Inductive-Inductive Definitions

Fredrik Nordvall Forsberg* Anton Setzer*
Department of Computer Science
Swansea University
Singleton Park
Swansea SA2 8PP, UK
{csfnf, a.g.setzer}@swansea.ac.uk

Abstract

We present a new principle for introducing new types in type theory which generalises strictly positive indexed inductive data types. In this new principle a set A is defined inductively simultaneously with an A-indexed set B, which is also defined inductively. Compared to indexed inductive definitions, the novelty is that the index set A is generated inductively simultaneously with B, or from another point of view: we mutually define two inductive sets, of which one depends on the other.

Instances of this principle have previously been used in order to formalise type theory inside type theory. However the consistency of framework used (the theorem prover Agda) is not so clear, as it allows the definition of a universe containing a code for itself. In this article we give an axiomatisation of the new principle in such a way that the resulting type theory is consistent, which we prove by constructing a set-theoretic model.

1. Introduction

Martin-Löf Type Theory is a foundational framework for constructive mathematics, where induction plays a major part in the construction of sets. Martin-Löf's formulation [17] includes inductive definitions of e.g. Cartesian products, disjoint unions, the identity set, finite sets, the natural numbers, wellorderings and lists. Backhouse [2, 3] and Dybjer [7] gave external schemas for general inductive sets and inductive families respectively.

Another induction principle is *induction-recursion*, where a set U is constructed inductively simultaneously with a recursively defined function $T : U \to D$ for some possibly large type D. The constructor for U may depend negatively on T applied to elements of U. The main example is Martin-Löf's universe à la Tarski [19]. Dybjer [9] gave a schema for such inductive-recursive definitions, which Dybjer and Setzer [10, 11, 12] internalised. Inductive-recursive definitions have also been used for generic programming in dependent type theory [4].

In this article, we present yet another induction principle, which we in reference to induction-recursion call *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. So we have formation rules $A : \text{Set}$, $B : A \to \text{Set}$ and typical introduction rules might look like

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

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

Inductive-inductive definitions have been used by Dybjer [8], Danielsson [6] and Chapman [5] to internalise the syntax and semantics of type theory. Slightly simplified, they define a set Ctxt of contexts, a family $\text{Ty} : \text{Ctxt} \to \text{Set}$ of types in a given context, and a family $\text{Term} : (\Gamma : \text{Ctxt}) \to \text{Ty}(\Gamma) \to \text{Set}$ of terms of a given type. Let us for simplicity only consider contexts and types. The set Ctxt of contexts has two constructors

$$\varepsilon : \text{Ctxt}$$
$$\text{cons} : (\Gamma : \text{Ctxt}) \times \text{Ty}(\Gamma) \to \text{Ctxt},$$

corresponding to the empty context and extending a context Γ with a new type. In our simplified setting, $\text{Ty} : \text{Ctxt} \to \text{Set}$ has the following constructors

$$\text{'set'} : (\Gamma : \text{Ctxt}) \to \text{Ty}(\Gamma)$$
$$\Pi : (\Gamma : \text{Ctxt}) \times (A : \text{Ty}(\Gamma)) \times \text{Ty}(\text{cons}(\Gamma, A)) \to \text{Ty}(\Gamma).$$

*Supported by EPSRC grant EP/G033374/1, Theory and applications of induction-recursion.

The first constructor states that Set is a type in any context. Π is the constructor for the Π-type: If we have a type A in a context Γ, and another type B in Γ extended by A (corresponding to abstracting a variable of type A), then $\Pi(A, B)$ is also a type in Γ.

Note how the constructor cons for Ctxt has an argument of type $\mathrm{Ty}(\Gamma)$, even though Ty is indexed by Ctxt. It is also worth noting that Π has an argument of type $\mathrm{Ty}(\mathrm{cons}(\Gamma, A))$, i.e. we are using the constructor for Ctxt in the index of Ty. In general, we could of course imagine an argument of type $\mathrm{Ty}(\mathrm{cons}(\mathrm{cons}(\Gamma, A), A'))$ etc.

Both Danielsson and Chapman used the proof assistant Agda [22] as a framework for their formalisation. Agda supports inductive-inductive (and inductive-recursive) definitions by the `mutual` keyword. However, the implementation in Agda also allows the definition of a universe U à la Tarski, with a code $u : U$ for itself, i.e. $T(u) = U$. If we also demand closure under Π or Σ, the positivity checker rejects the code, so this does not necessarily mean that Agda is inconsistent by Girard's paradox [13].

A reasonable implementation of induction-induction should not allow a definition of a universe containing a code for itself, i.e. if the principle is consistent. It is thus important to investigate induction-induction, especially as the work of Danielsson and Chapman relies on the principle.

Plan of the paper In Section 2, we present the type theoretical preliminaries. In Section 3, we discuss the new induction principle. We then proceed with an axiomatisation in Section 4, which we prove to be consistent in Section 5 by constructing a set-theoretic model.

2. Type theoretic preliminaries

We work in a type theory with at least two universes Set and Type, with Set : Type and if A : Set then A : Type. Both Set and Type are closed under dependent function types, written $(x : A) \to B$. Abstraction is written as $\lambda x : A.e$ and application as $f(x)$, with repeated abstraction and application written as $\lambda x_1 : A_1, \ldots, x_k : A_k.e$ and $f(x_1, \ldots, x_k)$. 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 $(x : A) \times B$ with pairs $\langle a, b\rangle$. We also have β- and η-rules for both dependent function types and products.

We need an empty type $\mathbf{0}$: Set with elimination $!_A : \mathbf{0} \to A$ for every A : Set, and a unit type $\mathbf{1}$: Set with unique element $\star : \mathbf{1}$. We include an η-rule stating that if $x : \mathbf{1}$, then $x = \star : \mathbf{1}$. Moreover, we include a two element set $\mathbf{2}$: Set with elements $\mathrm{tt} : \mathbf{2}$ and $\mathrm{ff} : \mathbf{2}$ and elimination constant $\mathrm{case}_{\mathbf{2}} : (a : \mathbf{2}) \to A(\mathrm{tt}) \to A(\mathrm{ff}) \to A(a)$ where $i : \mathbf{2} \Rightarrow A(i) : \mathrm{Type}$.

With $\mathrm{case}_{\mathbf{2}}$ and dependent products, we can now as in [12, A.2] define the disjoint union of two sets $A + B \coloneqq (x : \mathbf{2}) \times \mathrm{case}_{\mathbf{2}}(x, A, B)$ with constructors $\mathrm{inl} = \lambda a : A.\langle \mathrm{tt}, a\rangle$ and $\mathrm{inr} = \lambda b : B.\langle \mathrm{ff}, b\rangle$, and prove the usual formation, introduction, elimination and equality rules. We write $A_0 + A_1 + \ldots + A_n$ for $A_0 + (A_1 + (\ldots + A_n)\cdots)$ and $\mathrm{in}_k(a)$ for the kth injection $\mathrm{inl}(\mathrm{inr}^k(a))$ (with special case $\mathrm{in}_n(a) = \mathrm{inr}^n(a)$).

Using (the derived) elimination rules for +, we can, for A, B : Set, $C : A + B \to \mathrm{Type}$ and $f : (a : A) \to C(\mathrm{inl}(a))$, $g : (b : B) \to C(\mathrm{inr}(b))$, define $f \sqcup g : (c : A + B) \to C(c)$ with equality rules

$$(f \sqcup g)(\mathrm{inl}(a)) = f(a)$$
$$(f \sqcup g)(\mathrm{inr}(b)) = g(b).$$

We will use the same notation even when C does not depend on $c : A + B$, and we will write $f \parallel g : A + B \to C + D$ for $(\mathrm{inl} \circ f) \sqcup (\mathrm{inr} \circ g)$.

Intensional type theory in Martin-Löf's logical framework extended with dependent products and $\mathbf{0}$, $\mathbf{1}$ and $\mathbf{2}$ have all the features we need. Our development could thus, if one so wishes, be seen as an extension of the logical framework.

3. Inductive-inductive definitions

Let us first informally consider how to formalise a simultaneous inductive definition of two sets A and B, given by constructors

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

where Φ_A and Φ_B are strictly positive in the following sense:

- The constant $\Phi(A, B) = \mathbf{1}$ is strictly positive. It corresponds to an introduction rule with no arguments (or more precisely, the trivial argument $x : \mathbf{1}$).
- If K is a set and Ψ_x is strictly positive, depending on $x : K$, then $\Phi(A, B) = (x : K) \times \Psi_x(A, B)$, corresponding to the addition of a non-inductive premise, is strictly positive. So intro_A has one non-inductive argument $x : K$, followed by the arguments given by $\Psi_x(A, B)$.
- If K is a set and Ψ is strictly positive, then $\Phi(A, B) = (K \to A) \times \Psi(A, B)$ is strictly positive. This corresponds to the addition of a premise inductive in A, where K corresponds to the hypothesis of this premise in a generalised inductive definition. So intro_A has one inductive argument $K \to A$, followed by the arguments given by $\Psi(A, B)$.
- Likewise, if K is a set and Ψ is strictly positive, then $\Phi(A, B) = (K \to B) \times \Psi(A, B)$ is strictly positive. This is similar to the previous case.

In an inductive-inductive definition, B is indexed by A, so the constructor for B is replaced by

$$\text{intro}_\text{B} : (a : \Phi_\text{B}(A, B)) \to B(i_{A,B}(a))$$

for some index $i_{A,B}(a) : A$ which might depend on $a : \Phi_\text{B}(A, B)$. Furthermore, we must modify the inductive case for B to supply an index as well. This index can (and usually does) depend on earlier inductive arguments, so that the new inductive cases become

- If K is a set, and Ψ_f is strictly positive, depending on $f : K \to A$ only in indices for B, then $\Phi(A, B) = (f : K \to A) \times \Psi_f(A, B)$ is strictly positive.
- If K is a set, $i_{A,B} : K \to A$ is a function and Ψ_f is strictly positive, depending on $f : (x : K) \to B(i_{A,B}(x))$ only in indices for B, then $\Phi(A, B) = (f : ((x : K) \to B(i_{A,B}(x)))) \times \Psi_f(A, B)$ is strictly positive.

The informal phrase "depending on f only in indices for B" will be given an exact meaning in the formalisation in the next section.

Let us consider the indices for B in inductive premises a little further. When we are writing down the constructor for A, we do not know anything about A as we are currently constructing it. Thus, there certainly cannot be any functions $f : C \to A$ yet, as we have not introduced A as a set. So we conclude that we can only use elements from A as indices as they are given to us.

When writing down the constructor for B, the situation is similar. At this point, however, we know one function into A, namely $\text{intro}_\text{A} : \Phi_\text{A}(A, B) \to A$. Furthermore, we can iterate intro_A any number of times to get new functions. This will all be taken into account in the formalisation.

4. An axiomatisation

We proceed as in [10] and introduce a datatype of codes for constructors. This time, however, we will have one datatype SP_A for constructors for A and another datatype SP_B for constructors for B. Both will come with "functors" Arg_A, Arg_B that map $A : \text{Set}$, $B : A \to \text{Set}$ to the domain of the constructor $\text{intro}_\text{A} : \text{Arg}_\text{A}(\gamma, A, B) \to A$ for a code $\gamma : \text{SP}_\text{A}$ and similarly for SP_B and Arg_B. For SP_B, we will also need a map Index_B which picks out the index in A of the constructed elements.

SP_A will have two parameters D, D' of elements we can refer to for indices for B in A and B respectively. More precisely, an element $x : D$ will represent an element $f(x) : A$, and an element $y : D'$ will represent an element $f'_A(y) : A$ together with an element $f'_B(y) : B(f'_A(y))$. For instance, the arguments of a constructor following an inductive argument $(j : K \to A)$ can refer to elements $j(k)$ for $k : K$. The code for those arguments would be an element of $\text{SP}_\text{A}(D+K, D')$, so K represents the new elements in A we can refer to. For finished codes, D and D' will always be the empty set $\mathbf{0}$. We have the following formation rule for SP_A

$$\frac{D : \text{Set} \quad D' : \text{Set}}{\text{SP}_\text{A}(D, D') : \text{Type}}$$

and we define $\text{SP}'_\text{A} \coloneqq \text{SP}_\text{A}(\mathbf{0}, \mathbf{0})$. The introduction rules for SP_A reflect the rules for strict positivity in Section 3, and are probably easiest understood together with the "decoding function" Arg_A defined below (we suppress the global premise $D, D' : \text{Set}$):

$$\frac{}{\text{nil}_\text{A} : \text{SP}_\text{A}(D, D')} \qquad \frac{K : \text{Set} \quad \gamma : K \to \text{SP}_\text{A}(D, D')}{\text{nonind}(K, \gamma) : \text{SP}_\text{A}(D, D')}$$

$$\frac{K : \text{Set} \quad \gamma : \text{SP}_\text{A}(D + K, D')}{\text{A-ind}(K, \gamma) : \text{SP}_\text{A}(D, D')}$$

$$\frac{K : \text{Set} \quad h : K \to D \quad \gamma : \text{SP}_\text{A}(D, D' + K)}{\text{B-ind}(K, h, \gamma) : \text{SP}_\text{A}(D, D')}$$

Note that the codomain of the function $h : K \to D$, which picks out the index in the rule B-ind, is D, the set of elements in A we can refer to.

We will now define Arg_A, corresponding to Φ_A above, which from a given code $\gamma : \text{SP}_\text{A}(D, D')$ constructs the domain of the constructor intro_A:

$$\frac{\begin{array}{ccc} & & f : D \to A \\ D, D' : \text{Set} & A : \text{Set} & f'_A : D' \to A \\ \gamma : \text{SP}_\text{A}(D, D') & B : A \to \text{Set} & f'_B : (x : D') \to B(f'_A(x)) \end{array}}{\text{Arg}_\text{A}(D, D', \gamma, A, B, f, f'_A, f'_B) : \text{Set}}$$

The function f translates elements in D to the real elements they denote in A. Similarly, f'_A gives the index and f'_B the real element in $B(f'_A(y))$ for an element $y : D'$. Again, for completed codes $\gamma : \text{SP}'_\text{A}$, we will define $\text{Arg}'_\text{A} : \text{SP}'_\text{A} \to (A : \text{Set}) \to (B : A \to \text{Set}) \to \text{Set}$ by $\text{Arg}'_\text{A}(\gamma, A, B) \coloneqq \text{Arg}_\text{A}(\mathbf{0}, \mathbf{0}, \gamma, A, B, !_A, !_A, !_B)$.

The definition of Arg_A also follows the rules for strict positivity in Section 3. We will suppress the arguments A, B, as they can be inferred from e.g. f'_A and f'_B.

The code nil_A represents the constructor with no argument (i.e. a trivial argument of type $\mathbf{1}$):

$$\text{Arg}_\text{A}(D, D', \text{nil}_\text{A}, f, f'_A, f'_B) = \mathbf{1}$$

The code $\text{nonind}(K, \gamma)$ represents one non-inductive argument $(k : K)$, with the rest of the arguments given by the code γ (depending on $k : K$):

$$\text{Arg}_\text{A}(D, D', \text{nonind}(K, \gamma), f, f'_A, f'_B) = \big(k : K\big) \times \text{Arg}_\text{A}(D, D', \gamma(k), f, f'_A, f'_B)$$

The code $\text{A-ind}(K, \gamma)$ represents one generalised inductive argument $(j : K \to A)$, with the rest of the arguments given

by the code γ. Notice that after this argument we can refer to more elements in (the suppressed argument) A, namely $j(k)$ for $k : K$:

$$\mathrm{Arg_A}(D, D', \text{A-ind}(K, \gamma), f, f'_A, f'_B) = \\ \big(j : K \to A\big) \times \mathrm{Arg_A}(D + K, D', \gamma, f \sqcup j, f'_A, f'_B)$$

Finally, B-ind(K, h, γ) represents one generalised inductive argument $(j : (x : K) \to B(f(h(x))))$, where $f \circ h$ picks out the index of $j(x)$. This time, we can refer to more elements in B afterwards:

$$\mathrm{Arg_A}(D, D', \text{B-ind}(K, h, \gamma), f, f'_A, f'_B) = \\ \big(j : (k : K) \to B(f(h(k)))\big) \times \\ \mathrm{Arg_A}(D, D' + K, \gamma, f, f'_A \sqcup (f \circ h), f'_B \sqcup j)$$

$\mathrm{Arg_A}$ is now functorial in the following sense: if we have maps

$$g : A \to A^*, \quad g' : (x : A) \to B(x) \to B^*(g(x))$$

that respect the translations f and f^*, i.e. $g(f(x)) = f^*(x)$ for all $x : D$, then we can lift g to a map

$$\mathrm{lift}(\dots, g, \dots) : \mathrm{Arg_A}(D, D', \gamma, A, B, f, f'_A, f'_B) \to \\ \mathrm{Arg_A}(D, D', \gamma, A^*, B^*, f^*, f_A^{*'}, f_B^{*'}).$$

This will be needed for the soon to be introduced $\mathrm{Arg_B}$, as we will need to translate from "$\mathrm{Arg_A}(D, D')$" to $\mathrm{Arg_A}(A, B)$ in order to use the constructor for A as an index for B in an inductive argument.

To avoid using the identity type, we will use a specialised version of Leibniz equality $p : (x : D) \to B^*(g(f(x))) \to B^*(f^*(x))$ for the proof that $g(f(x)) = f^*(x)$. Under the assumption that $\gamma : \mathrm{SP_A}(D, D')$, and with A, B, f, f'_A, f'_B as above, and A^*, B^*, f^*, $f_A^{*'}$, $f_B^{*'}$ analogously, we now have formation rule

$$\mathrm{lift}(D, D', \gamma, A, B, f, f'_A, f'_B, A^*, B^*, f^*, f_A^{*'}, f_B^{*'}) : \\ (g : A \to A^*) \to (g' : (x : A) \to B(x) \to B^*(g(x))) \\ \to (p : (x : D) \to B^*(g(f(x))) \to B^*(f^*(x))) \\ \to \mathrm{Arg_A}(D, D', \gamma, A, B, f, f'_A, f'_B) \\ \to \mathrm{Arg_A}(D, D', \gamma, A^*, B^*, f^*, f_A^{*'}, f_B^{*'})$$

with defining equations (A, B, A^*, B^* suppressed)

$$\mathrm{lift}(D, D', \mathrm{nil_A}, f, f'_A, f'_B, f^*, f_A^{*'}, f_B^{*'}, g, g', p, \star) = \star$$
$$\mathrm{lift}(\dots, \text{nonind}(K, \gamma), \dots, g, g', p, \langle k, y\rangle) = \\ \langle k, \mathrm{lift}(D, D', \gamma(k), f, f'_A, f'_B, f^*, f_A^{*'}, f_B^{*'}, g, g', p, y)\rangle$$
$$\mathrm{lift}(\dots, \text{A-ind}(K, \gamma), \dots, g, g', p, \langle j, y\rangle) = \\ \langle g \circ j, \mathrm{lift}(D + K, D', \gamma, f \sqcup j, \dots, f^* \sqcup (g \circ j), \dots, \\ p \sqcup (\lambda k, b.b), y)\rangle$$
$$\mathrm{lift}(\dots, \text{B-ind}(K, h, \gamma), \dots, g, g', p, \langle j, y\rangle) = \\ \langle \lambda k.p(h(k), g'(f(h(k)), j(k))), \\ \mathrm{lift}(D, D' + K, \gamma, f, f'_A \sqcup (f \circ h), f'_B \sqcup j, f^*, \\ f_A^{*'} \sqcup (f^* \circ h), f_B^{*'} \sqcup (\lambda k.p(h(k), g'(f(h(k)), \\ j(k)))), g, g', p, y)\rangle.$$

Note how the proof $p : (x : D) \to B^*(g(f(x))) \to B^*(f^*(x))$ gets updated to $p \sqcup (\lambda k, b.b)$ when D is extended to $D + K$. For $\gamma : \mathrm{SP'_A}$, $g : A \to A^*$, $g' : (x : A) \to B(x) \to B^*(g(x))$, we define

$$\mathrm{lift'}(\gamma, A, B, A^*, B^*, g, g') : \\ \mathrm{Arg'_A}(\gamma, A, B) \to \mathrm{Arg'_A}(\gamma, A^*, B^*)$$

by

$$\mathrm{lift'}(\gamma, A, B, A^*, B^*, g, g', a) \coloneqq \\ \mathrm{lift}(\mathbf{0}, \mathbf{0}, \gamma, A, B, !_A, !_A, !_B, \\ A^*, B^*, !_{A^*}, !_{A^*}, !_{B^*}, g, g', !_p, a).$$

Note that also the proof $!_p : (x : \mathbf{0}) \to B^*(g(f(x)) \to B^*(f^*(x))$ is supplied by ex falso quodlibet.

Armed with the lift function, we can now introduce the datatype $\mathrm{SP_B}$ of codes for constructors for B. All constructions from now on will be parameterised on the maximum number k of nested constructors for A that we are using, so we are really introducing $\mathrm{SP_{B,}}_k$, $\mathrm{Arg_{B,}}_k$ etc. However, we will work with an arbitrary k but suppress it as a premise.

$\mathrm{SP_B}$ have parameters D, D'_0 : Set just as $\mathrm{SP_A}$, but in addition also parameters $\gamma_A : \mathrm{SP'_A}$ and D'_i : Set for $0 < i \le k$. We need the code γ_A to know the constructors for A, and D'_i will represent elements in B indexed by i nested constructors for A. We have the following formation rule for $\mathrm{SP_B}$:

$$\frac{\gamma_A : \mathrm{SP'_A} \quad D : \text{Set} \quad D'_0, D'_1, \dots, D'_k : \text{Set}}{\mathrm{SP_B}(\gamma_A, D, D'_0, D'_1, \dots, D'_k) : \text{Type}.}$$

Let $\mathrm{SP'_B}(\gamma_A) \coloneqq \mathrm{SP_B}(\gamma_A, \mathbf{0}, \mathbf{0}, \dots, \mathbf{0})$ for $\gamma_A : \mathrm{SP'_A}$.

Let us now define some abbreviations which will be useful later. $\mathrm{Arg'^i_A}$ is $\mathrm{Arg'_A}$ iterated i times, i.e. $\mathrm{Arg'^i_A}$ is the set of arguments used by i nested constructors.

$$\mathrm{Arg'^0_A}(\gamma, A, \vec{B}) = A$$
$$\mathrm{Arg'^{n+1}_A}(\gamma, A, \vec{B}) = \mathrm{Arg'_A}(\gamma, \coprod_{i=0}^{n} \mathrm{Arg'^i_A}(\gamma, A, \vec{B}_{(i-1)}), \bigsqcup_{i=0}^{n} B_i)$$

where $\vec{B}_{(n)}$ is the sequence $B_0, B_1, \dots, B_n$ of families $B_i : \mathrm{Arg'^i_A}(\gamma_A, A, \vec{B}_{(i-1)}) \to \text{Set}$.

We think of D and D' as "syntactical" counterparts to A and B respectively. However, D' is flat, in the sense that $B : A \to \text{Set}$ but $D' : \text{Set}$. But every element $x : D'$ corresponds to exactly one pair $f'_A(x) : A$, $f'_B(x) : B(f'_A(x))$, so that $(\lambda d' : D'.\mathbf{1}) : D' \to \text{Set}$ is a good non-flat representation of D'; we can map $x : D'$ to $f'_A(x) : A$, and $\star : (\lambda d'.\mathbf{1})(x)$ to $f'_B(x) : B(f'_A(x))$. These are the two translation functions $g : A \to A^*$, $g' : (x : A) \to B(x) \to B^*(g(x))$ needed in lift$'$. We also have to take elements $d : D$ into account, though, representing elements $f(d) : A$ where we cannot refer to any elements in $B(f(d))$ yet. So our syntactical non-flat version of A and B is going to be $D + D'$ together with $(\lambda d.\mathbf{0}) \sqcup (\lambda d'.\mathbf{1})$:

For a code $\gamma : \mathrm{SP'_A}$ and D, D' : Set, define

$$\mathrm{arg_A}(\gamma, D, D') \coloneqq \mathrm{Arg'_A}(\gamma, D + D', (\lambda d.\mathbf{0}) \sqcup (\lambda d'.\mathbf{1})).$$

Also arg_A can be iterated in a similar way as Arg'_A:

$$\mathrm{arg}^0_\mathrm{A}(\gamma, D, \vec{D}') = D$$
$$\mathrm{arg}^{n+1}_\mathrm{A}(\gamma, D, \vec{D}') = \mathrm{arg}_\mathrm{A}(\gamma, \dot{+}_{i=0}^{n} \mathrm{arg}^i_\mathrm{A}(\gamma, D, \vec{D}'_{(i-1)}), D'_n)$$

Given functions $f : D \to A$, $f'_{A,i} : D'_i \to \mathrm{Arg}'^i_\mathrm{A}(\gamma, A, \vec{B})$, and $f'_{B,i} : (x : D'_i) \to B_i(f'_{A,i}(x))$, we can now construct functions $f_n : \mathrm{arg}^n_\mathrm{A}(\gamma, D, \vec{D}') \to \mathrm{Arg}'^n_\mathrm{A}(\gamma, A, \vec{B})$ with the help of lift' by defining

$$\begin{aligned} f_0 &= f \\ f_{n+1} &= \mathrm{lift}'(\gamma, (\dot{+}_{i=0}^{n} \mathrm{arg}^i_\mathrm{A}(\gamma, D, \vec{D}')) + D'_n, \\ &\quad (\lambda d.\mathbf{0}) \sqcup (\lambda d'.\mathbf{1}), \dot{+}_{i=0}^{n} \mathrm{Arg}'^i_\mathrm{A}(\gamma, A, \vec{B}_{(i-1)}), \\ &\quad \bigsqcup_{i=0}^{n} B_i, (\|_{i=0}^{n} f_i) \sqcup (\mathrm{in}_n \circ f'_{A,n}), \\ &\quad !_{\sqcup B} \sqcup (\lambda d' \star .f'_{B,n}(d'))). \end{aligned}$$

We can now finally introduce the introduction rules for SP_B. The rules are very similar to the rules for SP_A, but now we specify an index in nil_B, and we have $k+1$ rules B_0-ind, …, B_k-ind corresponding to how many nested constructors for A we want to use:

$$\frac{d : D + \mathrm{arg}^1_\mathrm{A}(\gamma_A, D, \vec{D}') + \ldots + \mathrm{arg}^k_\mathrm{A}(\gamma_A, D, \vec{D}')}{\mathrm{nil}_\mathrm{B}(d) : \mathrm{SP}_\mathrm{B}(\gamma_A, D, D'_0, \ldots, D'_k)}$$

$$\frac{K : \mathrm{Set} \quad \gamma : K \to \mathrm{SP}_\mathrm{B}(\gamma_A, D, D'_0, \ldots, D'_k)}{\mathrm{nonind}(K, \gamma) : \mathrm{SP}_\mathrm{B}(\gamma_A, D, D'_0, \ldots, D'_k)}$$

$$\frac{K : \mathrm{Set} \quad \gamma : \mathrm{SP}_\mathrm{B}(\gamma_A, D + K, D'_0, \ldots, D'_k)}{\text{A-ind}(K, \gamma) : \mathrm{SP}_\mathrm{B}(\gamma_A, D, D'_0, \ldots, D'_k)}$$

$$\frac{\begin{array}{c} h : K \to \mathrm{arg}^n_\mathrm{A}(\gamma_A, D, \vec{D}') \\ K : \mathrm{Set} \quad \gamma : \mathrm{SP}_\mathrm{B}(\gamma_A, D, D'_0, \ldots, D'_n + K, \ldots, D'_k) \end{array}}{\mathrm{B}_n\text{-ind}(K, h, \gamma) : \mathrm{SP}_\mathrm{B}(\gamma_A, D, D'_0, \ldots, D'_k)}$$

We will now, analogously with Arg_A, define Arg_B, which takes a code $\gamma : \mathrm{SP}_\mathrm{B}(\gamma_A, D, D'_0, \ldots, D'_k)$, a set A and $k+1$ families $B_i : \mathrm{Arg}'^i_\mathrm{A}(\gamma_A, A, \vec{B}_{(i-1)}) \to \mathrm{Set}$, together with functions $f : D \to A$, $f'_{A,i} : D'_i \to \mathrm{Arg}'^i_\mathrm{A}(\gamma, A, \vec{B}_{(i-1)})$ and $f'_{B,i} : (x : D'_i) \to B_i(f'_{A,i}(x))$, and returns the domain of the constructor for B. Recall that from f, $f_{A,i}$ and $f_{B,i}$, we can construct functions $f_n : \mathrm{arg}^n_\mathrm{A}(\gamma, D, \vec{D}') \to \mathrm{Arg}'^n_\mathrm{A}(\gamma, A, \vec{B})$.

The definition follows the same pattern as Arg_A, except that B and f are replaced by B_n and f_n respectively in B_n-ind, and D', f'_A, f'_B are split up into $k+1$ different D'_n, $f'_{A,n}$, $f'_{B,n}$. For $\gamma_B : \mathrm{SP}'_\mathrm{B}(\gamma_A)$, let $\mathrm{Arg}'_\mathrm{B}(\gamma_A, \gamma_B A, \vec{B}) := \mathrm{Arg}_\mathrm{B}(\gamma_A, \mathbf{0}, \vec{\mathbf{0}}, \gamma_B, A, \vec{B}, !_A, !_{\vec{\mathrm{Arg}_\mathrm{A}}}, !_{\vec{B}})$. (We have suppressed the arguments A, $\vec{B}$, as they can be inferred.)

$$\begin{aligned} &\mathrm{Arg}_\mathrm{B}(\gamma_A, D, \vec{D}', \mathrm{nil}_\mathrm{B}(d), f, \vec{f'_A}, \vec{f'_B}) &&= \mathbf{1} \\ &\mathrm{Arg}_\mathrm{B}(\gamma_A, D, \vec{D}', \mathrm{nonind}(K, \gamma), f, \vec{f'_A}, \vec{f'_B}) &&= \\ &\qquad (k : K) \times \mathrm{Arg}_\mathrm{B}(\gamma_A, D, \vec{D}', \gamma(k), f, \vec{f'_A}, \vec{f'_B}) \\ &\mathrm{Arg}_\mathrm{B}(\gamma_A, D, \vec{D}', \text{A-ind}(K, \gamma), f, \vec{f'_A}, \vec{f'_B}) &&= \\ &(j : K \to A) \times \mathrm{Arg}_\mathrm{B}(\gamma_A, D + K, \vec{D}', \gamma, f \sqcup j, \vec{f'_A}, \vec{f'_B}) \\ &\mathrm{Arg}_\mathrm{B}(\gamma_A, D, \vec{D}', \mathrm{B}_n\text{-ind}(K, h, \gamma), f, \vec{f'_A}, \vec{f'_B}) = \\ &\quad (j : (k : K) \to B_n(f_n(h(k)))) \times \\ &\qquad \mathrm{Arg}_\mathrm{B}(\gamma_A, D, \ldots, D'_n + K, \ldots, \gamma, f, \ldots, \\ &\qquad\quad f'_{A,n} \sqcup (f_n \circ h), \ldots, f'_{B,n} \sqcup j, \ldots) \end{aligned}$$

The last missing piece is now $\mathrm{Index}_\mathrm{B}$, which assigns to each $b : \mathrm{Arg}_\mathrm{B}(\gamma_A, \gamma_B, A, \vec{B})$ the index of $\mathrm{intro}_\mathrm{B}(b)$. This index is in $\mathrm{Arg}'^i_\mathrm{A}(\gamma_A, A, \vec{B})$ for some $i \le k$ and is stored in the nil_B constructor. Under the assumption that γ_A, D, D'_i, γ_B, A, B_i, f, $f'_{A,i}$ and $f'_{B,i}$ are as above, $\mathrm{Index}_\mathrm{B}$ has formation rule

$$\begin{aligned} &\mathrm{Index}_\mathrm{B}(\gamma_A, D, \vec{D}', \gamma_B, A, \vec{B}, f, \vec{f'_A}, \vec{f'_B}) : \\ &\qquad \mathrm{Arg}_\mathrm{B}(\gamma_A, D, \vec{D}', \gamma_B, A, \vec{B}, f, \vec{f'_A}, \vec{f'_B}) \to \\ &\qquad\quad A + \mathrm{Arg}'^1_\mathrm{A}(\gamma_A, A, \vec{B}) + \ldots + \mathrm{Arg}'^k_\mathrm{A}(\gamma_A, A, \vec{B}) \end{aligned}$$

and defining equations (with A, $\vec{B}$ suppressed)

$$\begin{aligned} &\mathrm{Index}_\mathrm{B}(\gamma_A, D, \vec{D}', \mathrm{nil}_\mathrm{B}(d), f, \vec{f'_A}, \vec{f'_B}, \star) = (\|_{i=0}^{k} f_i)(d) \\ &\mathrm{Index}_\mathrm{B}(\gamma_A, D, \vec{D}', \mathrm{nonind}(K, \gamma), f, \vec{f'_A}, \vec{f'_B}, \langle k, y\rangle) = \\ &\qquad \mathrm{Index}_\mathrm{B}(\gamma_A, D, \vec{D}', \gamma(k), f, \vec{f'_A}, \vec{f'_B}, y) \\ &\mathrm{Index}_\mathrm{B}(\gamma_A, D, \vec{D}', \text{A-ind}(K, \gamma), f, \vec{f'_A}, \vec{f'_B}, \langle j, y\rangle) = \\ &\qquad \mathrm{Index}_\mathrm{B}(\gamma_A, D + K, \vec{D}', \gamma, f \sqcup j, \vec{f'_A}, \vec{f'_B}, y) \\ &\mathrm{Index}_\mathrm{B}(\gamma_A, D, \vec{D}', \mathrm{B}_n\text{-ind}(K, h, \gamma), f, \vec{f'_A}, \vec{f'_B}, \langle j, y\rangle) = \\ &\quad \mathrm{Index}_\mathrm{B}(\gamma_A, D, \ldots, D'_n + K, \ldots, \gamma, f, \ldots, \\ &\qquad f'_{A,n} \sqcup (f_n \circ h), \ldots, f'_{B,n} \sqcup j, \ldots, y). \end{aligned}$$

Let $\mathrm{Index}'_\mathrm{B}(\gamma_A, \gamma_B, A, \vec{B}, b) := \mathrm{Index}_\mathrm{B}(\gamma_A, \mathbf{0}, \vec{\mathbf{0}}, \gamma_B, A, \vec{B}, !_A, !_{\vec{\mathrm{Arg}_\mathrm{A}}}, !_B, b)$.

4.1. Formation and introduction rules

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

Formation rules:

$$A_{\gamma_A,\gamma_B} : \mathrm{Set} \qquad B_{\gamma_A,\gamma_B} : A_{\gamma_A,\gamma_B} \to \mathrm{Set}$$

Introduction rule for A_{γ_A,γ_B}:

$$\frac{a : \mathrm{Arg}'_\mathrm{A}(\gamma_A, A_{\gamma_A,\gamma_B}, B_{\gamma_A,\gamma_B})}{\mathrm{intro}_\mathrm{A}(a) : A_{\gamma_A,\gamma_B}}$$

For the introduction rule for B_{γ_A,γ_B}, we need some preliminary definitions. Define

$$\mathrm{intro}_0 : \mathrm{Arg}'^0_\mathrm{A}(\gamma_A, A_{\gamma_A,\gamma_B}, B_{\gamma_A,\gamma_B}) \to A_{\gamma_A,\gamma_B}$$
$$\mathrm{intro}_1 : \mathrm{Arg}'^1_\mathrm{A}(\gamma_A, A_{\gamma_A,\gamma_B}, B_{\gamma_A,\gamma_B}) \to A_{\gamma_A,\gamma_B}$$

by $\text{intro}_0 = \text{id}$, $\text{intro}_1 = \text{intro}_\text{A}$. We can now define

$$B_0 : \text{Arg}'^0_\text{A}(\gamma_A, A_{\gamma_A,\gamma_B}, B_{\gamma_A,\gamma_B}) \to \text{Set}$$
$$B_1 : \text{Arg}'^1_\text{A}(\gamma_A, A_{\gamma_A,\gamma_B}, B_{\gamma_A,\gamma_B}) \to \text{Set}$$

by letting $B_i(x) = B_{\gamma_A,\gamma_B}(\text{intro}_i(x))$. Then, we can define $\text{intro}_2 : \text{Arg}'^2_\text{A}(\gamma_A, A_{\gamma_A,\gamma_B}, B_0, B_1) \to A_{\gamma_A,\gamma_B}$ by (we have omitted the parameters γ_A, γ_B in A_{γ_A,γ_B}, B_{γ_A,γ_B} for readability)

$$\begin{aligned}\text{intro}_2 = \text{intro}_\text{A} \circ \text{lift}'(\gamma_A, A + \text{Arg}'_\text{A}(\gamma_A, A, B),\\ B_0 \sqcup B_1, A, B, \text{intro}_0 \sqcup \text{intro}_1,\\ (\lambda a.\text{id}) \sqcup (\lambda a.\text{id}))\end{aligned}$$

(i.e. intro_2 applies intro_1 and intro_0 in the right places to reduce everything in $\text{Arg}^2_\text{A}(\gamma_A, A, \vec{B})$ to A) and can thus next define $B_2 : \text{Arg}'^2_\text{A}(\gamma_A, A_{\gamma_A,\gamma_B}, B_0, B_1) \to \text{Set}$ by $B_2(x) = B_{\gamma_A,\gamma_B}(\text{intro}_2(x))$. In general for $n \le k$, we get $\text{intro}_n : \text{Arg}'^n_\text{A}(\gamma_A, A_{\gamma_A,\gamma_B}, B_0, \ldots, B_{n-1}) \to A_{\gamma_A,\gamma_B}$ by

$$\begin{aligned}\text{intro}_n = \text{intro}_\text{A} \circ \text{lift}'(\gamma_A, \dot{+}_{i=0}^{n-1} \text{Arg}'^i_\text{A}(\gamma_A, A, \vec{B}_{(i-1)}),\\ \bigsqcup_{i=0}^{n-1} B_i, A, B, \bigsqcup_{i=0}^{n-1} \text{intro}_i,\\ \bigsqcup_{i=0}^{n-1} (\lambda a.\text{id})),\end{aligned}$$

and we define $B_n : \text{Arg}'^n_\text{A}(\gamma_A, A_{\gamma_A,\gamma_B}, \vec{B}_{(n-1)}) \to \text{Set}$ by $B_n(x) = B_{\gamma_A,\gamma_B}(\text{intro}_n(x))$.

The introduction rule for B_{γ_A,γ_B}, finally, is

$$\frac{b : \text{Arg}'_\text{B}(\gamma_A, A_{\gamma_A,\gamma_B}, B_{\gamma_A,\gamma_B}, B_1, \ldots, B_k)}{\text{intro}_\text{B}(b) : B_{\gamma_A,\gamma_B}(\text{index}(b))}$$

where

$$\begin{aligned}&\text{index} : \text{Arg}'_\text{B}(\gamma_A, A_{\gamma_A,\gamma_B}, B_0, \ldots, B_k) \to A_{\gamma_A,\gamma_B}\\ &\text{index} = (\bigsqcup_{i=0}^{k} \text{intro}_i) \circ\\ &\qquad \text{Index}'_\text{B}(\gamma_A, \gamma_B, A_{\gamma_A,\gamma_B}, B_0, \ldots, B_k).\end{aligned}$$

4.2. Elimination rules

The intuitive idea behind the elimination rules is the following: we have $E : A \to \text{Set}$ and $E' : (a : A) \to B(a) \to \text{Set}$, and would like to construct functions $f : (a : A) \to E(a)$ and $g : (a : A) \to (b : B(a)) \to E'(a, b)$. f and g might very well be mutually recursive, since A and B are mutually defined. The elimination rules now state that if we can define functions $f' : (a : \text{Arg}'_\text{A}(\gamma_A, A, B)) \to E(\text{intro}_\text{A}(a))$ and $g' : (b : \text{Arg}'_\text{B}(\gamma_A, \gamma_B, A, B)) \to E'(\text{index}(b), \text{intro}_\text{B}(b))$, given that we already know the value of (mutual) recursive calls, we have functions $f : (a : A) \to E(a)$ and $g : (a : A) \to (b : B(a)) \to E'(a, b)$.

To this end, let us define sets IH_A and IH_B of inductive hypotheses, together with maps $\text{mapIH}_\text{A} : \text{Arg}'_\text{A}(\gamma_A, A, B) \to \text{IH}_\text{A}$, $\text{mapIH}_\text{B} : \text{Arg}'_\text{B}(\gamma_A, \gamma_B, A, \vec{B}) \to \text{IH}_\text{B}$ which take care of the recursive call.

More specifically, if D, D', γ_A, A, B, f, f'_A, f'_B are as in the premises for Arg_A, and $E : A \to \text{Set}$, $E' : (a : A) \to B(a) \to \text{Set}$, then

$$\begin{aligned}&\text{IH}_\text{A}(D, D', \gamma_A, A, B, f, f'_A, f'_B, E, E') :\\ &\qquad \text{Arg}_\text{A}(D, D', \gamma_A, A, B, f, f'_A, f'_B) \to \text{Set}\end{aligned}$$

and

$$\begin{aligned}&\text{mapIH}_\text{A}(D, D', \gamma_A, A, B, f, f'_A, f'_B, E, E') :\\ &\quad (R : (a : A) \to E(a)) \to\\ &\qquad (R' : (a : A) \to (b : B(a)) \to E'(a, b)) \to\\ &\qquad (a : \text{Arg}_\text{A}(D, D', \gamma_A, A, B, f, f'_A, f'_B)) \to\\ &\qquad\quad \text{IH}_\text{A}(D, D', \gamma_A, A, B, f, f'_A, f'_B, E, E', a).\end{aligned}$$

IH_A is defined as follows (with A, B, E, E' suppressed):

$$\begin{aligned}&\text{IH}_\text{A}(D, D', \text{nil}_\text{A}, f, f'_A, f'_B, \star) &&= \mathbf{1}\\ &\text{IH}_\text{A}(D, D', \text{nonind}(K, \gamma), f, f'_A, f'_B, \langle k, y\rangle) &&=\\ &\qquad \text{IH}_\text{A}(D, D', \gamma(k), f, f'_A, f'_B, y)\\ &\text{IH}_\text{A}(D, D', \text{A-ind}(K, \gamma), f, f'_A, f'_B, \langle j, y\rangle) &&=\\ &\quad \big((k : K) \to E(j(k))\big) \times\\ &\qquad \text{IH}_\text{A}(D + K, D', \gamma, f \sqcup j, f'_A, f'_B, y)\\ &\text{IH}_\text{A}(D, D', \text{B-ind}(K, h, \gamma), f, f'_A, f'_B, \langle j, y\rangle) =\\ &\quad \big((k : K) \to E'(f(h(k)), j(k))\big) \times\\ &\qquad \text{IH}_\text{A}(D, D' + K, \gamma, f, f'_A \sqcup (f \circ h), f'_B \sqcup j, y).\end{aligned}$$

mapIH_A maps $a : \text{Arg}_\text{A}(\gamma_A, A, B)$ to $\text{IH}_\text{A}(\gamma_A, A, B, a)$, given that we have functions R, R' for the recursive calls. It has the following defining equations (once again with A, B, E, E' suppressed):

$$\begin{aligned}&\text{mapIH}_\text{A}(D, D', \text{nil}_\text{A}, f, f'_A, f'_B, R, R', \star) &&= \star\\ &\text{mapIH}_\text{A}(\ldots, \text{nonind}(K, \gamma), \ldots, R, R', \langle k, y\rangle) &&=\\ &\qquad \text{mapIH}_\text{A}(D, D', \gamma(k), f, f'_A, f'_B, R, R', y)\\ &\text{mapIH}_\text{A}(\ldots, \text{A-ind}(K, \gamma), \ldots, R, R', \langle j, y\rangle) &&=\\ &\langle R \circ j, \text{mapIH}_\text{A}(D + K, D', \gamma, f \sqcup j, f'_A, f'_B, R, R', y)\rangle\\ &\text{mapIH}_\text{A}(\ldots, \text{B-ind}(K, h, \gamma), \ldots, R, R', \langle j, y\rangle) =\\ &\quad \langle \lambda k.R'(f(h(k)), j(k)),\\ &\qquad \text{mapIH}_\text{A}(D, D' + K, \gamma, f,\\ &\qquad\qquad f'_A \sqcup (f \circ h), f'_B \sqcup j, R, R', y)\rangle.\end{aligned}$$

We now repeat the process for IH_B and mapIH_B. This time, however, we have not one $B : A \to \text{Set}$ but $k + 1$ families $B_i : \text{Arg}'^i_\text{A}(\gamma_A, A, \vec{B}_{(i-1)}) \to \text{Set}$, so we need $k + 1$ families $E'_i : (a : \text{Arg}'^i_\text{A}(\gamma_A, A, \vec{B})) \to B_i(a) \to \text{Set}$ and $k + 1$ functions $R'_i : (a : \text{Arg}'^i_\text{A}(\gamma_A, A, \vec{B})) \to (b : B_i(a)) \to E'_i(a, b)$. Otherwise, the pattern is the same.

The formation rules are as follows: if γ_A, D, D'_i, γ_B, A, B_i, f, $f'_{A,i}$,$f'_{B,i}$ are as before, and $E : A \to \text{Set}$, $E'_i : (a : \text{Arg}'^i_\text{A}(\gamma_A, A, \vec{B}_{(i-1)})) \to B_i(a) \to \text{Set}$, then

$$\begin{aligned}&\text{IH}_\text{B}(\gamma_A, D, \vec{D}', \gamma_A, A, \vec{B}, f, \vec{f'_A}, \vec{f'_B}, E, \vec{E'}) :\\ &\qquad \text{Arg}_\text{B}(\gamma_A, D, \vec{D}', \gamma_B, A, \vec{B}, f, \vec{f'_A}, \vec{f'_B}) \to \text{Set}\end{aligned}$$

and

For $E : A_{\gamma_A,\gamma_B} \to \text{Set}$ and $E' : (a : A_{\gamma_A,\gamma_B}) \to B_{\gamma_A,\gamma_B}(a) \to \text{Set}$, we have elimination rules (we have abbreviated $A \coloneqq A_{\gamma_A,\gamma_B}$, $B \coloneqq B_{\gamma_A,\gamma_B}$)

$$\frac{\begin{array}{c} g : (a : \text{Arg}'_{\text{A}}(\gamma_A, A, B)) \to \text{IH'}_{\text{A}}(\gamma_A, A, B, E, E', a) \to E(\text{intro}_{\text{A}}(a)) \\ h : (b : \text{Arg}'_{\text{B}}(\gamma_A, \gamma_B, A, B_0, \ldots, B_k)) \to \text{IH'}_{\text{B}}(\gamma_A, \gamma_B, A, \vec{B}, E, E'_0, \ldots, E'_k, b) \to E'(\text{index}(b), \text{intro}_{\text{B}}(b)) \end{array}}{\text{R}_{\text{A}}(g,h) : (a : A_{\gamma_A,\gamma_B}) \to E(a)}$$

$$\frac{\begin{array}{c} g : (a : \text{Arg}'_{\text{A}}(\gamma_A, A, B)) \to \text{IH'}_{\text{A}}(\gamma_A, A, B, E, E', a) \to E(\text{intro}_{\text{A}}(a)) \\ h : (b : \text{Arg}'_{\text{B}}(\gamma_A, \gamma_B, A, B_0, \ldots, B_k)) \to \text{IH'}_{\text{B}}(\gamma_A, \gamma_B, A, \vec{B}, E, E'_0, \ldots, E'_k, b) \to E'(\text{index}(b), \text{intro}_{\text{B}}(b)) \end{array}}{\text{R}_{\text{B}}(g,h) : (a : A_{\gamma_A,\gamma_B}) \to (b : B_{\gamma_A,\gamma_B}(a)) \to E'(a,b)}$$

with equality rules (premises omitted)

$$\text{R}_{\text{A}}(g,h,\text{intro}_{\text{A}}(a)) = g(a, \text{mapIH}'_{\text{A}}(\gamma_A, A, B, E, E', \text{R}_{\text{A}}(g,h), \text{R}_{\text{B}}(g,h), a))$$

$$\text{R}_{\text{B}}(g,h,\text{index}(b),\text{intro}_{\text{B}}(b)) = h(b, \text{mapIH}'_{\text{B}}(\gamma_A, \gamma_B, A, B_0, \ldots, B_k, E, E'_0, \ldots, E'_k, \text{R}_{\text{A}}(g,h), \text{R}_{\text{B}}(g,h), b))$$

Figure 1. Elimination and equality rules for A_{γ_A,γ_B}, B_{γ_A,γ_B}

$$\begin{array}{l} \text{mapIH}_{\text{B}}(\gamma_A, D, \vec{D'}, \gamma_B, A, \vec{B}, f, \vec{f'_A}, \vec{f'_B}, E, \vec{E'}) : \\ \quad (R : (a : A) \to E(a)) \to \\ \qquad (R'_0 : (a : A) \to (b : B_0(a)) \to E'_0(a,b)) \to \\ (R'_1 : (a : \text{Arg}'^1_{\text{A}}(\gamma_A, A, B_0)) \to (b : B_1(a)) \to E'_1(a,b)) \to \\ \qquad \vdots \\ (R'_k : (a : \text{Arg}'^k_{\text{A}}(\gamma_A, A, B_{k-1})) \to (b : B_k(a)) \to E'_k(a,b)) \to \\ \quad (b : \text{Arg}_{\text{B}}(\gamma_A, D, \vec{D'}, \gamma_B, A, \vec{B}, f, \vec{f'_A}, \vec{f'_B})) \to \\ \qquad \text{IH}_{\text{B}}(\gamma_A, D, \vec{D'}, \gamma_B, A, \vec{B}, f, \vec{f'_A}, \vec{f'_B}, E, \vec{E'}, b). \end{array}$$

We get the equations for IH_{B} and mapIH_{B} by changing IH_{A} and mapIH_{A} in more or less the same way we changed Arg_{A} to get Arg_{B}; in the B_n-ind case, we replace f with f_n, f'_A with $f'_{A,n}$, f'_B with $f'_{B,n}$, B with B_n and now also E' with E'_n and R' with R'_n. (Once again, we have suppressed A, $B_0, \ldots, B_k$, E and $E'_0, \ldots, E'_k$. In the B_n-ind cases, D'_i and $f'_{A,i}$, $f'_{B,i}$ with $i \neq n$ are passed along unmodified in the recursive call.)

$$\begin{array}{l} \text{IH}_{\text{B}}(\gamma_A, D, \vec{D'}, \text{nil}_{\text{B}}(d), f, \vec{f'_A}, \vec{f'_B}, \star) \quad = \mathbf{1} \\ \text{IH}_{\text{B}}(\gamma_A, D, \vec{D'}, \text{nonind}(K,\gamma), f, \vec{f'_A}, \vec{f'_B}, \langle k, y\rangle) \quad = \\ \qquad \text{IH}_{\text{B}}(\gamma_A, D, \vec{D'}, \gamma(k), f, \vec{f'_A}, \vec{f'_B}, y) \\ \text{IH}_{\text{B}}(\gamma_A, D, \vec{D'}, \text{A-ind}(K,\gamma), f, \vec{f'_A}, \vec{f'_B}, \langle j, y\rangle) \quad = \\ \quad \big((k : K) \to E(j(k))\big) \times \\ \qquad \text{IH}_{\text{B}}(\gamma_A, D + K, \vec{D'}, \gamma, f \sqcup j, \vec{f'_A}, \vec{f'_B}, y) \\ \text{IH}_{\text{B}}(\gamma_A, D, \vec{D'}, \text{B}_n\text{-ind}(K,h,\gamma), f, \vec{f'_A}, \vec{f'_B}, \langle j, y\rangle) = \\ \quad \big((k : K) \to E'_n(f_n(h(k)), j(k))\big) \times \\ \text{IH}_{\text{B}}(\gamma_A, D, D'_n + K, \gamma, f, f'_{A,n} \sqcup (f_n \circ h), f'_{B,n} \sqcup j, y). \end{array}$$

$$\text{mapIH}_{\text{B}}(\gamma_A, D, \vec{D'}, \text{nil}_{\text{B}}(d), f, \vec{f'_A}, \vec{f'_B}, R, \vec{R'}, \star) \quad = \star$$

$$\begin{array}{l} \text{mapIH}_{\text{B}}(\ldots, \text{nonind}(K,\gamma), \ldots, R, \vec{R'}, \langle k, y\rangle) \quad = \\ \qquad \text{mapIH}_{\text{B}}(\gamma_A, D, \vec{D'}, \gamma(k), f, \vec{f'_A}, \vec{f'_B}, R, \vec{R'}, y) \\ \text{mapIH}_{\text{B}}(\ldots, \text{A-ind}(K,\gamma), \ldots, R, \vec{R'}, \langle j, y\rangle) \quad = \\ \quad \langle R \circ j, \text{mapIH}_{\text{B}}(\gamma_A, D + K, \vec{D'}, \gamma, f \sqcup j, \\ \qquad \vec{f'_A}, \vec{f'_B}, R, \vec{R'}, y)\rangle \\ \text{mapIH}_{\text{B}}(\ldots, \text{B}_n\text{-ind}(K,h,\gamma), \ldots, R, \vec{R'}, \langle j, y\rangle) = \\ \quad \langle \lambda k. R'_n(f_n(h(k)), j(k)), \\ \qquad \text{mapIH}_{\text{A}}(\gamma_A, D, D'_n + K, \gamma, f, \\ \qquad\quad f'_{A,n} \sqcup (f_n \circ h), f'_{B,n} \sqcup j, R, \vec{R'}, y)\rangle. \end{array}$$

We make the usual abbreviations

$$\begin{array}{l} \text{IH'}_{\text{A}}(\gamma_A, A, B, E, E', a) \coloneqq \\ \qquad \text{IH}_{\text{A}}(\mathbf{0}, \mathbf{0}, \gamma_A, A, B, !_A, !_A, !_B, E, E', a) \\ \text{mapIH}'_{\text{A}}(\gamma_A, A, B, E, E', R, R', a) \coloneqq \\ \quad \text{mapIH}_{\text{A}}(\mathbf{0}, \mathbf{0}, \gamma_A, A, B, !_A, !_A, !_B, E, E', R, R', a) \\ \text{IH'}_{\text{B}}(\gamma_A, \gamma_B, A, \vec{B}, E, \vec{E'}, b) \coloneqq \\ \qquad \text{IH}_{\text{B}}(\gamma_A, \mathbf{0}, \vec{\mathbf{0}}, \gamma_B, A, \vec{B}, !_A, !_{\text{Arg}_{\text{A}}}, !\vec{}_B, E, \vec{E'}, b) \\ \text{mapIH}'_{\text{B}}(\gamma_A, \gamma_B, A, \vec{B}, E, \vec{E'}, R, \vec{R'}, b) \coloneqq \\ \qquad \text{mapIH}_{\text{B}}(\gamma_A, \mathbf{0}, \vec{\mathbf{0}}, \gamma_B, A, \vec{B}, !_A, !_{\text{Arg}_{\text{A}}}, \\ \qquad\qquad !\vec{}_B, E, \vec{E'}, R, \vec{R'}, b). \end{array}$$

It is now time to introduce the elimination and equality rules for A_{γ_A,γ_B} and B_{γ_A,γ_B}, which can be found in Figure 1. There, we have used the abbreviations B_n from Section 4.1, as well as a new abbreviation $E'_n \coloneqq \lambda a, b. E'(\text{intro}_n(a), b)$.

4.3. Contexts and types again

Let us first, as in [12], introduce the abbreviation

$$\gamma_0 +_{SP} \gamma_1 \coloneqq \text{nonind}(\mathbf{2}, \lambda x. \text{case}_{\mathbf{2}}(x, \gamma_0, \gamma_1))$$

for codes γ_i in $\mathrm{SP}'_{\mathrm{A}}$ or $\mathrm{SP}'_{\mathrm{B}}(\gamma_A)$. With the help of $+_{SP}$, we can encode several constructors into one.

We now give the code for the example of contexts and types from the introduction. Let first

$$\begin{aligned}\gamma_\varepsilon &= \mathrm{nil}_{\mathrm{A}}\\ \gamma_{\mathrm{cons}} &= \text{A-ind}(\mathbf{1}, \text{B-ind}(\mathbf{1}, \lambda\star.\mathrm{inr}(\star), \mathrm{nil}_{\mathrm{A}}))\\ \gamma_{\mathrm{Ctxt}} &= \gamma_\varepsilon +_{SP} \gamma_{\mathrm{cons}} : \mathrm{SP}'_{\mathrm{A}}\end{aligned}$$

and continue by defining

$$\begin{aligned}\gamma_{\text{'set'}} &= \text{A-ind}(\mathbf{1}, \mathrm{nil}_{\mathrm{B}}(\mathrm{inl}(\mathrm{inr}(\star))))\\ \gamma_\Pi &= \text{A-ind}(\mathbf{1}, \mathrm{B}_0\text{-ind}(\mathbf{1}, \lambda\star.\mathrm{inr}(\star),\\ &\qquad \mathrm{B}_1\text{-ind}(\mathbf{1}, \lambda\star.\langle \mathrm{ff}, \langle\lambda\star.\mathrm{inr}(\mathrm{inr}(\star)),\\ &\qquad\quad \langle\lambda\star.\star, \star\rangle\rangle\rangle, \mathrm{nil}_{\mathrm{B}}(\mathrm{inl}(\mathrm{inr}(\star))))))\\ \gamma_{\mathrm{Ty}} &= \gamma_{\text{'set'}} +_{SP} \gamma_\Pi : \mathrm{SP}'_{\mathrm{B}}(\gamma_{\mathrm{Ctxt}}).\end{aligned}$$

We now have $\mathrm{Ctxt} = A_{\gamma_{\mathrm{Ctxt}}, \gamma_{\mathrm{Ty}}}$ and $\mathrm{Ty} = B_{\gamma_{\mathrm{Ctxt}}, \gamma_{\mathrm{Ty}}}$. The domain of $\mathrm{intro}_{\mathrm{A}}$, $\mathrm{intro}_{\mathrm{B}}$ is respectively

$$(i : \mathbf{2}) \times \begin{cases}\mathbf{1} & i = \mathrm{tt}\\ (\Gamma : \mathbf{1} \to \mathrm{Ctxt}) \times ((\mathbf{1} \to \mathrm{Ty}(\Gamma(\star))) \times \mathbf{1}) & i = \mathrm{ff},\end{cases}$$

$$(i : \mathbf{2}) \times \begin{cases}(\mathbf{1} \to \mathrm{Ctxt}) \times \mathbf{1} & i = \mathrm{tt}\\ \quad (\Gamma : \mathbf{1} \to \mathrm{Ctxt}) \times ((b : \mathbf{1} \to \mathrm{Ty}(\Gamma(\star))) \times & \\ ((\mathbf{1} \to \mathrm{Ty}(\mathrm{intro}_{\mathrm{A}}(\langle \mathrm{ff}, \langle\Gamma, \langle b, \star\rangle\rangle\rangle))) \times \mathbf{1})) & i = \mathrm{ff},\end{cases}$$

and we can define the usual constructors by

$$\begin{aligned}&\varepsilon : \mathrm{Ctxt}\\ &\varepsilon = \mathrm{intro}_{\mathrm{A}}\langle \mathrm{tt}, \star\rangle\end{aligned}$$

$$\begin{aligned}&\mathrm{cons} : (\Gamma : \mathrm{Ctxt}) \to \mathrm{Ty}(\Gamma) \to \mathrm{Ctxt}\\ &\mathrm{cons}(\Gamma, b) = \mathrm{intro}_{\mathrm{A}}(\langle \mathrm{ff}, \langle(\lambda\star.\Gamma), \langle(\lambda\star.b), \star\rangle\rangle\rangle)\end{aligned}$$

$$\begin{aligned}&\text{'set'} : (\Gamma : \mathrm{Ctxt}) \to \mathrm{Ty}(\Gamma)\\ &\text{'set'}(\Gamma) = \mathrm{intro}_{\mathrm{B}}\langle \mathrm{tt}, \langle(\lambda\star.\Gamma), \star\rangle\rangle\end{aligned}$$

$$\begin{aligned}&\Pi : (\Gamma : \mathrm{Ctxt}) \to (A : \mathrm{Ty}(\Gamma)) \to\\ &\qquad\qquad \mathrm{Ty}(\mathrm{cons}(\Gamma, A)) \to \mathrm{Ty}(\Gamma)\\ &\Pi(\Gamma, A, B) = \mathrm{intro}_{\mathrm{B}}(\langle \mathrm{ff}, \langle(\lambda\star.\Gamma), \langle(\lambda\star.A),\\ &\qquad\qquad \langle(\lambda\star.B), \star\rangle\rangle\rangle\rangle).\end{aligned}$$

5. A set-theoretic 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 in [10, 12]. Hence the proof theoretical strength required is, as expected significantly lower but still too strong. See [1] for a more detailed treatment of interpreting type theory in set theory.

5.1. Preliminaries

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

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

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 $A : \mathrm{Type}$ 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.

5.2. Interpretation of expressions

The interpretation of the logical framework is as in [10]:

$$\begin{aligned}[\![\mathrm{Set}]\!] &\coloneqq\!\simeq V_{\mathfrak{m}_0} \qquad [\![\mathrm{Type}]\!] \coloneqq\!\simeq V_{\mathfrak{m}_1}\\ [\![(x : A) \to B]\!]_\rho &\coloneqq\!\simeq \Pi_{y\in[\![A]\!]_\rho} [\![B]\!]_{\rho_{[y\mapsto x]}}\\ [\![\lambda x : A.e]\!]_\rho &\coloneqq\!\simeq \lambda y \in [\![A]\!]_\rho.[\![e]\!]_{\rho_{[y\mapsto x]}}\\ [\![(x : A) \times B]\!]_\rho &\coloneqq\!\simeq \Sigma_{y\in[\![A]\!]_\rho} [\![B]\!]_{\rho_{[y\mapsto x]}}\\ [\![\langle a, b\rangle]\!]_\rho &\coloneqq\!\simeq \langle [\![a]\!]_\rho, [\![b]\!]_\rho\rangle\\ [\![\mathbf{0}]\!] &\coloneqq\!\simeq 0 \qquad [\![\mathbf{1}]\!] \coloneqq\!\simeq 1 \qquad [\![\mathbf{2}]\!] \coloneqq\!\simeq 2\\ [\![\star]\!] &\coloneqq\!\simeq 0 \qquad [\![\mathrm{tt}]\!] \coloneqq\!\simeq 0 \qquad [\![\mathrm{ff}]\!] \coloneqq\!\simeq 1\end{aligned}$$

$$[\![\mathrm{case}_{\mathbf{2}}(x, a, b)]\!]_\rho \coloneqq\!\simeq \begin{cases}[\![a]\!]_\rho & \text{if } [\![x]\!]_\rho = 0\\ [\![b]\!]_\rho & \text{if } [\![x]\!]_\rho = 1\\ \text{undefined} & \text{otherwise}\end{cases}$$

$$[\![!_A]\!]_\rho \coloneqq\!\simeq \text{unique inclusion } \iota_{[\![A]\!]} : \varnothing \to [\![A]\!]_\rho$$

To interpret terms containing SP_{A}, SP_{B}, $\mathrm{Arg}_{\mathrm{A}}$, $\mathrm{Arg}_{\mathrm{B}}$, $\mathrm{Index}_{\mathrm{B}}$, IH_{A}, IH_{B}, $\mathrm{mapIH}_{\mathrm{A}}$, $\mathrm{mapIH}_{\mathrm{B}}$, $\mathrm{nil}_{\mathrm{A}}$, nonind, A-ind,

B-ind, $\mathrm{nil_B}$ and B_n-ind, we first define $[\![\mathrm{SP_A}]\!]$, $[\![\mathrm{SP_B}]\!]$, $[\![\mathrm{Arg_A}]\!]$, ... and interpret

$$\begin{aligned}
[\![\mathrm{SP_A}(D, D')]\!]_\rho &\coloneqq [\![\mathrm{SP_A}]\!]([\![D]\!]_\rho, [\![D']\!]_\rho)\\
&\vdots\\
[\![\mathrm{Arg_A}(\ldots, \gamma, A, \ldots)]\!]_\rho &\coloneqq [\![\mathrm{Arg_A}]\!](\ldots, [\![\gamma]\!]_\rho, [\![A]\!]_\rho, \ldots)\\
&\vdots\\
[\![\mathrm{nonind}(K, \gamma)]\!]_\rho &\coloneqq [\![\mathrm{nonind}]\!]([\![K]\!]_\rho, [\![\gamma]\!]_\rho)\\
&\vdots \quad \text{etc.}
\end{aligned}$$

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

$[\![\mathrm{SP_A}]\!](D, D')$ is defined as the least set such that

$$\begin{aligned}
[\![\mathrm{SP_A}]\!](D, D') = 1 &+ \sum_{K \in [\![\mathrm{Set}]\!]} (K \to [\![\mathrm{SP_A}]\!](D, D'))\\
&+ \sum_{K \in [\![\mathrm{Set}]\!]} [\![\mathrm{SP_A}]\!](D + K, D')\\
&+ \sum_{K \in [\![\mathrm{Set}]\!]} \sum_{h : K \to D} [\![\mathrm{SP_A}]\!](D, D' + K).
\end{aligned}$$

By the inaccessibility of $\mathfrak{m}_1$, there is a regular cardinal $\kappa < \mathfrak{m}_1$ such that for all $K \in [\![\mathrm{Set}]\!]$, we have that the cardinality of K, D, D', $D + K$, $(K \to D)$, $D' + K$ is less than κ. If we now iterate an appropriate operator κ times, we get our solution, which must be an element of $[\![\mathrm{Type}]\!] = V_{\mathfrak{m}_1}$ by the inaccessibility of $\mathfrak{m}_1$.

$$[\![\mathrm{nil_A}]\!] \coloneqq \langle 0, 0\rangle \quad [\![\text{B-ind}(K, h, \gamma)]\!] \coloneqq \langle 3, \langle K, \langle h, \gamma\rangle\rangle\rangle$$
$$[\![\mathrm{nonind}(K, \gamma)]\!] \coloneqq \langle 1, \langle K, \gamma\rangle\rangle \quad [\![\text{A-ind}(K, \gamma)]\!] \coloneqq \langle 2, \langle K, \gamma\rangle\rangle$$

$[\![\mathrm{SP_B}]\!]$ and $[\![\mathrm{nil_B}]\!]$, $[\![\mathrm{B}_n\text{-ind}]\!]$ are defined analogously. $[\![\mathrm{Arg_A}]\!]$, $[\![\mathrm{Arg_B}]\!]$, $[\![\mathrm{Index_B}]\!]$, $[\![\mathrm{IH_A}]\!]$, $[\![\mathrm{IH_B}]\!]$, $[\![\mathrm{mapIH_A}]\!]$ and $[\![\mathrm{mapIH_B}]\!]$ are defined according to their equations, e.g.

$$\begin{aligned}
&[\![\mathrm{Arg_A}]\!](D, D', [\![\mathrm{nil_A}]\!], f, f'_A, f'_B) &&\coloneqq 1\\
&[\![\mathrm{Arg_A}]\!](D, D', [\![\mathrm{nonind}]\!](K, \gamma), f, f'_A, f'_B) &&\coloneqq\\
&\qquad \sum_{k \in K} [\![\mathrm{Arg_A}]\!](D, D', \gamma(k), f, f'_A, f'_B)\\
&[\![\mathrm{Arg_A}]\!](D, D', [\![\text{A-ind}]\!](K, \gamma), f, f'_A, f'_B) &&\coloneqq\\
&\qquad \sum_{j : K \to A} [\![\mathrm{Arg_A}]\!](D + K, D', \gamma, f \sqcup j, f'_A, f'_B)\\
&[\![\mathrm{Arg_A}]\!](D, D', [\![\text{B-ind}]\!](K, h, \gamma), f, f'_A, f'_B) &&\coloneqq\\
&\qquad \sum_{j \in \Pi_{k \in K} B(f(h(k)))} [\![\mathrm{Arg_A}]\!](D, D' + K, \gamma, f,\\
&\qquad\qquad f'_A \sqcup (f \circ h), f'_B \sqcup j).
\end{aligned}$$

Finally, we have to interpret A_{γ_A, γ_B}, B_{γ_A, γ_B}, $\mathrm{intro_A}$, $\mathrm{intro_B}$, $\mathrm{R_A}$ and $\mathrm{R_B}$. Let

$$\begin{aligned}
[\![A_{\gamma_A, \gamma_B}]\!] &\coloneqq A^{\mathfrak{m}_0} & [\![B_{\gamma_A, \gamma_B}]\!](a) &\coloneqq B^{\mathfrak{m}_0}(a)\\
[\![\mathrm{intro_A}]\!](a) &\coloneqq a & [\![\mathrm{intro_B}]\!](a) &\coloneqq b\\
[\![\mathrm{R_A}]\!] &\coloneqq \mathrm{R_A^{\mathfrak{m}_0}} & [\![\mathrm{R_B}]\!] &\coloneqq \mathrm{R_B^{\mathfrak{m}_0}}
\end{aligned}$$

where A^α, B^α and $\mathrm{R_A^\alpha}$, $\mathrm{R_B^\alpha}$ are simultaneously defined by recursion on α as

$$\begin{aligned}
A^\alpha &\coloneqq [\![\mathrm{Arg'_A}]\!](\gamma_A, A^{<\alpha}, B^{<\alpha})\\
B^\alpha(a) &\coloneqq \{b \mid b \in [\![\mathrm{Arg'_B}]\!](\gamma_A, \gamma_B, A^{<\alpha}, B^{<\alpha}, \ldots, B^{<\alpha})\\
&\qquad \wedge [\![\mathrm{Index'_B}]\!](\gamma_A, \gamma_B, A^{<\alpha}, B^{<\alpha}, \ldots, B^{<\alpha}, b) = a\}
\end{aligned}$$

and

$$\begin{aligned}
\mathrm{R_A^\alpha}(g, h, a) &\coloneqq g(a, \mathrm{mapIH'_A}(\gamma_A, [\![A]\!], [\![B]\!], E, E',\\
&\qquad\qquad \mathrm{R_A^{<\alpha}}(g, h), \mathrm{R_B^{<\alpha}}(g, h), a))\\
\mathrm{R_B^\alpha}(g, h, a, b) &\coloneqq h(b, \mathrm{mapIH'_B}(\gamma_A, \gamma_B, [\![A]\!], [\![B]\!], \ldots, [\![B]\!],\\
&\qquad\qquad E, E', \ldots, E', \mathrm{R_A^{<\alpha}}(g, h), \mathrm{R_B^{<\alpha}}(g, h), b))
\end{aligned}$$

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

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

5.3. Soundness of the rules

Theorem 1 (Soundness)

(i) If $\vdash \Gamma$ *context, then* $[\![\Gamma]\!] \downarrow$.

(ii) If $\Gamma \vdash A : E$, *then* $[\![\Gamma]\!] \downarrow$, *and for all* $\rho \in [\![\Gamma]\!]$, $[\![A]\!]_\rho \in [\![E]\!]_\rho$, *and also* $[\![E]\!]_\rho \in [\![\mathit{Type}]\!]$ *if* $E \notin \mathit{Type}$.

(iii) If $\Gamma \vdash A = B : E$, *then* $[\![\Gamma]\!] \downarrow$, *and for all* $\rho \in [\![\Gamma]\!]$, $[\![A]\!]_\rho = [\![B]\!]_\rho$, $[\![A]\!]_\rho \in [\![E]\!]_\rho$ *and also* $[\![E]\!]_\rho \in [\![\mathit{Type}]\!]$ *if* $E \notin \mathit{Type}$.

(iv) $\nvdash a : \mathbf{0}$. □

The proof of the soundness theorem is rather straightforward. For the verification that $[\![A_{\gamma_A, \gamma_B}]\!] \in [\![\mathrm{Set}]\!]$ and $[\![B_{\gamma_A, \gamma_B}]\!] : [\![A_{\gamma_A, \gamma_B}]\!] \to [\![\mathrm{Set}]\!]$, one first verifies $[\![\mathrm{Arg'_A}]\!]$, $[\![\mathrm{Arg'_B}]\!]$, $[\![\mathrm{Index'_B}]\!]$ are monotone in the following sense:

Lemma 2 *Assume* $A \subseteq A'$ *and* $B_i(a) \subseteq B'_i(a)$ *for all* $a \in A$. *Then (we omit the parameters* $\gamma_A : [\![\mathit{SP'_A}]\!]$, $\gamma_B : [\![\mathit{SP'_B}]\!](\gamma_A)$*)*

(i) $[\![\mathit{Arg'_A}]\!](A, B) \subseteq [\![\mathit{Arg'_A}]\!](A', B')$,

(ii) $[\![\mathit{Arg'_B}]\!](A, \vec{B}) \subseteq [\![\mathit{Arg'_B}]\!](A', \vec{B}')$.

(iii) $[\![\mathit{Index'_B}]\!](A', B') \restriction [\![\mathit{Arg'_B}]\!](A, B) = [\![\mathit{Index'_B}]\!](A, B)$. □

One can then prove some useful facts about A^α and B^α by induction on α:

Lemma 3

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

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

The cardinal κ from *(iii)* can be found by considering a regular cardinal of cardinality greater than that of all index sets which starts an inductive argument. By the inaccessibility of $\mathfrak{m}_0$, $\kappa < \mathfrak{m}_0$. With these lemmas, we are done, since $[\![A]\!] = A^{\mathfrak{m}_0} = A^\kappa \in [\![\mathrm{Set}]\!]$ and similarly for $[\![B]\!]$.

6. Conclusions and future work

We have introduced a new principle, induction-induction, for defining sets in Martin-Löf type theory. The principle allows us to simultaneously introduce $A : \text{Set}$ and $B : A \to \text{Set}$, both defined inductively. This principle is used in recent formulations of the metatheory of type theory in type theory [6, 5].

We have formalised the theory and proved it to be consistent. Our formalisation consists of a universe of codes for constructors of inductive-inductive sets. The codes are quite similar to how one would normally like to write down the constructors, although care has to be taken with indices in recursive arguments in B.

Some possible future areas of work include

- **Arbitrary number of levels.** Currently, the principle allows us to define $A : \text{Set}$ and $B : A \to \text{Set}$. It would be interesting to be able to define any number of levels $C : (a : A) \to B(a) \to \text{Set}$, $D : (a : A) \to (b : B(a)) \to C(a, b) \to \text{Set}$ etc. However, on the higher levels, one has to be careful about extensionality problems.
- **Proof theoretic strength.** The model we have developed is obviously, proof theoretically, far stronger than necessary. In order to get sharper bounds on the proof theoretical strength of the theory, it might be possible to construct a model in an extension of Kripke-Platek set theory as in Setzer [20, 21].
- **Categorical semantics.** Ordinary inductive types have a well known categorical semantics in the form of initial algebras [16], and Dybjer and Setzer [11] modelled induction-recursion as initial algebras in slice categories. Can induction-induction be modelled as initial algebras in some suitable category?
- **Generic programming.** The generic elimination rules we have defined leads the way to generic programming also for inductive-inductive definitions.
- **Combining induction-induction and induction-recursion.** Can we combine induction-induction and induction-recursion into one (consistent) theory? If so, this would be a good candidate for an underlying theory of the `mutual` keyword in Agda.

References

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

[2] R. Backhouse. On the meaning and construction of the rules in Martin-Löf's theory of types. In A. Avron, R. Harper, F. Honsell, I. Mason, and G. Plotkin, editors, *Proceedings of the Workshop on general logic, Edinburgh, February, 1987*, volume ECS-LFCS-88-52, 1988.

[3] R. Backhouse, P. Chisholm, G. Malcolm, and E. Saaman. Do-it-yourself type theory. *Formal Aspects of Computing*, 1(1):19–84, 1989.

[4] M. Benke, P. Dybjer, and P. Jansson. Universes for generic programs and proofs in dependent type theory. *Nordic Journal of Computing*, 10:265–269, 2003.

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

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

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

[8] P. Dybjer. Internal type theory. *Lecture Notes in Computer Science*, 1158:120–134, 1996.

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

[10] P. Dybjer and A. 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.

[11] P. Dybjer and A. Setzer. Induction–recursion and initial algebras. *Annals of Pure and Applied Logic*, 124(1-3):1–47, 2003.

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

[13] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. PhD thesis, University of Paris VII, 1972.

[14] P. Hancock, N. Ghani, and D. Pattinson. Representations of stream processors using nested fixed points. *Logical Methods in Computer Science*, 5(3):1 – 17, 2009.

[15] G. Jäger and T. Strahm. Reflections on reflections in explicit mathematics. *Annals of Pure and Applied Logic*, 136(1-2):116–133, 2005.

[16] P. Johann and N. Ghani. Initial algebra semantics is enough! *Lecture Notes in Computer Science*, 4583:207 – 222, 2007.

[17] P. Martin-Löf. *Intuitionistic type theory*. Bibliopolis Naples, 1984.

[18] R. Matthes. Nested datatypes with generalized Mendler iteration: map fusion and the example of the representation of untyped lambda calculus with explicit flattening. *Lecture Notes in Computer Science*, 5133:220–242, 2008.

[19] E. Palmgren. On universes in type theory. In G. Sambin and J. Smith, editors, *Twenty five years of constructive type theory*, pages 191 – 204, Oxford, 1998. Oxford University Press.

[20] A. Setzer. Well-ordering proofs for Martin-Löf's type theory with W-type and one universe. *Annals of Pure and Applied Logic*, 92:113 – 159, 1998.

[21] A. Setzer. Universes in type theory part I – Inaccessibles and Mahlo. In A. Andretta, K. Kearnes, and D. Zambella, editors, *Logic Colloquium '04*, pages 123 – 156. Association of Symbolic Logic, Lecture Notes in Logic 29, Cambridge University Press, 2008.

[22] The Agda Team. The Agda wiki, 2010. Available at `http://wiki.portal.chalmers.se/agda/`.