_{\mathrm{X}}) \to \mathsf{Arg}_{\mathrm{A}}(X_{\mathrm{ref}}, \gamma, X', Y', \mathsf{rep}'_{\mathrm{X}})$$

by induction over γ:

$$\begin{aligned}
\mathsf{Arg}_{\mathrm{A}}(\mathsf{nil}, f, g, p, \star) &= \star \\
\mathsf{Arg}_{\mathrm{A}}(\mathsf{non\text{-}ind}(K, \gamma), f, g, p, \langle k, y\rangle) &= \langle k, \mathsf{Arg}_{\mathrm{A}}(\gamma(k), f, g, p, y)\rangle \\
\mathsf{Arg}_{\mathrm{A}}(\mathsf{A\text{-}ind}(K, \gamma), f, g, p, \langle j, y\rangle) &= \langle f \circ j, \mathsf{Arg}_{\mathrm{A}}(\gamma, f, g, [p, \lambda x\,.\,\mathsf{id}], y)\rangle \\
\mathsf{Arg}_{\mathrm{A}}(\mathsf{B\text{-}ind}(K, h_{\mathrm{index}}, \gamma), f, g, p, \langle j, y\rangle) &= \\
&\langle \lambda k\,.\,p(h_{\mathrm{index}}(k), g(\mathsf{rep}_{\mathrm{X}}(h_{\mathrm{index}}(k)), j(k))), \mathsf{Arg}_{\mathrm{A}}(\gamma, f, g, p, y)\rangle
\end{aligned}$$

Finally, we can define $\mathsf{Arg}^0_{\mathrm{A}}(\gamma, f, g) : \mathsf{Arg}^0_{\mathrm{A}}(\gamma, A, B) \to \mathsf{Arg}^0_{\mathrm{A}}(\gamma, A', B')$ by

$$\mathsf{Arg}^0_{\mathrm{A}}(\gamma, f, g) \coloneqq \mathsf{Arg}_{\mathrm{A}}(\gamma, f, g, !)\ .$$

□

Recall that we want to use the lemma to represent elements in $\mathsf{Arg}^0_{\mathrm{A}}(\gamma_{\mathrm{A}}, X, Y)$ by elements in $\mathsf{Arg}^0_{\mathrm{A}}(\gamma_{\mathrm{A}}, X_{\mathrm{ref}} + Y_{\mathrm{ref}}, [\lambda x\,.\,\mathbf{0}, \lambda x\,.\,\mathbf{1}])$. We can actually do better, and represent arbitrarily terms built from elements in X and Y with the use of a constructor $\mathsf{intro}_{\mathrm{A}} : \mathsf{Arg}^0_{\mathrm{A}}(\gamma_{\mathrm{A}}, X, Y) \to X$. For this, define the set $\mathsf{A\text{-}Term}(\gamma_{\mathrm{A}}, X_{\mathrm{ref}}, Y_{\mathrm{ref}})$ of terms "built from $\mathsf{intro}_{\mathrm{A}}$, X_{ref} and Y_{ref}" with introduction rules

$$\frac{x : X_{\mathrm{ref}}}{\mathsf{a}_{\mathrm{ref}}(x) : \mathsf{A\text{-}Term}(\gamma_{\mathrm{A}}, X_{\mathrm{ref}}, Y_{\mathrm{ref}})}$$

$$\frac{x : Y_{\mathrm{ref}}}{\mathsf{b}_{\mathrm{ref}}(x) : \mathsf{A\text{-}Term}(\gamma_{\mathrm{A}}, X_{\mathrm{ref}}, Y_{\mathrm{ref}})}$$

$$\frac{x : \mathsf{Arg}^0_{\mathrm{A}}(\gamma_{\mathrm{A}}, \mathsf{A\text{-}Term}(\gamma_{\mathrm{A}}, X_{\mathrm{ref}}, Y_{\mathrm{ref}}), \mathsf{B\text{-}Term}(\gamma_{\mathrm{A}}, X_{\mathrm{ref}}, Y_{\mathrm{ref}}))}{\mathsf{arg}(x) : \mathsf{A\text{-}Term}(\gamma_{\mathrm{A}}, X_{\mathrm{ref}}, Y_{\mathrm{ref}})}$$

Here, $\mathsf{B\text{-}Term}(\gamma_{\mathrm{A}}, X_{\mathrm{ref}}, Y_{\mathrm{ref}}) : \mathsf{A\text{-}Term}(\gamma_{\mathrm{A}}, X_{\mathrm{ref}}, Y_{\mathrm{ref}}) \to \mathsf{Set}$ is defined by

$$\begin{aligned}
\mathsf{B\text{-}Term}(\gamma_{\mathrm{A}}, X_{\mathrm{ref}}, Y_{\mathrm{ref}}, \mathsf{a}_{\mathrm{ref}}(x)) &= \mathbf{0} \\
\mathsf{B\text{-}Term}(\gamma_{\mathrm{A}}, X_{\mathrm{ref}}, Y_{\mathrm{ref}}, \mathsf{b}_{\mathrm{ref}}(x)) &= \mathbf{1} \\
\mathsf{B\text{-}Term}(\gamma_{\mathrm{A}}, X_{\mathrm{ref}}, Y_{\mathrm{ref}}, \mathsf{arg}(x)) &= \mathbf{0}
\end{aligned}$$

Note that this is formally an inductive-recursive definition. The intuition behind the definition of $\mathsf{B\text{-}Term}$ is that all elements of Y we know are represented in Y_{ref}, and only in Y_{ref}.

All elements in $\mathsf{A\text{-}Term}(\gamma_{\mathrm{A}}, X_{\mathrm{ref}}, Y_{\mathrm{ref}})$ represents elements in X, given that we have a function $\mathsf{intro}_{\mathrm{A}} : \mathsf{Arg}^0_{\mathrm{A}}(\gamma_{\mathrm{A}}, X, Y) \to X$ and the elements of X_{ref} and Y_{ref} represents elements of X and Y respectively (i.e. we have $\mathsf{rep}_{\mathrm{X}} : X_{\mathrm{ref}} \to X$, $\mathsf{rep}_{\mathrm{index}} : Y_{\mathrm{ref}} \to X$ and $\mathsf{rep}_{\mathrm{Y}} : (x : Y_{\mathrm{ref}}) \to Y(\mathsf{rep}_{\mathrm{index}}(x))$). Formally, we can simultaneously define the following two functions:

$$\frac{\begin{array}{ccc} & & \mathsf{rep}_{\mathrm{X}} : X_{\mathrm{ref}} \to X \\ & & \mathsf{rep}_{\mathrm{index}} : Y_{\mathrm{ref}} \to X \\ \gamma_{\mathrm{A}} : \mathsf{SP}^0_{\mathrm{A}} & \mathsf{intro}_{\mathrm{A}} : \mathsf{Arg}^0_{\mathrm{A}}(\gamma_{\mathrm{A}}, X, Y) \to X & \mathsf{rep}_{\mathrm{Y}} : (x : Y_{\mathrm{ref}}) \to Y(\mathsf{rep}_{\mathrm{index}}(x)) \end{array}}{\begin{array}{c} \overline{\mathsf{rep}_{\mathrm{A}}}(\ldots) : \mathsf{A\text{-}Term}(\gamma_{\mathrm{A}}, X_{\mathrm{ref}}, Y_{\mathrm{ref}}) \to X \\ \overline{\mathsf{rep}_{\mathrm{B}}}(\ldots) : (x : \mathsf{A\text{-}Term}(\gamma_{\mathrm{A}}, X_{\mathrm{ref}}, Y_{\mathrm{ref}})) \to \mathsf{B\text{-}Term}(\gamma_{\mathrm{A}}, X_{\mathrm{ref}}, Y_{\mathrm{ref}}, x) \to Y(\overline{\mathsf{rep}_{\mathrm{A}}}(\ldots, x)) \end{array}}$$

The definition of $\overline{\mathsf{rep}_\mathrm{A}}$ is straightforward. The interesting case is $\mathsf{arg}(x)$, where we make use of the constructor $\mathsf{intro}_\mathrm{A}$, the functoriality of $\mathsf{Arg}^0_\mathrm{A}$ and the mutually defined $\overline{\mathsf{rep}_\mathrm{B}}$:

$$\overline{\mathsf{rep}_\mathrm{A}}(\gamma_\mathrm{A}, \mathsf{intro}_\mathrm{A}, \mathsf{rep}_\mathrm{X}, \mathsf{rep}_\mathrm{index}, \mathsf{rep}_\mathrm{Y}, \mathsf{a}_\mathrm{ref}(x)) = \mathsf{rep}_\mathrm{X}(x)$$
$$\overline{\mathsf{rep}_\mathrm{A}}(\gamma_\mathrm{A}, \mathsf{intro}_\mathrm{A}, \mathsf{rep}_\mathrm{X}, \mathsf{rep}_\mathrm{index}, \mathsf{rep}_\mathrm{Y}, \mathsf{b}_\mathrm{ref}(x)) = \mathsf{rep}_\mathrm{index}(x)$$
$$\overline{\mathsf{rep}_\mathrm{A}}(\gamma_\mathrm{A}, \mathsf{intro}_\mathrm{A}, \mathsf{rep}_\mathrm{X}, \mathsf{rep}_\mathrm{index}, \mathsf{rep}_\mathrm{Y}, \mathsf{arg}(x)) = \mathsf{intro}_\mathrm{A}(\mathsf{Arg}^0_\mathrm{A}(\gamma_\mathrm{A}, \overline{\mathsf{rep}_\mathrm{A}}(\ldots), \overline{\mathsf{rep}_\mathrm{B}}(\ldots), x))$$

The simultaneously defined $\overline{\mathsf{rep}_\mathrm{B}}$ is very simple:

$$\overline{\mathsf{rep}_\mathrm{B}}(\gamma_\mathrm{A}, \mathsf{intro}_\mathrm{A}, \mathsf{rep}_\mathrm{X}, \mathsf{rep}_\mathrm{index}, \mathsf{rep}_\mathrm{Y}, \mathsf{a}_\mathrm{ref}(x), y) = !(y)$$
$$\overline{\mathsf{rep}_\mathrm{B}}(\gamma_\mathrm{A}, \mathsf{intro}_\mathrm{A}, \mathsf{rep}_\mathrm{X}, \mathsf{rep}_\mathrm{index}, \mathsf{rep}_\mathrm{Y}, \mathsf{b}_\mathrm{ref}(x), \star) = \mathsf{rep}_\mathrm{Y}(y)$$
$$\overline{\mathsf{rep}_\mathrm{B}}(\gamma_\mathrm{A}, \mathsf{intro}_\mathrm{A}, \mathsf{rep}_\mathrm{X}, \mathsf{rep}_\mathrm{index}, \mathsf{rep}_\mathrm{Y}, \mathsf{arg}(x), y) = !(y)$$

Example 8. We define some terms in $\mathsf{A\text{-}Term}(\gamma_\triangleright, X_\mathrm{ref}, Y_\mathrm{ref})$, where

$$\gamma_\triangleright = \mathsf{A\text{-}ind}(\mathbf{1}, \mathsf{B\text{-}ind}(\mathbf{1}, \lambda(\star : \mathbf{1})\,.\,\mathsf{inr}(\star), \mathsf{nil}))$$

is the code for the constructor

$$\triangleright : \big((\Gamma : \mathbf{1} \to A) \times (\mathbf{1} \to B(\Gamma(\star))) \times \mathbf{1}\big) \to A \ .$$

Suppose that we have $\hat{a} : X_\mathrm{ref}$ with $\mathsf{rep}_\mathrm{X}(\hat{a}) = a : A$ and $\hat{b} : Y_\mathrm{ref}$ with $\mathsf{rep}_\mathrm{index}(\hat{b}) = a$ and $\mathsf{rep}_\mathrm{Y}(\hat{b}) = b : B(a)$. We then have

- $\mathsf{a}_\mathrm{ref}(\hat{a}) : \mathsf{A\text{-}Term}(\gamma_\triangleright, X_\mathrm{ref}, Y_\mathrm{ref})$ with $\overline{\mathsf{rep}_\mathrm{A}}(\gamma_\triangleright, \triangleright, \ldots, \hat{a}) = a$ (so elements from X_ref are terms).
- $\mathsf{b}_\mathrm{ref}(\hat{b}) : \mathsf{A\text{-}Term}(\gamma_\triangleright, X_\mathrm{ref}, Y_\mathrm{ref})$ with $\overline{\mathsf{rep}_\mathrm{A}}(\gamma_\triangleright, \triangleright, \ldots, \mathsf{b}_\mathrm{ref}(\hat{b})) = a$ (so elements from Y_ref are terms, representing the index of the element in B they represent). Furthermore $\overline{\mathsf{rep}_\mathrm{B}}(\gamma_\triangleright, \triangleright, \ldots, \mathsf{b}_\mathrm{ref}(\hat{b}), \star) = b$.
- $\widehat{a, b} \coloneqq \mathsf{arg}(\langle(\lambda \star\,.\,\mathsf{b}_\mathrm{ref}(\hat{b})), \langle(\lambda \star\,.\,\star), \star\rangle\rangle) : \mathsf{A\text{-}Term}(\gamma_\triangleright, X_\mathrm{ref}, Y_\mathrm{ref})$ with

$$\overline{\mathsf{rep}_\mathrm{A}}(\gamma_\triangleright, \triangleright, \ldots, \widehat{a, b}) = (\mathsf{rep}_\mathrm{index}(\hat{b})) \triangleright (\mathsf{rep}_\mathrm{Y}(\hat{b})) = a \triangleright b \ .$$

■

3.2.3 The universe SP^0_B of descriptions of B

We now introduce the universe SP_B of descriptions for B. It has formation rule

$$\frac{A_\mathrm{ref}, B_\mathrm{ref} : \mathsf{Set} \qquad \gamma_\mathrm{A} : \mathsf{SP}^0_\mathrm{A}}{\mathsf{SP}_\mathrm{B}(A_\mathrm{ref}, B_\mathrm{ref}, \gamma_\mathrm{A}) : \mathsf{Type}}$$

Again, we are interested in codes which initially do not refer to any elements and define $\mathsf{SP}^0_{\mathrm{B}} : \mathsf{SP}^0_{\mathrm{A}} \to \mathsf{Type}$ by $\mathsf{SP}^0_{\mathrm{B}}(\gamma_{\mathrm{A}}) \coloneqq \mathsf{SP}_{\mathrm{B}}(\mathbf{0}, \mathbf{0}, \gamma_{\mathrm{A}})$.

The introduction rules for SP_{B} are similar to the ones for SP_{A}. However, we now need to specify an index for the codomain of the constructor, and indices for arguments inductive in B can be arbitrary terms built up from $\mathsf{intro}_{\mathrm{A}}$ and elements we can refer to.

$$\frac{a : \mathsf{A\text{-}Term}(\gamma_{\mathrm{A}}, A_{\mathrm{ref}}, B_{\mathrm{ref}})}{\mathsf{nil}(a) : \mathsf{SP}_{\mathrm{B}}(A_{\mathrm{ref}}, B_{\mathrm{ref}}, \gamma_{\mathrm{A}})}$$

The code $\mathsf{nil}(\widehat{a})$ represents a trivial constructor $c : \mathbf{1} \to B(a)$ (a base case), where the index a is encoded by $\widehat{a} : \mathsf{A\text{-}Term}(\gamma_{\mathrm{A}}, A_{\mathrm{ref}}, B_{\mathrm{ref}})$.

$$\frac{K : \mathsf{Set} \qquad \gamma : K \to \mathsf{SP}_{\mathrm{B}}(A_{\mathrm{ref}}, B_{\mathrm{ref}}, \gamma_{\mathrm{A}})}{\mathsf{non\text{-}ind}(K, \gamma) : \mathsf{SP}_{\mathrm{B}}(A_{\mathrm{ref}}, B_{\mathrm{ref}}, \gamma_{\mathrm{A}}))}$$

The code $\mathsf{non\text{-}ind}(K, \gamma)$ represents a non-inductive argument $x : K$, with the rest of the arguments given by $\gamma(x)$.

$$\frac{K : \mathsf{Set} \qquad \gamma : \mathsf{SP}_{\mathrm{B}}(A_{\mathrm{ref}} + K, B_{\mathrm{ref}}, \gamma_{\mathrm{A}})}{\mathsf{A\text{-}ind}(K, \gamma) : \mathsf{SP}_{\mathrm{B}}(A_{\mathrm{ref}}, B_{\mathrm{ref}}, \gamma_{\mathrm{A}})}$$

The code $\mathsf{A\text{-}ind}(K, \gamma)$ represents an inductive argument with type $K \to A$, with the rest of the arguments given by γ.

$$\frac{K : \mathsf{Set} \qquad h_{\mathrm{index}} : K \to \mathsf{A\text{-}Term}(A_{\mathrm{ref}}, B_{\mathrm{ref}}, \gamma_{\mathrm{A}}) \qquad \gamma : \mathsf{SP}_{\mathrm{B}}(A_{\mathrm{ref}}, B_{\mathrm{ref}} + K, \gamma_{\mathrm{A}})}{\mathsf{B\text{-}ind}(K, h_{\mathrm{index}}, \gamma) : \mathsf{SP}_{\mathrm{B}}(A_{\mathrm{ref}}, B_{\mathrm{ref}}, \gamma_{\mathrm{A}})}$$

At last, the code $\mathsf{B\text{-}ind}(K, h_{\mathrm{index}}, \gamma)$ represents an inductive argument with type $(x : K) \to B(i(x))$, where the index $i(x)$ is determined by h_{index}, and the rest of the arguments are given by γ. Notice how the index is now encoded by arbitrary terms in $\mathsf{A\text{-}Term}(A_{\mathrm{ref}}, B_{\mathrm{ref}}, \gamma_{\mathrm{A}})$.

Example 9. The constructor

$$\Pi : \big((\Gamma : \mathsf{Ctxt}) \times (\sigma : \mathsf{Ty}(\Gamma)) \times \mathsf{Ty}(\Gamma \triangleright \sigma)\big) \to \mathsf{Ty}(\Gamma)$$

is represented by the code

$$\gamma_{\Pi} = \mathsf{A\text{-}ind}(\mathbf{1}, \mathsf{B\text{-}ind}(\mathbf{1}, \lambda \star\ .\widehat{\Gamma}, \mathsf{B\text{-}ind}(\mathbf{1}, \lambda \star\ .\widehat{\mathrm{in}\langle \Gamma, \sigma \rangle}, \mathsf{nil}(\widehat{\Gamma}))))$$

where $\widehat{\Gamma} = \mathsf{a}_{\mathrm{ref}}(\mathsf{inr}(\star))$ is the element representing the first argument $\Gamma : \mathsf{Ctxt}$ and $\widehat{\mathrm{in}\langle \Gamma, \sigma \rangle} = \mathsf{arg}(\langle(\lambda \star .\mathsf{b}_{\mathrm{ref}}(\mathsf{inr}(\star))), \langle \lambda \star .\star, \star \rangle\rangle)$ is the element representing $\Gamma \triangleright \sigma$. ■

The definition of $\mathsf{Arg}_{\mathrm{B}}$ should now not come as a surprise. First, we have a formation rule:

$$\frac{\begin{array}{ccc} \gamma_{\mathrm{A}} : \mathsf{SP}^0_{\mathrm{A}} & \begin{array}{c} X : \mathsf{Set} \\ Y : X \to \mathsf{Set} \end{array} & \begin{array}{c} \mathsf{rep}_{\mathrm{X}} : X_{\mathrm{ref}} \to X \\ \mathsf{rep}_{\mathrm{index}} : Y_{\mathrm{ref}} \to X \end{array} \\ X_{\mathrm{ref}}, Y_{\mathrm{ref}} : \mathsf{Set} & \mathsf{intro}_{\mathrm{A}} : \mathsf{Arg}_{\mathrm{A}}(\gamma_{\mathrm{A}}, X, Y) \to X & \mathsf{rep}_{\mathrm{Y}} : (x : Y_{\mathrm{ref}}) \to Y(\mathsf{rep}_{\mathrm{index}}(x)) \qquad \gamma : \mathsf{SP}_{\mathrm{B}}(X_{\mathrm{ref}}, Y_{\mathrm{ref}}, \gamma_{\mathrm{A}}) \end{array}}{\mathsf{Arg}_{\mathrm{B}}(X_{\mathrm{ref}}, Y_{\mathrm{ref}}, \gamma_{\mathrm{A}}, X, Y, \mathsf{intro}_{\mathrm{A}}, \mathsf{rep}_{\mathrm{X}}, \mathsf{rep}_{\mathrm{index}}, \mathsf{rep}_{\mathrm{Y}}, \gamma) : \mathsf{Set}}$$

Once again, the definition is simpler for codes in $\mathsf{SP}^0_{\mathrm{B}}(\gamma_{\mathrm{A}})$:

$$\mathsf{Arg}^0_{\mathrm{B}}(\gamma_{\mathrm{A}}, X, Y, \mathsf{intro}_{\mathrm{A}}, \gamma) \coloneqq \mathsf{Arg}_{\mathrm{B}}(\mathbf{0}, \mathbf{0}, \gamma_{\mathrm{A}}, X, Y, \mathsf{intro}_{\mathrm{A}}, !, !, !, \gamma)$$

We define:

$$\begin{aligned}
&\mathsf{Arg}_{\mathrm{B}}(X_{\mathrm{ref}}, Y_{\mathrm{ref}}, \gamma_{\mathrm{A}}, X, Y, \mathsf{intro}_{\mathrm{A}}, \mathsf{rep}_{\mathrm{X}}, \mathsf{rep}_{\mathrm{index}}, \mathsf{rep}_{\mathrm{Y}}, \mathsf{nil}(a)) = \mathbf{1}\\
&\mathsf{Arg}_{\mathrm{B}}(X_{\mathrm{ref}}, Y_{\mathrm{ref}}, \gamma_{\mathrm{A}}, X, Y, \mathsf{intro}_{\mathrm{A}}, \mathsf{rep}_{\mathrm{X}}, \mathsf{rep}_{\mathrm{index}}, \mathsf{rep}_{\mathrm{Y}}, \mathsf{non\text{-}ind}(K, \gamma))\\
&\quad = (x : K) \times \mathsf{Arg}_{\mathrm{B}}(X_{\mathrm{ref}}, Y_{\mathrm{ref}}, \gamma_{\mathrm{A}}, X, Y, \mathsf{intro}_{\mathrm{A}}, \mathsf{rep}_{\mathrm{X}}, \mathsf{rep}_{\mathrm{index}}, \mathsf{rep}_{\mathrm{Y}}, \gamma(x))\\
&\mathsf{Arg}_{\mathrm{B}}(X_{\mathrm{ref}}, Y_{\mathrm{ref}}, \gamma_{\mathrm{A}}, X, Y, \mathsf{intro}_{\mathrm{A}}, \mathsf{rep}_{\mathrm{X}}, \mathsf{rep}_{\mathrm{index}}, \mathsf{rep}_{\mathrm{Y}}, \mathsf{A\text{-}ind}(K, \gamma))\\
&\quad = (j : K \to X) \times \mathsf{Arg}_{\mathrm{B}}(X_{\mathrm{ref}} + K, Y_{\mathrm{ref}}, \gamma_{\mathrm{A}}, X, Y, \mathsf{intro}_{\mathrm{A}}, [\mathsf{rep}_{\mathrm{X}}, j], \mathsf{rep}_{\mathrm{index}}, \mathsf{rep}_{\mathrm{Y}}, \gamma)\\
&\mathsf{Arg}_{\mathrm{B}}(X_{\mathrm{ref}}, Y_{\mathrm{ref}}, \gamma_{\mathrm{A}}, X, Y, \mathsf{intro}_{\mathrm{A}}, \mathsf{rep}_{\mathrm{X}}, \mathsf{rep}_{\mathrm{index}}, \mathsf{rep}_{\mathrm{Y}}, \mathsf{B\text{-}ind}(K, h_{\mathrm{index}}, \gamma))\\
&\quad = (j : (x : K) \to Y((\overline{\mathsf{rep}_{\mathrm{A}}}(\gamma_{\mathrm{A}}, \mathsf{intro}_{\mathrm{A}}, \mathsf{rep}_{\mathrm{X}}, \mathsf{rep}_{\mathrm{index}}, \mathsf{rep}_{\mathrm{Y}}) \circ h_{\mathrm{index}})(x))) \times\\
&\qquad \mathsf{Arg}_{\mathrm{B}}(X_{\mathrm{ref}}, Y_{\mathrm{ref}} + K, \gamma_{\mathrm{A}}, X, Y, \mathsf{intro}_{\mathrm{A}}, \mathsf{rep}_{\mathrm{X}}, [\mathsf{rep}_{\mathrm{index}}, \overline{\mathsf{rep}_{\mathrm{A}}}(\ldots) \circ h_{\mathrm{index}}], [\mathsf{rep}_{\mathrm{Y}}, j], \gamma)
\end{aligned}$$

Finally, we need the function $\mathsf{Index}^0_{\mathrm{B}}(\ldots) : \mathsf{Arg}_{\mathrm{B}}(\gamma_{\mathrm{A}}, \gamma_{\mathrm{B}}, X, Y, \mathsf{intro}_{\mathrm{A}}) \to X$ which to each $b : \mathsf{Arg}_{\mathrm{B}}(\gamma_{\mathrm{A}}, \gamma_{\mathrm{B}}, X, Y, \mathsf{intro}_{\mathrm{A}})$ assigns an index $a : X$ such that the element constructed from b is in $Y(a)$.

$$\frac{\begin{array}{ccccc}\gamma_{\mathrm{A}} : \mathsf{SP}^0_{\mathrm{A}} & & X : \mathsf{Set} & & \mathsf{rep}_{\mathrm{X}} : X_{\mathrm{ref}} \to X\\ & & Y : X \to \mathsf{Set} & & \mathsf{rep}_{\mathrm{index}} : Y_{\mathrm{ref}} \to X\\ X_{\mathrm{ref}}, Y_{\mathrm{ref}} : \mathsf{Set} & & \mathsf{intro}_{\mathrm{A}} : \mathsf{Arg}_{\mathrm{A}}(\gamma_{\mathrm{A}}, X, Y) \to X & & \mathsf{rep}_{\mathrm{Y}} : (x : Y_{\mathrm{ref}}) \to Y(\mathsf{rep}_{\mathrm{index}}(x)) \qquad \gamma : \mathsf{SP}_{\mathrm{B}}(X_{\mathrm{ref}}, Y_{\mathrm{ref}}, \gamma_{\mathrm{A}})\end{array}}{\mathsf{Index}_{\mathrm{B}}(X_{\mathrm{ref}}, Y_{\mathrm{ref}}, \gamma_{\mathrm{A}}, X, Y, \mathsf{intro}_{\mathrm{A}}, \mathsf{rep}_{\mathrm{X}}, \mathsf{rep}_{\mathrm{index}}, \mathsf{rep}_{\mathrm{Y}}, \gamma) : \mathsf{Arg}_{\mathrm{B}}(\ldots) \to X}$$

As we usually do, we define $\mathsf{Index}^0_{\mathrm{B}} : \mathsf{Arg}^0_{\mathrm{B}}(\gamma_{\mathrm{A}}, X, Y, \mathsf{intro}_{\mathrm{A}}, \gamma_{\mathrm{B}}) \to X$ by $\mathsf{Index}^0_{\mathrm{B}}(\gamma_{\mathrm{A}}, X, Y, \mathsf{intro}_{\mathrm{A}}, \gamma_{\mathrm{B}}) :$ $= \mathsf{Index}_{\mathrm{B}}(\mathbf{0}, \mathbf{0}, \gamma_{\mathrm{A}}, X, Y, \mathsf{intro}_{\mathrm{A}}, !, !, !, \gamma_{\mathrm{B}})$.

For the equations, we will suppress all inferable arguments:

$$\begin{aligned}
\mathsf{Index}_{\mathrm{B}}(\ldots, \mathsf{nil}(a), \star) &= \overline{\mathsf{rep}_{\mathrm{A}}}(\ldots, a)\\
\mathsf{Index}_{\mathrm{B}}(\ldots, \mathsf{non\text{-}ind}(K, \gamma), \langle k, y\rangle) &= \mathsf{Index}_{\mathrm{B}}(\ldots, \gamma(k), y)\\
\mathsf{Index}_{\mathrm{B}}(\ldots, \mathsf{A\text{-}ind}(K, \gamma), \langle j, y\rangle) &= \mathsf{Index}_{\mathrm{B}}(\ldots, \gamma, y)\\
\mathsf{Index}_{\mathrm{B}}(\ldots, \mathsf{B\text{-}ind}(K, h_{\mathrm{index}}, \gamma), \langle j, y\rangle) &= \mathsf{Index}_{\mathrm{B}}(\ldots, \gamma, y)
\end{aligned}$$

Example 10. The constructor $\Pi : \big((\Gamma : \mathsf{Ctxt}) \times (\sigma : \mathsf{Ty}(\Gamma)) \times \mathsf{Ty}(\Gamma \rhd \sigma)\big) \to \mathsf{Ty}(\Gamma)$ from Example 1 is represented by the code

$$\gamma_\Pi = \mathsf{A\text{-}ind}(\mathbf{1}, \mathsf{B\text{-}ind}(\mathbf{1}, (\lambda \star . \widehat{\Gamma}), \mathsf{B\text{-}ind}(\mathbf{1}, (\lambda \star . \widehat{\Gamma \rhd \sigma}, \mathsf{nil}(\widehat{\Gamma}))))) : \mathsf{SP}^0_{\mathrm{B}}(\gamma_\rhd) \ ,$$

where $\widehat{\Gamma} = \mathsf{a}_{\mathrm{ref}}(\mathsf{inr}(\star)) : \mathsf{A\text{-}Term}(\mathbf{0} + \mathbf{1}, \mathbf{0}, \gamma_\rhd)$ and

$$\widehat{\Gamma \rhd \sigma} = \mathsf{arg}(\langle(\lambda \star . \mathsf{b}_{\mathrm{ref}}(\mathsf{inr}(\star))), \langle \lambda \star . \star, \star\rangle\rangle) : \mathsf{A\text{-}Term}(\mathbf{0} + \mathbf{1}, \mathbf{0} + \mathbf{1}, \gamma_\rhd) \ .$$

We have

$$\begin{aligned}
&\mathsf{Arg}^0_{\mathrm{B}}(\gamma_\rhd, \mathsf{Ctxt}, \mathsf{Ty}, \rhd, \gamma_\Pi) =\\
&\qquad (\Gamma : \mathbf{1} \to \mathsf{Ctxt}) \times (\sigma : \mathbf{1} \to \mathsf{Ty}(\Gamma(\star))) \times (\mathbf{1} \to \mathsf{Ty}(\Gamma(\star)) \rhd \sigma(\star)) \times \mathbf{1}
\end{aligned}$$

and $\mathsf{Index}^0_{\mathrm{B}}(\gamma_\rhd, \mathsf{Ctxt}, \mathsf{Ty}, \rhd, \gamma_\Pi, \langle \Gamma, \sigma, \tau, \star\rangle) = \Gamma(\star)$.

■

3.2.4 Formation and introduction rules

We are now ready to give the formation and introduction rules for A and B. They all have the common premises $\gamma_{\mathrm{A}} : \mathsf{SP}^0_{\mathrm{A}}$, $\gamma_{\mathrm{B}} : \mathsf{SP}^0_{\mathrm{B}}(\gamma_{\mathrm{A}})$, which will be omitted.

Formation rules:

$$A_{\gamma_{\mathrm{A}},\gamma_{\mathrm{B}}} : \mathsf{Set} \qquad B_{\gamma_{\mathrm{A}},\gamma_{\mathrm{B}}} : A_{\gamma_{\mathrm{A}},\gamma_{\mathrm{B}}} \to \mathsf{Set}$$

Introduction rule for $A_{\gamma_{\mathrm{A}},\gamma_{\mathrm{B}}}$:

$$\frac{a : \mathsf{Arg}^0_{\mathrm{A}}(\gamma_{\mathrm{A}}, A_{\gamma_{\mathrm{A}},\gamma_{\mathrm{B}}}, B_{\gamma_{\mathrm{A}},\gamma_{\mathrm{B}}})}{\mathsf{intro}_{A_{\gamma_{\mathrm{A}},\gamma_{\mathrm{B}}}}(a) : A_{\gamma_{\mathrm{A}},\gamma_{\mathrm{B}}}}$$

Introduction rule for $B_{\gamma_{\mathrm{A}},\gamma_{\mathrm{B}}}$:

$$\frac{a : \mathsf{Arg}^0_{\mathrm{B}}(\gamma_{\mathrm{A}}, A_{\gamma_{\mathrm{A}},\gamma_{\mathrm{B}}}, B_{\gamma_{\mathrm{A}},\gamma_{\mathrm{B}}}, \mathsf{intro}_{A_{\gamma_{\mathrm{A}},\gamma_{\mathrm{B}}}}, \gamma_{\mathrm{B}})}{\mathsf{intro}_{B_{\gamma_{\mathrm{A}},\gamma_{\mathrm{B}}}}(a) : B_{\gamma_{\mathrm{A}},\gamma_{\mathrm{B}}}(\mathsf{Index}^0_{\mathrm{B}}(\gamma_{\mathrm{A}}, A_{\gamma_{\mathrm{A}},\gamma_{\mathrm{B}}}, B_{\gamma_{\mathrm{A}},\gamma_{\mathrm{B}}}, \mathsf{intro}_{A_{\gamma_{\mathrm{A}},\gamma_{\mathrm{B}}}}, \gamma_{\mathrm{B}}, a))}$$

3.2.5 Elimination rules by example

Elimination rules can also be formulated [AMNFS11]. Here, we just give the elimination rules for the data type of sorted lists (Example 2) as an example, and show how one can use them to define a function which inserts a number into a sorted list.[3]

Example 11. The elimination rules for sorted lists and the $\leq_{\mathrm{L}}$ predicate state that functions $\mathsf{elim}_{\mathsf{SortedList}}$ and $\mathsf{elim}_{\leq_{\mathrm{L}}}$ with the following types exist:

$$\begin{aligned}
\mathsf{elim}_{\mathsf{SortedList}} : {} & (P : \mathsf{SortedList} \to \mathsf{Set}) \to \\
& (Q : (n : \mathbb{N}) \to (\ell : \mathsf{SortedList}) \to n \leq_{\mathrm{L}} \ell \to P(\ell) \to \mathsf{Set}) \to \\
& (\mathsf{step}_{\mathsf{nil}} : P(\mathsf{nil})) \to \\
& \big(\mathsf{step}_{\mathsf{cons}} : (n : \mathbb{N}) \to (\ell : \mathsf{SortedList}) \to (p : n \leq_{\mathrm{L}} \ell) \to (\widetilde{\ell} : P(\ell)) \\
& \qquad \to Q(n, \ell, p, \widetilde{\ell}) \to P(\mathsf{cons}(n, \ell, p))\big) \to \\
& \big(\mathsf{step}_{\mathsf{triv}} : (m : \mathbb{N}) \to Q(m, \mathsf{nil}, \mathsf{triv}_n, \mathsf{step}_{\mathsf{nil}})\big) \to \\
& \big(\mathsf{step}_{\ll\cdot\gg} : (m : \mathbb{N}) \to (n : \mathbb{N}) \to (\ell : \mathsf{SortedList}) \to (p : n \leq_{\mathrm{L}} \ell) \\
& \qquad \to (q : m \leq n) \to (p' : m \leq_{\mathrm{L}} \ell) \to (\widetilde{\ell} : P(\ell)) \\
& \qquad \to (\widetilde{p} : Q(n, \ell, p, \widetilde{\ell})) \to (\widetilde{p'} : Q(m, \ell, p', \widetilde{\ell})) \\
& \qquad \to Q(m, \mathsf{cons}(n, \ell, p), \ll q, p' \gg, \mathsf{step}_{\mathsf{cons}}(n, \ell, p, \widetilde{\ell}, \widetilde{p}))\big) \to \\
& (\ell : \mathsf{SortedList}) \to P(\ell) \ ,
\end{aligned}$$

[3] The inductive-inductive definition of the data type of sorted lists falls outside the axiomatisation presented in this article, as remarked at the end of Section 1.1. We still include this example, as it shows the use of elimination rules in a real computer science example.

$$
\begin{aligned}
\mathsf{elim}_{\leq_{\mathrm{L}}} : {} & (P : \mathsf{SortedList} \to \mathsf{Set}) \to \\
& (Q : (n : \mathbb{N}) \to (\ell : \mathsf{SortedList}) \to n \leq_{\mathrm{L}} \ell \to P(\ell) \to \mathsf{Set}) \to \\
& (\mathsf{step}_{\mathsf{nil}} : \dots) \to \\
& \big(\mathsf{step}_{\mathsf{cons}} : \dots\big) \to \\
& \big(\mathsf{step}_{\mathsf{triv}} : \dots\big) \to \\
& \big(\mathsf{step}_{\ll\cdot\gg} : \dots\big) \to \\
& (n : \mathbb{N}) \to (\ell : \mathsf{SortedList}) \to (p : n \leq_{\mathrm{L}} \ell) \\
& \qquad \to Q(n, \ell, p, \mathsf{elim}_{\mathsf{SortedList}}(\dots, \ell)) \ .
\end{aligned}
$$

with computation rules

$$
\mathsf{elim}_{\mathsf{SortedList}}(P, Q, \mathsf{step}_{\mathsf{nil}}, \mathsf{step}_{\mathsf{cons}}, \mathsf{step}_{\mathsf{triv}}, \mathsf{step}_{\ll\cdot\gg}, \mathsf{nil}) = \mathsf{step}_{\mathsf{nil}}
$$

and

$$
\begin{aligned}
& \mathsf{elim}_{\mathsf{SortedList}}(P, Q, \mathsf{step}_{\mathsf{nil}}, \mathsf{step}_{\mathsf{cons}}, \mathsf{step}_{\mathsf{triv}}, \mathsf{step}_{\ll\cdot\gg}, \mathsf{cons}(n, \ell, p)) \\
& \qquad = \mathsf{step}_{\mathsf{cons}}(n, \ell, p, \mathsf{elim}_{\mathsf{SortedList}}(\dots, \ell), \mathsf{elim}_{\leq_{\mathrm{L}}}(\dots, n, \ell, p))
\end{aligned}
$$

for $\mathsf{elim}_{\mathsf{SortedList}}$, and

$$
\mathsf{elim}_{\leq_{\mathrm{L}}}(P, Q, \mathsf{step}_{\mathsf{nil}}, \mathsf{step}_{\mathsf{cons}}, \mathsf{step}_{\mathsf{triv}}, \mathsf{step}_{\ll\cdot\gg}, m, \mathsf{nil}, \mathsf{triv}_m) = \mathsf{step}_{\mathsf{triv}}(m)
$$

and

$$
\begin{aligned}
& \mathsf{elim}_{\leq_{\mathrm{L}}}(P, Q, \mathsf{step}_{\mathsf{nil}}, \mathsf{step}_{\mathsf{cons}}, \mathsf{step}_{\mathsf{triv}}, \mathsf{step}_{\ll\cdot\gg}, m, \mathsf{cons}(n, \ell, p), \ll q, p' \gg) \\
& \qquad = \mathsf{step}_{\ll\cdot\gg}(m, n, \ell, p, q, p', \mathsf{elim}_{\mathsf{SortedList}}(\dots, \ell), \\
& \qquad\qquad \mathsf{elim}_{\leq_{\mathrm{L}}}(\dots, n, \ell, p), \mathsf{elim}_{\leq_{\mathrm{L}}}(\dots, m, \ell, p'))
\end{aligned}
$$

for $\mathsf{elim}_{\leq_{\mathrm{L}}}$. Notice how the computation rules for $\mathsf{elim}_{\leq_{\mathrm{L}}}$ are well-typed because of the computation rules for $\mathsf{elim}_{\mathsf{SortedList}}$.

Now, suppose that we want to define a function $\mathsf{insert} : \mathsf{SortedList} \to \mathbb{N} \to \mathsf{SortedList}$ which inserts a number m into its appropriate place in a sorted list ℓ to create a new sorted list. From a high-level perspective, this is easy: the elimination rules allows us to make case distinctions between empty and non-empty lists, so it suffices to handle these two cases separately. The empty list is easy to handle, and for non-empty lists, we compare m with the first element n of the list $\ell = [n, \dots]$, which is possible since $\leq$ on natural numbers is decidable. If $m \leq n$, the result should be $[m, n, \dots]$, otherwise we recursively insert m into the tail of the list.

In detail, we choose $P(\ell) = \mathbb{N} \to \mathsf{SortedList}$ and, in our first attempt, $Q(n, \ell, p, \widetilde{\ell}) = \mathbf{1}$, since we are only interested in getting a function $\mathsf{elim}_{\mathsf{SortedList}}(\dots) : \mathsf{SortedList} \to \mathbb{N} \to \mathsf{SortedList}$. We need to give functions $\mathsf{step}_{\mathsf{nil}} : (m : \mathbb{N}) \to \mathsf{SortedList}$ and $\mathsf{step}_{\mathsf{cons}}(n, \ell, p) : (\widetilde{\ell} : \mathbb{N} \to \mathsf{SortedList}) \to Q(n, \ell, p, \widetilde{\ell}) \to (m : \mathbb{N}) \to \mathsf{SortedList}$ to use when inserting into the empty list or the list $\mathsf{cons}(n, \ell, p)$ respectively. The argument $\widetilde{\ell} : \mathbb{N} \to \mathsf{SortedList}$ gives the result of a recursive call on ℓ.

The function $\mathsf{step}_{\mathsf{nil}}$ is easy to define: it should be

$$\mathsf{step}_{\mathsf{nil}}(m) = \mathsf{cons}(m, \mathsf{nil}, \mathsf{triv}_m)$$

For $\mathsf{step}_{\mathsf{cons}}$, the decidability of $\leq$ (combined with the fact that $\leq$ is total) allows us to distinguish between the cases when $m \leq n$ and $n \leq m$. We try:

$$\mathsf{step}_{\mathsf{cons}}(n, \ell, p, \widetilde{\ell}, \star, m) = \begin{cases} \mathsf{cons}(m, \mathsf{cons}(n, \ell, p), \ll q, \mathsf{tra}_{\leq_{\mathrm{L}}}(q, p) \gg) & \text{where } q : m \leq n \\ \mathsf{cons}(n, \widetilde{\ell}(m), \{?\}) & \text{where } q : n \leq m \end{cases}$$

Here, $\mathsf{tra}_{\leq_{\mathrm{L}}} : m \leq n \to n \leq_{\mathrm{L}} \ell \to m \leq_{\mathrm{L}} \ell$ witnesses a kind of transitivity of $\leq$ and $\leq_{\mathrm{L}}$. It can be straightforwardly defined with the elimination rules. The question is what we should fill the hole $\{?\}$ with. We need to provide a proof that $n \leq_{\mathrm{L}} \widetilde{l}(m)$, i.e. that $n \leq_{\mathrm{L}} \mathsf{insert}(l, m)$ if we remember that $\widetilde{l}$ is the result of the recursive call on ℓ. We need to prove this simultaneously as we define insert! Fortunately, this is exactly what the elimination rules allow us to do if we choose a more meaningful Q.

Thus, we try again, but this time with

$$Q(n, \ell, p, \widetilde{\ell}) = (m : \mathbb{N}) \to n \leq m \to n \leq_{\mathrm{L}} \widetilde{l}(m) \ .$$

The argument $\star : \mathbf{1}$ to $\mathsf{step}_{\mathsf{cons}}$ in our first attempt has now been replaced with the argument $\widetilde{p} : (m : \mathbb{N}) \to n \leq m \to n \leq_{\mathrm{L}} \widetilde{l}(m)$, and we can define

$$\mathsf{step}_{\mathsf{cons}}(n, \ell, p, \widetilde{\ell}, \widetilde{p}, m) = \begin{cases} \mathsf{cons}(m, \mathsf{cons}(n, \ell, p), \ll q, \mathsf{tra}_{\leq_{\mathrm{L}}}(q, p) \gg) & \text{where } q : m \leq n \\ \mathsf{cons}(n, \widetilde{\ell}(m), \widetilde{p}(m, q)) & \text{where } q : n \leq m \end{cases}$$

Now we must also define $\mathsf{step}_{\mathsf{triv}} : (n : \mathbb{N}) \to Q(n, \mathsf{nil}, \mathsf{triv}_n, \mathsf{step}_{\mathsf{nil}})$ and $\mathsf{step}_{\ll\cdot\gg}$ with type as above for our choice of P and Q. This presents us with no further difficulties. For $\mathsf{step}_{\mathsf{triv}}$, expanding $Q(n, \mathsf{nil}, \mathsf{triv}_n, \mathsf{step}_{\mathsf{nil}})$ and replacing $\mathsf{step}_{\mathsf{triv}}$ with its definition, we see that we should give a function of type

$$\mathsf{step}_{\mathsf{triv}} : (n : \mathbb{N}) \to (m : \mathbb{N}) \to n \leq m \to n \leq_{\mathrm{L}} \mathsf{cons}(m, \mathsf{nil}, \mathsf{triv}_m) \ ,$$

so we can define $\mathsf{step}_{\mathsf{triv}}(n, m, p) = \ll p, \mathsf{triv}_n \gg$. The definition of $\mathsf{step}_{\ll\cdot\gg}$ follows the pattern of $\mathsf{step}_{\mathsf{cons}}$ above. Rather than trying to explain it, we just give the definition:

$$\mathsf{step}_{\ll\cdot\gg}(m, n, \ell, p, q, p', \widetilde{\ell}, \widetilde{p}, \widetilde{p'}, x, r) = \begin{cases} \ll r, \ll q, p' \gg\gg & \text{where } s : m \leq n \\ \ll q, \widetilde{p'}(x, r) \gg & \text{where } s : n \leq m \end{cases}$$

With all pieces in place, we can now define $\mathsf{insert} : \mathsf{SortedList} \to \mathbb{N} \to \mathsf{SortedList}$ as $\mathsf{insert} = \mathsf{elim}_{\mathsf{SortedList}}(P, Q, \mathsf{step}_{\mathsf{nil}}, \mathsf{step}_{\mathsf{cons}}, \mathsf{step}_{\mathsf{triv}}, \mathsf{step}_{\ll\cdot\gg})$. ∎

3.3 The examples revisited

We show how to find γ_{A}, γ_{B} for some well-known sets, including the examples in Section 1.1.

3.3.1 Encoding multiple constructors into one

The theory we have presented assumes that both A and B have exactly one constructor each. This is no limitation, as multiple constructors can always be encoded into one by using non-inductive arguments. Suppose that $\mathsf{intro}_0 : F_0(A,B) \to A$ and $\mathsf{intro}_1 : F_1(A,B) \to A$ are two constructors for A. Then we can combine them into one constructor

$$\mathsf{intro}_{0+1} : \big((i : \mathbf{2}) \times F_i(A,B)\big) \to A$$

by defining $\mathsf{intro}_{0+1}(i,x) = \mathsf{intro}_i(x)$.

If intro_0 is described by the code γ_0 and intro_1 by γ_1, then intro_{0+1} is described by the code

$$\gamma_0 +_{\mathsf{SP}} \gamma_1 \coloneqq \mathsf{non\text{-}ind}(\mathbf{2}, \lambda x.\mathsf{if}\ x\ \mathsf{then}\ \gamma_0\ \mathsf{else}\ \gamma_1) \ .$$

3.3.2 Examples of codes for inductive-inductive definitions

Well-orderings Ordinary inductive definitions can be interpreted as inductive-inductive definitions where we only care about the index set A and not about the family $B : A \to \mathsf{Set}$. A canonical choice is to let B have constructor $\mathsf{intro}_{\mathrm{B}} : (x : A) \to B(x)$, which is described by the code $\gamma_{\mathrm{dummy}} \coloneqq \mathsf{A\text{-}ind}(\mathbf{1}, \mathsf{nil}(\mathsf{a}_{\mathrm{ref}}(\mathsf{inr}(\star))))$[4].

For every $A : \mathsf{Set}$, $B : A \to \mathsf{Set}$, let

$$\gamma_{W(A,B)} \coloneqq \mathsf{non\text{-}ind}(A, \lambda x\,.\,\mathsf{A\text{-}ind}(B(x), \mathsf{nil}))$$

and define $W(A,B) \coloneqq A_{\gamma_{W(A,B)},\gamma_{\mathrm{dummy}}}$. Then $W(A,B)$ has constructor

$$\mathsf{intro}_{W(A,B)} : \big((x : A) \times (B(x) \to W(A,B)) \times \mathbf{1}\big) \to W(A,B) \ .$$

Finite sets Indexed inductive definitions can also be interpreted as inductive-inductive definitions, namely those where the index set just is an isomorphic copy of a previously constructed set (i.e. with constructor $\mathsf{intro}_{\mathrm{A}} : I \to A$ for some $I : \mathsf{Set}$).

For the family $\mathsf{Fin} : \mathbb{N} \to \mathsf{Set}$ of finite sets, the index set is $\mathbb{N}$, so we define

$$\gamma_A \coloneqq \mathsf{non\text{-}ind}(\mathbb{N}, \lambda n\,.\,\mathsf{nil}) : \mathsf{SP}^0_{\mathrm{A}}$$

and

$$\gamma_{\mathsf{Fin}} \coloneqq \gamma_{\mathsf{z}} +_{\mathsf{SP}} \gamma_{\mathsf{s}} : \mathsf{SP}^0_{\mathrm{B}}(\gamma_{\mathrm{A}})$$

where

$$\gamma_{\mathsf{z}} \coloneqq \mathsf{non\text{-}ind}(\mathbb{N}, \lambda n\,.\,\mathsf{nil}(\mathsf{arg}(\langle n+1, \star\rangle))) \ ,$$
$$\gamma_{\mathsf{s}} \coloneqq \mathsf{non\text{-}ind}(\mathbb{N}, \lambda n\,.\,\mathsf{B\text{-}ind}(\mathbf{1}, (\lambda\,\star\,.\mathsf{arg}(\langle n, \star\rangle)), \mathsf{nil}(\mathsf{arg}(\langle n+1, \star\rangle)))) \ .$$

[4] Another choice is $\gamma_{\mathrm{dummy}} = \mathsf{non\text{-}ind}(\mathbf{0}, !_{\mathsf{SP}^0_{\mathrm{B}}(\gamma_{\mathrm{A}})})$, which makes $B(x)$ an empty type.

Then $\mathsf{intro}_{A_{\gamma_A,\gamma_{\mathsf{Fin}}}} : \mathbb{N}\times\mathbf{1} \to A_{\gamma_A,\gamma_{\mathsf{Fin}}}$ is one part of an isomorphism $\mathbb{N} \cong \mathbb{N}\times\mathbf{1} \cong A_{\gamma_A,\gamma_{\mathsf{Fin}}}$, and if we define $\mathsf{Fin} : \mathbb{N} \to \mathsf{Set}$ by $\mathsf{Fin}(n) = B_{\gamma_A,\gamma_{\mathsf{Fin}}}(\mathsf{intro}_{A_{\gamma_A,\gamma_{\mathsf{Fin}}}}(\langle n, \star\rangle))$, then we can define constructors

$$\frac{n : \mathbb{N}}{\mathsf{z}_n : \mathsf{Fin}(n+1)} \qquad \frac{n : \mathbb{N} \qquad m : \mathsf{Fin}(n)}{\mathsf{s}_n(m) : \mathsf{Fin}(n+1)}$$

by $\mathsf{z}_n = \mathsf{intro}_{B_{\gamma_A,\gamma_{\mathsf{Fin}}}}(\langle \mathsf{tt}, \langle n, \star\rangle\rangle)$ and

$$\mathsf{s}_n(m) = \mathsf{intro}_{B_{\gamma_A,\gamma_{\mathsf{Fin}}}}(\langle \mathsf{ff}, \langle n, \langle (\lambda\star\,.\,m), \star\rangle\rangle\rangle \ .$$

Contexts and types The codes for the contexts and types from Example 1 are as follows:

$$\begin{array}{lcl}
\gamma_{\mathsf{Ctxt}} & = & \mathsf{nil} +_{\mathsf{SP}} \mathsf{A\text{-}ind}(\mathbf{1}, \mathsf{B\text{-}ind}(\mathbf{1}, (\lambda\star\,.\,\mathsf{inr}(\star)), \mathsf{nil})) : \mathsf{SP}^0_{\mathrm{A}} \\
\gamma_\iota & = & \mathsf{A\text{-}ind}(\mathbf{1}, \mathsf{nil}(\mathsf{a}_{\mathrm{ref}}(\mathsf{inr}(\star)))) \\
\gamma_\Pi & = & \mathsf{A\text{-}ind}(\mathbf{1}, \mathsf{B\text{-}ind}(\mathbf{1}, (\lambda\star\,.\,\mathsf{a}_{\mathrm{ref}}(\mathsf{inr}(\star))), \mathsf{B\text{-}ind}(\mathbf{1}, \\
 & & \qquad (\lambda\star\,.\mathsf{arg}(\langle \mathsf{ff}, \langle (\lambda\star\,.\mathsf{b}_{\mathrm{ref}}(\mathsf{inr}(\star))),\ \langle \lambda\star\,.\star, \star\rangle\rangle\rangle), \\
 & & \qquad\quad \mathsf{nil}(\mathsf{a}_{\mathrm{ref}}(\mathsf{inr}(\star)))))) \\
\gamma_{\mathsf{Ty}} & = & \gamma_\iota +_{\mathsf{SP}} \gamma_\Pi : \mathsf{SP}^0_{\mathrm{B}}(\gamma_{\mathsf{Ctxt}}) \ .
\end{array}$$

We have $\mathsf{Ctxt} = A_{\gamma_{\mathsf{Ctxt}},\gamma_{\mathsf{Ty}}}$ and $\mathsf{Ty} = B_{\gamma_{\mathsf{Ctxt}},\gamma_{\mathsf{Ty}}}$ and we can define the usual constructors by

$$\begin{array}{ll}
\varepsilon : \mathsf{Ctxt} & \iota : (\Gamma : \mathsf{Ctxt}) \to \mathsf{Ty}(\Gamma) \\
\varepsilon = \mathsf{intro}_{A_{\gamma_{\mathsf{Ctxt}},\gamma_{\mathsf{Ty}}}}(\langle \mathsf{tt}, \star\rangle) \ , & \iota_\Gamma = \mathsf{intro}_{B_{\gamma_{\mathsf{Ctxt}},\gamma_{\mathsf{Ty}}}}(\langle \mathsf{tt}, \langle (\lambda\star\,.\Gamma), \star\rangle\rangle) \ ,
\end{array}$$

$$\begin{array}{l}
\triangleright : (\Gamma : \mathsf{Ctxt}) \to \mathsf{Ty}(\Gamma) \to \mathsf{Ctxt} \\
\Gamma \triangleright \sigma = \mathsf{intro}_{A_{\gamma_{\mathsf{Ctxt}},\gamma_{\mathsf{Ty}}}}(\langle \mathsf{ff}, \langle (\lambda\star\,.\Gamma), \langle (\lambda\star\,.\sigma), \star\rangle\rangle\rangle) \quad ,
\end{array}$$

$$\begin{array}{l}
\Pi : (\Gamma : \mathsf{Ctxt}) \to (\sigma : \mathsf{Ty}(\Gamma)) \to \mathsf{Ty}(\Gamma \triangleright \sigma) \to \mathsf{Ty}(\Gamma) \\
\Pi(\Gamma, \sigma, \tau) = \mathsf{intro}_{B_{\gamma_{\mathsf{Ctxt}},\gamma_{\mathsf{Ty}}}}(\langle \mathsf{ff}, \langle (\lambda\star\,.\Gamma), \langle (\lambda\star\,.\sigma), \langle (\lambda\star\,.\tau), \star\rangle\rangle\rangle\rangle) \ .
\end{array}$$

4 A set-theoretic model

Even though SP_{A} and SP_{B} themselves are straightforward (large) inductive definitions, this axiomatisation does not reduce inductive-inductive definitions to indexed inductive definitions, since the formation and introduction rules are not instances of ordinary indexed inductive definitions. (However, we do believe that the theory of inductive-inductive definitions *can* be reduced to the theory of indexed inductive definitions with a bit of more work, and plan to do this in the future.) To make sure that our theory is consistent, it is thus necessary to construct a model.

We will develop a model in ZFC set theory, extended by two inaccessible cardinals in order to interpret Set and Type. Our model will be a simpler version of the models developed by Dybjer and Setzer [DS99, DS06] for induction-recursion. See Aczel [Acz99] for a more detailed treatment of interpreting type theory in set theory.

4.1 Preliminaries

We will be working informally in ZFC extended with the existence of two strongly inaccessible cardinals $\mathfrak{i}_0 < \mathfrak{i}_1$, and will be using standard set theoretic constructions, e.g.

$$\begin{aligned}
\langle a, b\rangle &\coloneqq \{\{a\}, \{a, b\}\} \ , \\
\lambda x \in a.b(x) &\coloneqq \{\langle x, b(x)\rangle \,|\, x \in a\} \ , \\
\Pi_{x\in a} b(x) &\coloneqq \{f : a \to \bigcup_{x\in a} b(x) \mid \forall x \in a.f(x) \in b(x)\} \ , \\
\Sigma_{x\in a} b(x) &\coloneqq \{\langle c, d\rangle \mid c \in a \wedge d \in b(c)\} \ , \\
0 \coloneqq \varnothing, 1 &\coloneqq \{0\}, 2 \coloneqq \{0, 1\} \ , \\
a_0 + \ldots + a_n &\coloneqq \Sigma_{i\in\{0,\ldots,n\}} a_i
\end{aligned}$$

and the cumulative hierarchy $V_\alpha \coloneqq \bigcup_{\beta<\alpha} \mathcal{P}(V_\beta)$. Whenever we introduce sets A^α indexed by ordinals α, let

$$A^{<\alpha} \coloneqq \bigcup_{\beta<\alpha} A^\beta.$$

For every expression A of our type theory, we will give an interpretation $[\![A]\!]_\rho$, regardless if $A : \mathsf{Type}$ or $A : B$ or not. Interpretations might however be undefined, written $[\![A]\!]_\rho \uparrow$. If $[\![A]\!]_\rho$ is defined, we write $[\![A]\!]_\rho \downarrow$. We write $A \simeq B$ for partial equality, i.e. $A \simeq B$ if and only if $A \downarrow \Leftrightarrow B \downarrow$ and if $A \downarrow$, then $A = B$. We write $A \coloneqq\!\simeq B$ if we define A such that $A \simeq B$.

Open terms will be interpreted relative to an environment ρ, i.e. a function mapping variables to terms. Write $\rho_{[x\mapsto a]}$ for the environment ρ extended with $x \mapsto a$, i.e. $\rho_{[x\mapsto a]}(y) = a$ if $y = x$ and $\rho(y)$ otherwise. The interpretation $[\![t]\!]_\rho$ of closed terms t will not depend on the environment, and we omit the subscript ρ.

4.2 Interpretation of Expressions

The interpretation of the logical framework is as in [DS99]:

$$[\![\mathsf{Set}]\!] \coloneqq\!\simeq V_{\mathfrak{i}_0} \qquad [\![\mathsf{Type}]\!] \coloneqq\!\simeq V_{\mathfrak{i}_1}$$

$$[\![(x : A) \to B]\!]_\rho \coloneqq\!\simeq \Pi_{y\in[\![A]\!]_\rho} [\![B]\!]_{\rho_{[y\mapsto x]}} \qquad [\![\lambda x : A.e]\!]_\rho \coloneqq\!\simeq \lambda y \in [\![A]\!]_\rho.[\![e]\!]_{\rho_{[y\mapsto x]}}$$

$$[\![(x : A) \times B]\!]_\rho \coloneqq\!\simeq \Sigma_{y\in[\![A]\!]_\rho} [\![B]\!]_{\rho_{[y\mapsto x]}} \qquad [\![\langle a, b\rangle]\!]_\rho \coloneqq\!\simeq \langle [\![a]\!]_\rho, [\![b]\!]_\rho\rangle$$

$$[\![\mathbf{0}]\!] \coloneqq\!\simeq 0 \qquad [\![\mathbf{1}]\!] \coloneqq\!\simeq 1 \qquad [\![\mathbf{2}]\!] \coloneqq\!\simeq 2 \qquad [\![\star]\!] \coloneqq\!\simeq 0 \qquad [\![\mathsf{tt}]\!] \coloneqq\!\simeq 0 \qquad [\![\mathsf{ff}]\!] \coloneqq\!\simeq 1$$

$$[\![\mathsf{if}\ x\ \mathsf{then}\ a\ \mathsf{else}\ b]\!]_\rho \coloneqq\!\simeq \begin{cases} [\![a]\!]_\rho & \text{if } [\![x]\!]_\rho = 0 \\ [\![b]\!]_\rho & \text{if } [\![x]\!]_\rho = 1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$[\![!_A]\!]_\rho \coloneqq\!\simeq \varnothing \text{ (the unique inclusion } \varnothing \to [\![A]\!]_\rho \text{)}$$

To interpret terms containing SP_A, SP_B, Arg_A, Arg_B, $\mathsf{Index}_\mathrm{B}$, nil, non-ind, A-ind, B-ind, we first define $[\![\mathsf{SP}_\mathrm{A}]\!]$, $[\![\mathsf{SP}_\mathrm{B}]\!]$, $[\![\mathsf{Arg}_\mathrm{A}]\!]$, $[\![\mathsf{nil}]\!]$, $[\![\mathsf{non\text{-}ind}]\!]$, ... and inter-

pret

$$\begin{aligned}
[\![\mathsf{SP}_\mathrm{A}(X_\mathrm{ref})]\!]_\rho &\coloneqq [\![\mathsf{SP}_\mathrm{A}]\!]([\![X_\mathrm{ref}]\!]_\rho) \\
&\vdots \\
[\![\mathsf{Arg}_\mathrm{A}(X_\mathrm{ref}, \gamma, X, Y, \mathsf{rep}_\mathrm{X})]\!]_\rho &\coloneqq [\![\mathsf{Arg}_\mathrm{A}]\!]([\![X_\mathrm{ref}]\!]_\rho, [\![\gamma]\!]_\rho, [\![X]\!]_\rho, [\![Y]\!]_\rho, [\![\mathsf{rep}_\mathrm{X}]\!]_\rho) \\
&\vdots \\
[\![\mathsf{non\text{-}ind}(K, \gamma)]\!]_\rho &\coloneqq [\![\mathsf{non\text{-}ind}]\!]([\![K]\!]_\rho, [\![\gamma]\!]_\rho) \\
&\vdots \quad \text{etc.}
\end{aligned}$$

In all future definitions, if we are currently defining $[\![F]\!]_\rho$ where $F : D \to E$, say, let $[\![F]\!]_\rho(d) \uparrow$ if $d \notin [\![D]\!]_\rho$.

$[\![\mathsf{SP}_\mathrm{A}]\!](X_\mathrm{ref})$ is defined as the least set such that

$$\begin{aligned}
[\![\mathsf{SP}_\mathrm{A}]\!](X_\mathrm{ref}) = 1 + \sum_{K \in [\![\mathsf{Set}]\!]} (K \to [\![\mathsf{SP}_\mathrm{A}]\!](X_\mathrm{ref})) + \sum_{K \in [\![\mathsf{Set}]\!]} [\![\mathsf{SP}_\mathrm{A}]\!](X_\mathrm{ref} + K) \\
+ \sum_{K \in [\![\mathsf{Set}]\!]} \sum_{h : K \to X_\mathrm{ref}} [\![\mathsf{SP}_\mathrm{A}]\!](X_\mathrm{ref}) \ .
\end{aligned}$$

The constructors are then interpreted as

$$\begin{aligned}
[\![\mathsf{nil}]\!] &\coloneqq \langle 0, 0 \rangle \quad [\![\mathsf{B\text{-}ind}]\!](K, h, \gamma) \coloneqq \langle 3, \langle K, \langle h, \gamma \rangle \rangle \rangle \\
[\![\mathsf{non\text{-}ind}]\!](K, \gamma) &\coloneqq \langle 1, \langle K, \gamma \rangle \rangle \quad [\![\mathsf{A\text{-}ind}]\!](K, \gamma) \coloneqq \langle 2, \langle K, \gamma \rangle \rangle
\end{aligned}$$

$[\![\mathsf{SP}_\mathrm{B}]\!]$ and its constructors are defined analogously. The functions $[\![\mathsf{Arg}_\mathrm{A}]\!]$, $[\![\mathsf{Arg}_\mathrm{B}]\!]$ and $[\![\mathsf{Index}_\mathrm{B}]\!]$ are defined according to their equations, e.g.

$$\begin{aligned}
[\![\mathsf{Arg}_\mathrm{A}]\!](X_\mathrm{ref}, [\![\mathsf{nil}]\!], X, Y, \mathsf{rep}_\mathrm{X}) &\coloneqq 1 \\
[\![\mathsf{Arg}_\mathrm{A}]\!](X_\mathrm{ref}, [\![\mathsf{non\text{-}ind}]\!](K, \gamma), X, Y, \mathsf{rep}_\mathrm{X}) &\coloneqq \sum_{k \in K} [\![\mathsf{Arg}_\mathrm{A}]\!](X_\mathrm{ref}, \gamma(k), X, Y, \mathsf{rep}_\mathrm{X}) \\
[\![\mathsf{Arg}_\mathrm{A}]\!](X_\mathrm{ref}, [\![\mathsf{A\text{-}ind}]\!](K, \gamma), X, Y, \mathsf{rep}_\mathrm{X}) &\coloneqq \sum_{j : K \to A} [\![\mathsf{Arg}_\mathrm{A}]\!](X_\mathrm{ref} + K, \gamma, X, Y, [\mathsf{rep}_\mathrm{X}, j]) \\
[\![\mathsf{Arg}_\mathrm{A}]\!](X_\mathrm{ref}, [\![\mathsf{B\text{-}ind}]\!](K, h, \gamma), X, Y, \mathsf{rep}_\mathrm{X}) &\coloneqq \sum_{j \in \Pi_{k \in K} B(\mathsf{rep}_\mathrm{X}(h(k)))} [\![\mathsf{Arg}_\mathrm{A}]\!](X_\mathrm{ref}, \gamma, X, Y, \mathsf{rep}_\mathrm{X}).
\end{aligned}$$

Finally, we have to interpret $A_{\gamma_\mathrm{A}, \gamma_\mathrm{B}}$, $B_{\gamma_\mathrm{A}, \gamma_\mathrm{B}}$, $\mathsf{intro}_{A_{\gamma_\mathrm{A}, \gamma_\mathrm{B}}}$ and $\mathsf{intro}_{B_{\gamma_\mathrm{A}, \gamma_\mathrm{B}}}$. The high-level idea is to iterate $\mathsf{Arg}^0_\mathrm{A}$ until a fixed point is reached, then apply $\mathsf{Arg}^0_\mathrm{B}$ once, and repeat. This is necessary since $\mathsf{Arg}^0_\mathrm{B}$ expects an argument $\mathsf{intro}_\mathrm{A} : \mathsf{Arg}^0_\mathrm{A}(\gamma_\mathrm{A}, A, B) \to A$, which can be chosen to be the identity if A is a fixed point of $\mathsf{Arg}^0_\mathrm{A}(\gamma_\mathrm{A}, A, B)$ (with B fixed). In more detail, let

$$\begin{aligned}
[\![A_{\gamma_\mathrm{A}, \gamma_\mathrm{B}}]\!] &\coloneqq A^{\mathsf{i}_0} \ , \quad [\![B_{\gamma_\mathrm{A}, \gamma_\mathrm{B}}]\!](a) \coloneqq B^{\mathsf{i}_0}(a) \ , \\
[\![\mathsf{intro}_{A_{\gamma_\mathrm{A}, \gamma_\mathrm{B}}}]\!](a) &\coloneqq a \ , \quad [\![\mathsf{intro}_{B_{\gamma_\mathrm{A}, \gamma_\mathrm{B}}}]\!](b) \coloneqq b \ ,
\end{aligned}$$

where A^α and B^α are simultaneously defined by recursion on α as

$$\begin{aligned}
A^\alpha &\coloneqq \text{least fixed point containing } A^{<\alpha} \text{ of } \lambda X \,.\, [\![\mathsf{Arg}^0_\mathrm{A}]\!](\gamma_A, X, B^{<\alpha}) \ , \\
B^\alpha(a) &\coloneqq \{b \mid b \in [\![\mathsf{Arg}^0_\mathrm{B}]\!](\gamma_A, A^\alpha, B^{<\alpha}, \mathsf{id}, \gamma_B) \\
&\qquad \wedge [\![\mathsf{Index}^0_\mathrm{B}]\!](\gamma_A, A^\alpha, B^{<\alpha}, \mathsf{id}, \gamma_B, b) = a\} \ .
\end{aligned}$$

The (graph of the) eliminators can then be built up in the same stages.

Having interpreted all terms, we finally interpret contexts as sets of environments:

$$[\![\varnothing]\!] \coloneqq \varnothing \qquad\qquad [\![\Gamma, x : A]\!] \coloneqq \{\rho_{[x \mapsto a]} \mid \rho \in [\![\Gamma]\!] \wedge a \in [\![A]\!]_\rho\}.$$

4.3 Soundness of the Rules

A detailed verification of the soundness of all the rules falls outside the scope of this paper. The main difficulty lies in proving that $[\![\mathsf{SP}_\mathrm{A}]\!]$ and $[\![\mathsf{SP}_\mathrm{B}]\!]$ are well-defined, and that $[\![A_{\gamma_A,\gamma_B}]\!] \in [\![\mathsf{Set}]\!]$ and $[\![B_{\gamma_A,\gamma_B}]\!] : [\![A_{\gamma_A,\gamma_B}]\!] \to [\![\mathsf{Set}]\!]$. Full details of the proof will be provided in a future publication (in preparation).

$[\![\mathsf{SP}_\mathrm{A}]\!]$ is obtained by iterating the appropriate operator $\Gamma : ([\![\mathsf{Set}]\!] \to [\![\mathsf{Set}]\!]) \to ([\![\mathsf{Set}]\!] \to [\![\mathsf{Set}]\!])$ up to $\mathfrak{i}_0$ times. Since $X_\mathrm{ref} \in [\![\mathsf{Set}]\!]$, we have $(X_\mathrm{ref} + K)$, $(K \to X_\mathrm{ref}) \in [\![\mathsf{Set}]\!]$ for all $K \in [\![\mathsf{Set}]\!] = V_{\mathfrak{i}_0}$ by the inaccessibility of $\mathfrak{i}_0$. Hence all "premisses" have cardinality at most $\mathfrak{i}_0$, which is regular, so that the operator has a fixed point after $\mathfrak{i}_0$ iterations, which must be an element of $[\![\mathsf{Type}]\!] = V_{\mathfrak{i}_1}$ by the inaccessibility of $\mathfrak{i}_1$.

To see that $[\![A_{\gamma_A,\gamma_B}]\!] \in [\![\mathsf{Set}]\!]$ and $[\![B_{\gamma_A,\gamma_B}]\!] : [\![A_{\gamma_A,\gamma_B}]\!] \to [\![\mathsf{Set}]\!]$, one first verifies that $[\![\mathsf{Arg}^0_\mathrm{A}]\!]$, $[\![\mathsf{Arg}^0_\mathrm{B}]\!]$, $[\![\mathsf{Index}^0_\mathrm{B}]\!]$ are monotone in the following sense:

Lemma 12. *For all $\gamma_\mathrm{A} \in [\![\mathsf{SP}^0_\mathrm{A}]\!]$ and $\gamma_\mathrm{B} \in [\![\mathsf{SP}^0_\mathrm{B}]\!](\gamma_\mathrm{A})$:*

(i) If $A \subseteq A'$ and $B(x) \subseteq B'(x)$ then $[\![\mathsf{Arg}^0_\mathrm{A}]\!](\gamma_\mathrm{A}, A, B) \subseteq [\![\mathsf{Arg}^0_\mathrm{A}]\!](\gamma_\mathrm{A}, A', B')$.

(ii) If in addition $\mathsf{intro}_\mathrm{A}(x) = \mathsf{intro}'_\mathrm{A}(x)$ for all $x \in \mathsf{Arg}^0_\mathrm{A}(\gamma_\mathrm{A}, A, B)$, then

$$[\![\mathsf{Arg}^0_\mathrm{B}]\!](\gamma_\mathrm{A}, A, B, \mathsf{intro}_\mathrm{A}, \gamma_\mathrm{B}) \subseteq [\![\mathsf{Arg}^0_\mathrm{B}]\!](\gamma_\mathrm{A}, A', B', \mathsf{intro}'_\mathrm{A}, \gamma_\mathrm{B})$$

and

$$[\![\mathsf{Index}^0_\mathrm{B}]\!](\gamma_\mathrm{A}, A, B, \mathsf{intro}_\mathrm{A}, \gamma_\mathrm{B}, x) = [\![\mathsf{Index}^0_\mathrm{B}]\!](\gamma_\mathrm{A}, A', B', \mathsf{intro}'_\mathrm{A}, \gamma_\mathrm{B}, x)$$

for all $x \in [\![\mathsf{Arg}^0_\mathrm{B}]\!](\gamma_\mathrm{A}, A, B, \mathsf{intro}_\mathrm{A}, \gamma_\mathrm{B})$. □

We can then adapt the standard results [Acz77] about monotone operators. First, we note that one application of $[\![\mathsf{Arg}^0_\mathrm{A}]\!]$ and $[\![\mathsf{Arg}^0_\mathrm{B}]\!]$ is not enough to take us outside of $[\![\mathsf{Set}]\!]$:

Lemma 13. *For all $\gamma_\mathrm{A} \in [\![\mathsf{SP}^0_\mathrm{A}]\!]$ and $\gamma_\mathrm{B} \in [\![\mathsf{SP}^0_\mathrm{B}]\!](\gamma_\mathrm{A})$:*

(i) If $X \in [\![\mathsf{Set}]\!]$ and $Y(x) \in [\![\mathsf{Set}]\!]$ for each $x \in X$, then $[\![\mathsf{Arg}^0_\mathrm{A}]\!](\gamma_\mathrm{A}, X, Y) \in [\![\mathsf{Set}]\!]$.

(ii) If $X \in [\![\mathsf{Set}]\!]$ and $Y(x) \in [\![\mathsf{Set}]\!]$ for each $x \in X$, then $[\![\mathsf{Arg}^0_\mathrm{B}]\!](\gamma_\mathrm{A}, X, Y, \mathsf{intro}_X, \gamma_\mathrm{B}) \in [\![\mathsf{Set}]\!]$.

We then iterate, using A^α and B^α, in order to reach a fixed point. This uses that fact that both $[\![\mathsf{Arg}^0_\mathrm{A}]\!]$ and $[\![\mathsf{Arg}^0_\mathrm{B}]\!]$ are κ-continuous for large enough κ:

Lemma 14.

(i) *For* $\alpha < \mathfrak{i}_0$, $A^\alpha \in [\![\mathsf{Set}]\!]$ *and* $B^\alpha : A^\alpha \to [\![\mathsf{Set}]\!]$.

(ii) *For* $\alpha < \beta$, $A^\alpha \subseteq A^\beta$ *and* $B^\alpha(a) \subseteq B^\beta(a)$ *for all* $a \in A^\alpha$.

(iii) *There is* $\kappa < \mathfrak{i}_0$ *such that for all* $\alpha \geq \kappa$, $A^\alpha = A^\kappa$ *and* $B^\alpha(a) = B^\kappa(a)$ *for all* $a \in A^\alpha$. □

Now we are done, since $[\![A_{\gamma_A,\gamma_B}]\!] = A^{\mathfrak{i}_0} = A^\kappa \in [\![\mathsf{Set}]\!]$, and similarly for $[\![B_{\gamma_A,\gamma_B}]\!]$.

References

[Acz77] Peter Aczel. An introduction to inductive definitions. In *Handbook of Mathematical Logic*, pages 739–782. Elsevier, 1977.

[Acz99] Peter Aczel. On relating type theories and set theories. *Lecture Notes In Computer Science*, 1657:1–18, 1999.

[AMNFS11] Thorsten Altenkirch, Peter Morris, Fredrik Nordvall Forsberg, and Anton Setzer. A categorical semantics for inductive-inductive definitions. In Andrea Corradini, Bartek Klin, and Corina Cirstea, editors, *Conference on Algebra and Coalgebra in Computer Science*, volume 6859 of *Lecture Notes in Computer Science*, pages 70 – 84. Springer, Heidelberg, 2011.

[Cha09] James Chapman. Type theory should eat itself. *Electronic Notes in Theoretical Computer Science*, 228:21–36, 2009.

[Con01] John Conway. *On numbers and games*. AK Peters, 2001.

[Dan07] Nils Anders Danielsson. A formalisation of a dependently typed language as an inductive-recursive family. *Lecture Notes in Computer Science*, 4502:93–109, 2007.

[DS99] Peter Dybjer and Anton Setzer. A finite axiomatization of inductive-recursive definitions. In *Typed lambda calculi and applications: 4th international conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999: proceedings*, pages 129–146. Springer Verlag, 1999.

[DS06] Peter Dybjer and Anton Setzer. Indexed induction–recursion. *Journal of logic and algebraic programming*, 66(1):1–49, 2006.

[Dyb94] Peter Dybjer. Inductive families. *Formal aspects of computing*, 6(4):440–465, 1994.

[Dyb00] Peter Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic*, 65(2):525–549, 2000.

[Mam06] Lionel Mamane. Surreal numbers in Coq. In Jean-Christophe Fillitre, Christine Paulin-Mohring, and Benjamin Werner, editors, *Types for Proofs and Programs: International Workshop TYPES 2004*, volume 3839 of *Lecture Notes in Computer Science*, pages 170 – 185. Springer, 2006.

[ML84] Per Martin-Löf. *Intuitionistic type theory*. Bibliopolis Naples, 1984.

[NFS10] Fredrik Nordvall Forsberg and Anton Setzer. Inductive-inductive definitions. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic*, volume 6247 of *Lecture Notes in Computer Science*, pages 454–468. Springer Berlin / Heidelberg, 2010.