Universe Abstractions*

Sebastian Reichelt

October 10, 2021

1 Introduction

In this document, we define a mathematical framework that 'automates' many proofs of structural properties, both in a practical sense and as an abstract theory: On the one hand, the accompanying formalization in the Lean 4 theorem prover[**?**] can be used to derive lemmas about a wide range of mathematical objects, starting from relatively few axioms that these objects must satisfy. At the moment, this writeup is mainly intended as documentation for the Lean formalization. On the other hand, as the mathematical content may be interesting and useful in its own right, here we describe it in a conventional fashion without any Lean-specific prerequisites.

The framework is influenced by ideas from Homotopy Type Theory[**?**], but rather than offering a new foundation of mathematics, we transfer those ideas to more traditional mathematical settings: In this document, we work in Zermelo-Fraenkel (ZF) set theory with universes, whereas the Lean formalization is directly based on Lean's foundation without any additional axioms. This shows that our theory is as 'foundation-agnostic' as most other mathematical theories, which has the advantage that it can be used side-by-side with any other theory instead of having to fit all existing mathematics into it. (It can also be formalized *within* Homotopy Type Theory, but we will not cover this.)

The initial insight, on which the rest of the framework is built, is that categories and their functors form a simply-typed lambda calculus in which the S and K combinators exist (theorem 4.8). Therefore, functors can be derived algorithmically from lambda terms, and it is possible to extend this algorithm to incorporate other generic structures. The end result is that functions satisfying certain conditions are automatically functorial.

Similarly, it is possible to automatically derive a proof that two functors are naturally isomorphic from a proof that both functors map a fixed argument to equivalent values (theorem 4.17).

The main goal is to automate proofs of isomorphism invariance in a similar way, but this is till work in progress.

We define the framework in such a way that categories are just one (important) special case, i.e. we define axioms that are satisfied by categories but also by other structures that we define (and conjecturally also by higher categories and certain topological spaces). Therefore, we start with some very generic definitions that unify all of these structures.

2 Universes

As mentioned in the introduction, we work in Zermelo-Fraenkel (ZF) set theory with (a finite number of) Grothendieck universes, which we will use liberally: Since we have formalized our theory in the Lean theorem prover, and the foundation of Lean has been proved to be equiconsistent with ZF plus choice and universes[**?**], we relegate all universe-related details to the Lean formalization.

Therefore, we reserve the word 'universe' for our own use except where specified otherwise:

Definition 2.1. We define a *universe* to be a pair $(I, (S_A)_{A\in I})$ of

- an index set I, the members of which we call *types*, and

*Still searching for a better title.

- a family $(S_A)_{A\in I}$ of sets indexed by a type. We call the members of S_A the *instances* of A.

Given a universe $\mathcal{U} = (I, (S_A)_{A\in I})$, we write

- "$A \in \mathcal{U}$" for "$A \in I$" and
- "$a :_{\mathcal{U}} A$" or usually just "$a : A$" for "$a \in S_A$."

Despite the use of type-theoretic notation, we would like to stress that we are not defining a type theory; our definitions are just *informed* by type theory. Whenever we use the notation "$a : A$," note that in general A is not the set that a is a member of; it can actually be an arbitrary object from which that set can be obtained (similarly to Tarski-style universes in type theory[**?**]).

That makes the definition quite flexible. Fix a Grothendieck universe U. Then we have the following examples of universes according to the definition above.

- For every collection $\mathcal{C} \subseteq U$ of sets in U, $\mathsf{Set}_{\mathcal{C}} := (\mathcal{C}, (S)_{S\in\mathcal{C}})$ is a universe for which ":" and "$\in$" coincide. If $\mathcal{C} = U$, we just write "Set" (again, ignoring all details about Grothendieck universes).
- For every collection $\mathcal{C}$ of structures of the form (S, t_S) with $S \in U$, $(\mathcal{C}, (S)_{(S,t_S)\in\mathcal{C}})$ is a universe.
  As an example, let $\mathcal{C}$ be the set of all U-small groups. Then each type A is a group, and each instance $a : A$ is member of the carrier set of A.
  The universe of categories will be of particular importance, where types are categories and instances are objects in those categories.
  Another example is that $\mathcal{C}$ is a collection of universes. Then each type is itself a universe, and its instances are the types in that universe.
- The previous example generalizes to structures that are built not on sets but on types in a universe $\mathcal{U} = (I, (S_A)_{A\in I})$. Let $\mathcal{C}$ be a collection of structures of the form (A, t_A) with $A \in \mathcal{U}$. Then $(\mathcal{C}, (S_A)_{(A,t_A)\in\mathcal{C}})$ is also a universe.
- For a universe $\mathcal{U} = (I, (S_A)_{A\in I})$, any subset $J \subseteq I$ gives rise to a *subuniverse* $(J, (S_A)_{A\in J})$.
- For any two universes $\mathcal{U} = (I, (S_A)_{A\in I})$ and $\mathcal{V} = (J, (T_B)_{B\in J})$, we have a product universe $\mathcal{U}\times\mathcal{V} := (I\times J, (S_A\times T_B)_{(A,B)\in I\times J})$, as well as a sum universe $\mathcal{U}\uplus\mathcal{V} := (I \uplus J, (R_A)_{A\in I\uplus J})$ with $R_A := S_A$ for $A \in I$ and $R_B := T_B$ for $B \in J$ (where $I \times J$ denotes the Cartesian product and $I \uplus J$ denotes the disjoint union of I and J).

Furthermore, we define two specific universes of interest:

- $\mathsf{Bool} := \mathsf{Set}_{\{0,1\}}$, where 0 and 1 are to be understood as von Neumann ordinals[**?**], so that 0 is an empty type and 1 is a type with a single instance $\varnothing : 1$.[1]
- $\mathsf{Unit} := \mathsf{Set}_{\{1\}}$.

Universes are too generic to prove general statements about all universes, so in the following sections we will define additional structure that universes may or may not have, and prove statements depending on such additional structure.

3 Meta-relations

Definition 3.1. For a set S and a universe $\mathcal{V} = (J, (T_B)_{B\in J})$, we define a *$\mathcal{V}$-valued meta-relation* on S to be a function $(\prec) : S \times S \to J$.

(We reserve the word "relation" for section 8, where we replace the set S with a type in a universe.)

We will write $(\prec)$ in infix form, but note that $(\prec)$ is a function and for $a, b \in S$, the expression $a \prec b$ is not a formula but a type in $\mathcal{V}$. We say that $(\prec)$ is

- *reflexive* if for every $a \in S$ we have an instance $\mathsf{id}_a : a \prec a$,
- *symmetric* if for every $a, b \in S$ and $f : a \prec b$ we have an instance $f^{-1} : b \prec a$, and
- *transitive* if for every $a, b, c \in S$, $f : a \prec b$, and $g : b \prec c$ we have an instance $g \circ f : a \prec c$.

[1] In the Lean formalization of this theory, we have two different universes corresponding to the Lean types `Bool` and `Prop`. Both of them map to Bool in this document.

Let us first justify the terminology, then the notation. So first consider a (set-theoretic) relation $(\sim)$ on S. Then for $\mathcal{V} := \mathsf{Bool}$ and

$$(a \prec b) := \begin{cases} 1 & \text{if } a \sim b, \\ 0 & \text{otherwise} \end{cases}$$

for $a, b \in S$, $(\prec)$ is reflexive/symmetric/transitive whenever $(\sim)$ is.

The notation we use for the three specific instances can be understood category-theoretically: Let S be the set of objects in a category, and let $(a \to b)$ denote the set of morphisms from $a \in S$ to $b \in S$. Then the morphism arrow $(\to)$ is in fact a Set-valued meta-relation on S, and all notations coincide:

- A morphism $f : a \to b$ is indeed an instance of the type $(a \to b) \in \mathsf{Set}$.
- For each $a \in S$, we have $\mathsf{id}_a : a \to a$.
- For an isomorphism $f : a \to b$, we have $f^{-1} : b \to a$.
- For morphisms $f : a \to b$ and $g : b \to c$, we have $g \circ f : a \to c$.

So morphisms in a category form a reflexive and transitive Set-valued meta-relation. Moreover, isomorphisms form a reflexive, symmetric, and transitive Set-valued meta-relation.

Note, however, that in general we do not assume that $(\circ)$ is associative, that id_a is an identity with respect to $(\circ)$, or that f^{-1} is an inverse of f. We will add such assumptions later when needed, but in a more general form that avoids equality. For now, we arbitrarily define the symbol "$\circ$" to be right-associative.

(See also [**?**].)

3.1 Instance equivalences

Definition 3.2. From now on, we will assume every universe $\mathcal{U} = (I, (S_A)_{A \in I})$ to be equipped with an *instance equivalence*, which we define to be

- a universe $\mathcal{V}$, along with
- for each type $A \in \mathcal{U}$, a reflexive, symmetric, and transitive $\mathcal{V}$-valued meta-relation $(\simeq)_A$ on S_A. (In its infix form, we just write "$\simeq$".)

We say that "$\mathcal{U}$ has instances equivalences in $\mathcal{V}$."

The idea behind attaching an instance equivalence to a universe is that different universes have different 'natural' notions of equivalence of the instances of their types.[2] Therefore, we will explicitly define instance equivalences for some, but not all, of the examples given in section 2.

- For every collection $\mathcal{C}$ of sets, the universe $\mathsf{Set}_\mathcal{C}$ has instance equivalences in Bool, by converting the set-theoretic equality relation on each set in $\mathcal{C}$ to a meta-relation as specified in the previous section.
  Note that the equivalences of both Bool and Unit are then actually in Unit (as a subuniverse of Bool), as each type in Bool and Unit has at most one instance.
- Universes of algebraic structures (groups, rings, etc.) have the same instance equivalence as Set. (We may say that they inherit instance equivalences from Set because their instances do not have any internal structure, and this idea also generalizes to structures built on universes other than Set.)
- In the universe of categories, we define $a \simeq b$ to be the set of isomorphisms from a to b, as described in the previous section. That is, the universe of categories has instance equivalences in Set.
  (Similarly, higher categories generally have instance equivalences in some universe of categories or higher categories. We will not investigate this in detail, but it is likely that some concepts in this document correspond closely to higher category theory.)
- In section 7, we will give a definition of equivalence of types in a universe, and this will also be our definition of instance equivalences of universes of universes.

[2] In HoTT, we can assume that all of these instance equivalences are actually equality.

- Subuniverses inherit instance equivalences from their superuniverse.
- If $\mathcal{U}$ has instance equivalences in $\mathcal{V}$, and $\mathcal{U}'$ has instance equivalences in $\mathcal{V}'$, then the product and sum universes $\mathcal{U} \times \mathcal{U}'$ and $\mathcal{U} \uplus \mathcal{U}'$ have instance equivalences in $\mathcal{V} \times \mathcal{V}'$ and $\mathcal{V} \uplus \mathcal{V}'$, respectively.

We will make sure that in the universes we deal with, for every sequence of universes $\mathcal{U}_1, \mathcal{U}_2, \ldots$, where each $\mathcal{U}_k$ has instance equivalences in $\mathcal{U}_{k+1}$, there is a k such that $\mathcal{U}_k = \mathcal{U}_{k+1} = \cdots = \mathsf{Unit}$. However, at this point we do not consider the interactions between the steps in such a sequence.[3]

4 Functors

The next piece of structure that we attach to universes – and the first that lets us derive some concrete results – is a generalization of functions to what we call *functors*. Although functors between categories are indeed one special case of this definition, the conditions are much weaker.

Definition 4.1. For universes $\mathcal{U} = (I, (S_A)_{A \in I})$ and $\mathcal{V} = (J, (T_B)_{B \in J})$, a *functor type* from $A \in \mathcal{U}$ to $B \in \mathcal{V}$ is a type $(A \to B)$ in a universe $\mathcal{W}$ with the following two properties:

- For every instance $F : A \to B$ (which we call a *functor*) we have a function $\mathsf{map}_{ABF} : S_A \to T_B$. Given $a : A$, we will abbreviate "$\mathsf{map}_{ABF}(a)$" to "$F(a)$."
- Moreover, map_{ABF} must respect instance equivalences: For each $a, b : A$, we have a function wd_{ABFab} that maps instances of the type $a \simeq b$ to instances of $F(a) \simeq F(b)$.[4] For an equivalence $e : a \simeq b$, we write "$F(e)$" for $\mathsf{wd}_{ABFab}(e)$ as well, matching the corresponding overloaded notation in category theory.

We say that we *have functors from $\mathcal{U}$ to $\mathcal{V}$ in $\mathcal{W}$* if for every $A \in \mathcal{U}$ and $B \in \mathcal{V}$ we have a functor type $(A \to B) \in \mathcal{W}$. We say that a universe $\mathcal{U}$ has *embedded functors* if we have functors from $\mathcal{U}$ to $\mathcal{U}$ in $\mathcal{U}$.

As is common practice, we define the symbol "$\to$" to be right-associative, i.e. the notation "$A \to B \to C$" stands for "$A \to (B \to C)$." We call such a functor F a *bifunctor*, and we write "$F(a, b)$" for "$F(a)(b)$." (In section 6, we will identify $A \to B \to C$ with $A \times B \to C$, where $A \times B$ is a product type.)

The definition of functors is so generic that we can, in principle, define functors between many different types in many different universes. However, universes with embedded functors are much more rarer. Let us analyze a few examples.

- The functors of Set are just functions, which respect equality and are obviously in Set.
- The universe of categories has embedded functors: For categories $\mathcal{C}$ and $\mathcal{D}$, the (categorical) functors from $\mathcal{C}$ to $\mathcal{D}$ form a category $\mathcal{D}^{\mathcal{C}}$, and we define the type $(\mathcal{C} \to \mathcal{D})$ to be that category.
  We need to verify that functors respect instance equivalences. Recall that for objects a and b of either $\mathcal{C}$ or $\mathcal{D}$, the type $a \simeq b$ is the set of isomorphisms from a to b. Indeed, functors map isomorphisms to isomorphisms.
- The same is true for the universe of groupoids, as a subuniverse of the universe of categories, because the category of functors between two groupoids is a groupoid. (For suitable definitions of instance equivalences, it should also generalize to higher categories and groupoids.)
- The morphisms of some, but not all, algebraic structures are also embedded functors in our sense: In some cases morphisms of a class of structures are themselves instances of that class of structures, when operations on morphisms are defined 'pointwise'.

  For example, if $f, g : S \to T$ are two morphisms of commutative semigroups, then we can define $f \star g$ to be the function that sends each $a \in S$ to $f(a) * g(a)$. This is easily verified to be a morphism as well, based on associativity and commutativity of $(*)$. Moreover $(\star)$ inherits associativity and commutativity from $(*)$, turning the set of morphisms from S to T into a semigroup.

  We may investigate the necessary and sufficient conditions more generally later, but for now, the following non-exhaustive list of structures with embedded functors will have to do:

  – commutative semigroups, monoids, and groups

[3] Readers familiar with HoTT may have noticed that instance equivalences should have the structure of a higher groupoid that is reflected in this sequence. However, we want to specify our assumptions in a more fine-grained manner.

[4] Although ideally we want wd to be a functor as well, recursively, at this point we do not make such an assumption.

  - modules over a ring
  - vector spaces over a field
- Continuous functions between topological spaces can be regarded as functors.
- The universe Unit only has a single type 1, so we must set $(1 \to 1) := 1$. The type 1 has exactly one instance $\varnothing$, so map is completely defined by $\mathsf{map}_{11\varnothing}(\varnothing) := \varnothing$, and wd is completely defined by $\mathsf{wd}_{11\varnothing\varnothing\varnothing} := \varnothing$.
- For Bool, we set

$$\begin{aligned}(0 \to 0) &:= 1,\\ (0 \to 1) &:= 1,\\ (1 \to 0) &:= 0,\\ (1 \to 1) &:= 1,\end{aligned}$$

  matching logical implication.[5] Since 0 has no instances and 1 only has $\varnothing$, we need to define three functions $\mathsf{map}_{00\varnothing}$, $\mathsf{map}_{01\varnothing}$, and $\mathsf{map}_{11\varnothing}$. The first two have empty domains, and the third is again completely defined by $\mathsf{map}_{11\varnothing}(\varnothing) := \varnothing$.

Definition 4.2. For universes $\mathcal{U} = (I, (S_A)_{A\in I})$ and $\mathcal{V} = (J, (T_B)_{B\in J})$ with functors in $\mathcal{W}$, types $A \in \mathcal{U}$ and $B \in \mathcal{V}$, and a function $f : S_A \to T_B$, we define the notation

$$\begin{aligned}F : A &\to B\\ a &\mapsto f(a)\end{aligned}$$

to mean that F is a functor from A to B, and that for each $a : A$ we have an instance equivalence $\mathsf{def}_F(a) : F(a) \simeq f(a)$. We call def_F the *definition of* F.

We extend this notation to bifunctors: Given appropriate universes and types and an appropriate function f, we define

$$\begin{aligned}F : A \to B &\to C\\ (a, b) &\mapsto f(a, b)\end{aligned}$$

to mean that F is a functor from A to $(B \to C)$, and that for each $a : A$ and $b : B$ we have an instance equivalence $\mathsf{def}_F(a, b) : F(a, b) \simeq f(a, b)$.

Likewise for *trifunctors* $F : A \to B \to C \to D$, and so on.

Note that not every function f gives rise to a functor, even though the notation might suggest it (intentionally, as we will see). We will now assert the existence of certain functors axiomatically.

4.1 Functor operations

Definition 4.3. We say that a universe $\mathcal{U}$ with embedded functors has *linear functor operations*[6] if we have the following three functors and six instance equivalences for all types $A, B, C, D \in \mathcal{U}$.

$$\begin{aligned}\mathsf{I}_A : A &\to A\\ a &\mapsto a\end{aligned}$$

$$\begin{aligned}\mathsf{T}_{AB} : A &\to (A \to B) \to B\\ (a, F) &\mapsto F(a)\end{aligned}$$

$$\begin{aligned}\mathsf{B}'_{ABC} : (A \to B) &\to (B \to C) \to A \to C\\ (F, G, a) &\mapsto G(F(a))\end{aligned}$$

Before stating the required equivalences, we derive two additional functors from T and B'. To improve readability, we want to use the symbol "$\circ$" when B' is applied to two arguments, and indeed the functor arrow $(\to)$ is a $\mathcal{U}$-valued meta-relation on the set of types, which, given linear functor operations, is

[5] This can be regarded as a degenerate case of the Curry-Howard correspondence.
[6] The terms "linear" and "affine" refer to linear and affine logic.

- reflexive with $\mathsf{id}_A := \mathsf{I}_A$ and
- transitive with $G \circ F := \mathsf{B}'_{ABC}(F, G)$ for $F : A \to B$ and $G : B \to C$.

Now we define

$$\begin{aligned}\mathsf{C}_{ABC} &:= \mathsf{B}'_{B((B\to C)\to C)(A\to C)}(\mathsf{T}_{BC}) \circ \mathsf{B}'_{A(B\to C)C} \\ &: (A \to B \to C) \to B \to A \to C \\ &\qquad (F, b, a) \mapsto F(a, b)\end{aligned}$$

and

$$\begin{aligned}\mathsf{B}_{ABC} &:= \mathsf{C}(\mathsf{B}'_{ABC}) \\ &: (B \to C) \to (A \to B) \to A \to C \\ &\qquad (G, F, a) \mapsto G(F(a))\end{aligned}$$

(omitting the subscript of C when applying it to an argument) and assert the existence of the following instance equivalences.

$$\begin{aligned}\mathsf{rightId}_{AB} &: \mathsf{B}'_{AAB}(\mathsf{I}_A) \simeq \mathsf{I}_{A\to B} \\ \mathsf{leftId}_{AB} &: \mathsf{B}_{ABB}(\mathsf{I}_B) \simeq \mathsf{I}_{A\to B} \\ \mathsf{swapT}_{AB} &: \mathsf{C}(\mathsf{T}_{AB}) \simeq \mathsf{I}_{A\to B} \\ \mathsf{swapB}'_{ABC} &: \mathsf{C}_{(B\to C)AC} \circ \mathsf{B}'_{ABC} \simeq \mathsf{B}_{AB((B\to C)\to C)}(\mathsf{T}_{BC}) \\ \mathsf{swapB}_{ABC} &: \mathsf{C}_{(A\to B)AC} \circ \mathsf{B}_{ABC} \simeq \mathsf{B}_{A((A\to B)\to B)((A\to B)\to C)}(\mathsf{T}_{AB}) \circ \mathsf{B}_{(A\to B)BC} \\ \mathsf{assoc}_{ABCD} &: \mathsf{B}'_{(B\to C)((C\to D)\to(B\to D))((C\to D)\to(A\to D))}(\mathsf{B}'_{BCD}) \circ \mathsf{B}_{(C\to D)(B\to D)(A\to D)} \circ \mathsf{B}'_{ABD} \simeq \\ &\quad \mathsf{B}_{(B\to C)}(\mathsf{B}'_{ACD}) \circ \mathsf{B}'_{ABC}\end{aligned}$$

Remark. When defining C and B, we implicitly assumed that we can derive instance equivalences

$$\mathsf{def}_{\mathsf{C}}(F, b, a) : \mathsf{C}(F, b, a) \simeq F(a, b) \quad \text{for } F : A \to B \to C,\ b : B,\ a : A$$

and

$$\mathsf{def}_{\mathsf{B}}(G, F, a) : \mathsf{B}(G, F, a) \simeq G(F(a)) \quad \text{for } G : B \to C,\ F : A \to B,\ a : A.$$

In order to obtain these, we first establish one further aspect in which functors behave like functions.

Proposition 4.4. Given linear functor operations, two functors $F, G : A \to B$, an instance equivalence $e : F \simeq G$, and an instance $a : A$, we can obtain an instance equivalence $e(a) : F(a) \simeq G(a)$.

Proof. The functor $\mathsf{T}_{AB}(a)$ must respect instance equivalences, so we have an equivalence

$$\mathsf{T}_{AB}(a)(e) : \mathsf{T}_{AB}(a, F) \simeq \mathsf{T}_{AB}(a, G).$$

Applying the definition of T_{AB} yields an equivalence of the desired type:

$$\begin{aligned}e(a) &:= \mathsf{def}_{\mathsf{T}_{AB}}(a, G) \circ \mathsf{T}_{AB}(a)(e) \circ (\mathsf{def}_{\mathsf{T}_{AB}}(a, F))^{-1} \\ &: F(a) \simeq G(a).\end{aligned}$$

□

Together with the other properties of instance equivalences, this proposition establishes that when constructing an instance equivalence involving functors, we may freely rewrite along other equivalences, i.e. substitute arbitrary subterms. Still, the result will be an explicit construction, which may be important if e.g. equivalences are isomorphisms.[7]

We will usually avoid spelling out the details of such constructions,[8] but for demonstration purposes we will explicitly construct $\mathsf{def}_{\mathsf{C}}$ now.

[7] In terms of type theory, we are simply doing proof-relevant mathematics.

[8] Explicit constructions of all equivalences can be found in the Lean formalization.

Recall that "$\circ$" is just a shorthand for B' in reverse order. So for a given $F : A \to B \to C$, the term $\mathsf{C}(F)$ is exactly the left side of an equivalence given by $\mathsf{def}_{\mathsf{B}'}$:

$$\begin{aligned} e &:= \mathsf{def}_{\mathsf{B}'}(\mathsf{B}'_{A(B\to C)C}, \mathsf{B}'_{B((B\to C)\to C)(A\to C)}(\mathsf{T}_{BC}), F) \\ &: \mathsf{C}(F) \simeq \mathsf{B}'_{A(B\to C)C}(F) \circ \mathsf{T}_{BC}. \end{aligned}$$

Applying proposition 4.4 to e and an instance $b : B$, we obtain

$$e(b) : \mathsf{C}(F, b) \simeq (\mathsf{B}'_{A(B\to C)C}(F) \circ \mathsf{T}_{BC})(b).$$

The right side is again an application of B', so we have

$$\begin{aligned} f &:= \mathsf{def}_{\mathsf{B}'}(\mathsf{T}_{BC}, \mathsf{B}'_{A(B\to C)C}(F), b) \\ &: (\mathsf{B}'_{A(B\to C)C}(F) \circ \mathsf{T}_{BC})(b) \simeq \mathsf{T}_{BC}(b) \circ F, \end{aligned}$$

and, applying transitivity of $(\simeq)$,

$$\begin{aligned} g &:= f \circ e(b) \\ &: \mathsf{C}(F, b) \simeq \mathsf{T}_{BC}(b) \circ F. \end{aligned}$$

Apply proposition 4.4 to g and an instance $a : A$ to obtain

$$g(a) : \mathsf{C}(F, b, a) \simeq (\mathsf{T}_{BC}(b) \circ F)(a).$$

This time we have two relevant definitions

$$\begin{aligned} h &:= \mathsf{def}_{\mathsf{B}'}(F, \mathsf{T}_{BC}(b)) \\ &: (\mathsf{T}_{BC}(b) \circ F)(a) \simeq \mathsf{T}_{BC}(b, F(a)), \\ i &:= \mathsf{def}_{\mathsf{T}}(b, F(a)) \\ &: \mathsf{T}_{BC}(b, F(a)) \simeq F(a, b). \end{aligned}$$

Finally, we can apply transitivity twice to arrive at

$$\begin{aligned} \mathsf{def}_{\mathsf{C}}(F, b, a) &:= i \circ h \circ g(a) \\ &: \mathsf{C}(F, b, a) \simeq F(a, b). \end{aligned}$$

Remarks. For a bifunctor $F : A \to B \to C$, the bifunctor $\mathsf{C}(F) : B \to A \to C$ behaves (up to instance equivalences) like F with swapped arguments. So, informally speaking, a bifunctor is a bifunctor regardless of the order of its arguments.

This should also help clarify the role of B' and B. The existence of B' ensures that we can not only compose two functors $F : A \to B$ and $G : B \to C$ to $G \circ F : A \to C$, but also that composition itself is bifunctorial in F and G. B, then, is the corresponding bifunctor with reversed arguments, and in fact we could assert B as an axiom and derive B' from it instead.[9]

Similarly T says that for each $a : A$, the application of a to an $F : A \to B$ is functorial, and also that this application functor is functorial in a.

Moreover, all of these functors can in fact be regarded as combinators[**?**] in a simply-typed lambda calculus[**?**], so we use established names for these combinators as much as possible.

Definition 4.5. We say that a universe $\mathcal{U}$ with embedded functors has *affine functor operations* if it has linear functor operations and additionally the following functor and equivalences for all types $A, B, C \in \mathcal{U}$.

$$\begin{aligned} \mathsf{K}_{AB} : B &\to A \to B \\ (b, a) &\mapsto b \end{aligned}$$

$$\begin{aligned} \mathsf{rightConst}_{ABC} &: \mathsf{B}'_{B(A\to B)(A\to C)}(\mathsf{K}_{AB}) \circ \mathsf{B}_{ABC} \simeq \mathsf{B}_{BC(A\to C)}(\mathsf{K}_{AC}) \\ \mathsf{leftConst}_{ABC} &: \mathsf{B}'_{C(B\to C)(A\to C)}(\mathsf{K}_{BC}) \circ \mathsf{B}'_{ABC} \simeq \mathsf{K}_{(A\to B)(C\to A\to C)}(\mathsf{K}_{AC}) \end{aligned}$$

[9] Due to a minor technical detail, B' leads to a slightly more convenient definition of C.

Definition 4.6. We say that a universe $\mathcal{U}$ with embedded functors has *full functor operations* if it has affine functor operations and additionally the following functor and equivalences for all types $A, B, C \in \mathcal{U}$.

First we assert the existence of

$$\begin{aligned} \mathsf{W}_{AB} : (A \to A \to B) &\to A \to B \\ (F, a) &\mapsto F(a, a) \end{aligned}$$

and derive

$$\begin{aligned} \mathsf{S'}_{ABC} &:= \mathsf{B}_{(A\to B\to C)(A\to A\to C)(A\to C)}(\mathsf{W}_{AC}) \circ \mathsf{B}_{A(B\to C)(A\to C)} \circ \mathsf{B'}_{ABC} \\ &: (A \to B) \to (A \to B \to C) \to A \to C \\ &\qquad (F, G, a) \mapsto G(a, F(a)) \end{aligned}$$

and

$$\begin{aligned} \mathsf{S}_{ABC} &:= \mathsf{C}(\mathsf{S'}_{ABC}) \\ &: (A \to B \to C) \to (A \to B) \to A \to C \\ &\qquad (G, F, a) \mapsto G(a, F(a)). \end{aligned}$$

Then we assert the existence of the following equivalences.

$$\begin{aligned} \mathsf{dupC}_{AB} &: \mathsf{W}_{AB} \circ \mathsf{C}_{AAB} \simeq \mathsf{W}_{AB} \\ \mathsf{dupK}_{AB} &: \mathsf{W}_{AB} \circ \mathsf{K}_{A(A\to B)} \simeq \mathsf{I}_{A\to B} \\ \mathsf{dupW}_{AB} &: \mathsf{W}_{AB} \circ \mathsf{W}_{A(A\to B)} \simeq \mathsf{W}_{AB} \circ \mathsf{B}_{A(A\to A\to B)(A\to B)}(\mathsf{W}_{AB}) \\ \mathsf{rightDup}_{ABC} &: \mathsf{B'}_{ABC} \circ \mathsf{W}_{AB} \simeq \mathsf{B}_{(B\to C)(A\to A\to C)(A\to C)}(\mathsf{W}_{AC}) \circ \\ &\qquad \mathsf{B'}_{(B\to C)((A\to B)\to(A\to C))(A\to A\to C)}(\mathsf{B}_{ABC}) \circ \mathsf{B'}_{A(A\to B)(A\to C)} \\ \mathsf{leftDup}_{ABC} &: \mathsf{B'}_{(A\to B\to B\to C)(A\to B\to C)(A\to C)}(\mathsf{B}_{A(B\to B\to C)(B\to C)}(\mathsf{W}_{BC})) \circ \mathsf{S'}_{ABC} \simeq \\ &\quad \mathsf{S'}(\mathsf{S'}_{ABC}, \mathsf{B'}_{(A\to B\to B\to C)(A\to B\to C)(A\to C)} \circ \mathsf{S'}_{AB(B\to C)}) \end{aligned}$$

Lemma 4.7. The axioms $\mathsf{rightId}, \mathsf{leftId}, \ldots$ hold trivially (whenever the functors referenced in those axioms are defined) in any universe that satisfies the following extensionality condition:

If A and B are types, $F, G : A \to B$ are functors, and for every $a : A$ we have an equivalence $e(a) : F(a) \simeq G(a)$, then there is also an equivalence $e : F \simeq G$.

(Note that proposition 4.4 says that the converse is always true in a universe with linear functor operations.)

Proof. Under the extensionality condition, the equivalence

$$\mathsf{rightId}_{AB} : \mathsf{B'}_{AAB}(\mathsf{I}_A) \simeq \mathsf{I}_{A\to B}$$

exists if for every $F : A \to B$ we have an equivalence

$$\mathsf{rightId}_{AB}(F) : F \circ \mathsf{I}_A \simeq \mathsf{I}_{A\to B}(F).$$

By straightforward operations on equivalences and another application of the extensionality condition, this equivalence exists if for every $a : A$ we have an equivalence

$$\mathsf{rightId}_{AB}(F, a) : F(a) \simeq F(a),$$

which is given by $\mathsf{id}_{F(a)}$.

The other axioms are analogous. □

Theorem 4.8. The universes with embedded functors that are listed in section 4 have the following functor operations.

- Unit, Bool, Set, and the universes of categories and groupoids have full functor operations.

- The listed universes of algebraic structures have linear functor operations.

Proof. We will give explicit proofs for some important cases; the remaining axioms and universes are analogous.

- In Unit, we take each functor to be the single instance $\varnothing : 1$. All axioms are trivially satisfied.
- For Bool, we can show that all functors exist by doing a case-by-case analysis on the types $A, B, \ldots$ being either 0 or 1. This can be simplified by treating "$\to$" as implication and observing that all implications hold, or even further by applying the Curry-Howard correspondence[**?**].
  Since Bool has instance equivalences in Unit, those are trivially satisfied.
- The functors of Set are just functions. Since functions are extensional,[10] the axioms $\mathsf{rightId}, \mathsf{leftId}, \ldots$ hold by lemma 4.7.
- To show how to construct the required functors in universes of structures, we will take $\mathsf{K}_{\mathcal{CD}}$ for categories $\mathcal{C}$ and $\mathcal{D}$ as a simple but not completely trivial example.

  The definition of $\mathsf{K}_{\mathcal{CD}}$ says that for objects c of $\mathcal{C}$ and d of $\mathcal{D}$, $\mathsf{K}_{\mathcal{CD}}(d, c)$ must be isomorphic to d. In many cases, we do not actually need this flexibility; we can construct a functor that maps exactly to d. That is, we can take this expression at face value and show that it is functorial in all arguments starting from the last.[11]

  So the first step is to show that for a fixed object d of $\mathcal{D}$ we have a functor

$$\begin{aligned} K_d : \mathcal{C} &\to \mathcal{D} \\ c &\mapsto d, \end{aligned}$$

  i.e. for objects c, c' of $\mathcal{C}$ and a morphism $f : c \to c'$ we need to provide a morphism $K_d(f) : K_d(c) \to K_d(c')$. But $K_d(c)$ and $K_d(c')$ are both d, so we can define $K_d(f)$ to be the identity morphism on d. (Then K_d is just the constant functor, of course.)

  The second step is to show that the expression K_d is functorial in d, and this will give the desired functor

$$\begin{aligned} \mathsf{K}_{\mathcal{CD}} : \mathcal{D} &\to \mathcal{D}^{\mathcal{C}} \\ d &\mapsto K_d. \end{aligned}$$

  For objects d, d' of $\mathcal{D}$ and a morphism $f : d \to d'$, we need to provide a natural transformation $\mathsf{K}_{\mathcal{CD}}(f) : K_d \Rightarrow K_{d'}$. Thus, for each object c of $\mathcal{C}$ we need to give a morphism $g_c : K_d(c) \to K_{d'}(c)$. Since $K_d(c) = d$ and $K_{d'}(c) = d'$, we can take $g_c := f$. ($\mathsf{K}_{\mathcal{CD}}$ is known as the diagonal functor.)
- The least straightforward case is the construction of $\mathsf{W}_{\mathcal{CD}}$ for categories $\mathcal{C}$ and $\mathcal{D}$. Following the same strategy as before, first we fix a functor $F : \mathcal{C} \to \mathcal{D}^{\mathcal{C}}$ and need to construct a functor

$$\begin{aligned} W_F : \mathcal{C} &\to \mathcal{D} \\ c &\mapsto F(c)(c). \end{aligned}$$

  So for objects c, c' of $\mathcal{C}$ and a morphism $f : c \to c'$ we need to provide a morphism $W_F(f) : F(c)(c) \to F(c')(c')$. Since $F(f)$ is a natural transformation, the two choices for this morphism are equal:

$$W_F(f) := (F(f))_{c'} \circ F(c)(f) = F(c')(f) \circ (F(f))_c.$$

  W_F is indeed a functor: We have

$$W_F(\mathsf{id}_c) = (\mathsf{id}_{F(c)})_c \circ \mathsf{id}_{F(c)(c)} = \mathsf{id}_{F(c)(c)}$$

[10] The theory can also be formalized in a logic without function extensionality by *defining* functors between sets to be extensional functions. Alternatively/additionally, it is possible to define a universe of setoids, which is similar to Set except that equality is replaced with an equivalence relation.

[11] In the Lean formalization, the axioms are already divided into such individual steps, which is often more useful.

and for morphisms $f : c \to c'$ and $g : c' \to c''$ in $\mathcal{C}$

$$\begin{aligned} W_F(g \circ f) &= (F(g \circ f))_{c''} \circ F(c)(g \circ f) \\ &= (F(g))_{c''} \circ (F(f))_{c''} \circ F(c)(g) \circ F(c)(f) \\ &= (F(g))_{c''} \circ F(c')(g) \circ (F(f))_{c'} \circ F(c)(f) \quad \text{by naturality of } F(f) \\ &= W_F(g) \circ W_F(f). \end{aligned}$$

Now we need to show that W_F is functorial in F to obtain

$$\begin{aligned} \mathsf{W}_{\mathcal{CD}} : (\mathcal{D}^{\mathcal{C}})^{\mathcal{C}} &\to \mathcal{D}^{\mathcal{C}} \\ F &\mapsto W_F. \end{aligned}$$

Given two functors $F, F' : \mathcal{C} \to \mathcal{D}^C$ and a natural transformation $\eta : F \Rightarrow F'$, we need to provide a natural transformation $\mathsf{W}_{\mathcal{CD}}(\eta) : W_F \Rightarrow W_{F'}$. We set $(\mathsf{W}_{\mathcal{CD}}(\eta))_c := (\eta_c)_c$ for each object c of $\mathcal{C}$, and need to verify that this is natural in c. Indeed, for every morphism $f : c \to c'$ we have

$$\begin{aligned} (\eta_{c'})_{c'} \circ W_F(f) &= (\eta_{c'} \circ F(f))_{c'} \circ F(c)(f) \\ &= (F'(f) \circ \eta_c)_{c'} \circ F(c)(f) \quad \text{by naturality of } \eta \\ &= (F'(f))_{c'} \circ (\eta_c)_{c'} \circ F(c)(f) \\ &= (F'(f))_{c'} \circ F'(c)(f) \circ (\eta_c)_c \quad \text{by naturality of } \eta_c \\ &= W_{F'}(f) \circ (\eta_c)_c. \end{aligned}$$

It is easily verified that $\mathsf{W}_{\mathcal{CD}}$ respects identity and composition of natural transformations.

- Although lemma 4.7 does not apply to the universe of categories, the proof strategy for $\mathsf{rightId}, \ldots$ is the same as in the proof of that lemma. The difference is that at each step where we would apply the extensionality condition, instead we need to verify that the equivalences $e(a)$ are natural in a. □

Conjecture 4.9. Theorem 4.8 generalizes at least to n-groupoids and possibly also to n-categories.

Conjecture 4.10. Topological spaces satisfying some mild conditions also have full functor operations.

4.2 Functoriality algorithm

After having shown that several important universes do in fact have linear or even full functor operations, we will now describe an algorithm to prove functoriality automatically, i.e. to obtain a functor that matches a given definition.

This algorithm is actually just a slight adaptation of the well-known algorithm to transform lambda abstractions into terms built from combinators[**?**]. In a simply-typed lambda calculus, this algorithm always terminates. So as long as one is not interested in the specific behavior of the functor (beyond how it maps instances), there is no need to execute the algorithm explicitly – verifying the preconditions given in proposition 4.11 is sufficient.

First, we reduce definitions of bifunctors to definitions of functors; and analogously trifunctors to bifunctors, and so on. This principle closely resembles the proof strategy in theorem 4.8. Given a definition of a bifunctor

$$\begin{aligned} F : A \to B \to C \\ (a, b) \mapsto f(a, b), \end{aligned}$$

first recursively apply the functoriality algorithm to

$$\begin{aligned} F_a : B &\to C \\ b &\mapsto f(a, b) \end{aligned}$$

for fixed $a : A$, to obtain the functor $F_a : B \to C$ and an equivalence $\mathsf{def}_{F_a}(b) : F_a(b) \simeq f(a, b)$ for each $b : B$. Then apply the functoriality algorithm to

$$\begin{aligned} F' : A &\to (B \to C) \\ a &\mapsto F_a \end{aligned}$$

to obtain $F' : A \to B \to C$ and an equivalence $\mathsf{def}_{F'}(a) : F'(a) \simeq F_a$ for each $a : A$. Finally set $F := F'$, apply proposition 4.4 to obtain an equivalence $\mathsf{def}_{F'}(a)(b) : F(a,b) \simeq F_a(b)$, and set $\mathsf{def}_F(a,b) := \mathsf{def}_{F_a}(b) \circ \mathsf{def}_{F'}(a)(b)$.

Due to this reduction, we can limit ourselves to the simple case

$$\begin{aligned} F &: A \to B \\ a &\mapsto f(a) \end{aligned}$$

and perform a (non-exhaustive and non-unique) case split on f.

Definition		F
$\begin{aligned} F &: A \to B \\ a &\mapsto b \end{aligned}$	for $b : B$ (constant with respect to a)	$\mathsf{K}_{AB}(b)$
$\begin{aligned} F &: A \to A \\ a &\mapsto a \end{aligned}$		I_A
$\begin{aligned} F &: A \to B \\ a &\mapsto G(a) \end{aligned}$	for $G : A \to B$	G
$\begin{aligned} F &: A \to C \\ a &\mapsto G(b_a) \end{aligned}$	for $b_a : B$ and $G : B \to C$	$\mathsf{B}'_{ABC}(H,G)$ with $\begin{aligned} H &: A \to B \\ a &\mapsto b_a \end{aligned}$
$\begin{aligned} F &: (B \to C) \to C \\ G &\mapsto G(b) \end{aligned}$	for $b : B$	$\mathsf{T}_{BC}(b)$
$\begin{aligned} F &: A \to C \\ a &\mapsto G_a(b) \end{aligned}$	for $b : B$ and $G_a : B \to C$	$\mathsf{C}_{ABC}(G,b)$ with $\begin{aligned} G &: A \to (B \to C) \\ a &\mapsto G_a \end{aligned}$
$\begin{aligned} F &: A \to B \\ a &\mapsto G_a(a) \end{aligned}$	for $G_a : A \to B$	$\mathsf{W}_{AB}(G)$ with $\begin{aligned} G &: A \to (A \to B) \\ a &\mapsto G_a \end{aligned}$
$\begin{aligned} F &: A \to C \\ a &\mapsto G_a(b_a) \end{aligned}$	for $b_a : B$ and $G_a : B \to C$	$\mathsf{S}'_{ABC}(H,G)$ with $\begin{aligned} H &: A \to B \\ a &\mapsto b_a \end{aligned}$ and $\begin{aligned} G &: A \to (B \to C) \\ a &\mapsto G_a \end{aligned}$

In all cases except the first two, $f(a)$ is a functor application. In fact, the last case is the most general possible functor application, and in a universe with full functor operations, all other cases of functor applications may be regarded as mere optimizations.
Note that a functor application with multiple arguments $F(a_1, \ldots, a_n)$ is really an application of the functor $F(a_1, \ldots, a_{n-1})$ to the argument a_n, and must be treated as such.

One piece of information missing from the table is that the algorithm must produce not only the functor F but also an equivalence $\mathsf{def}_F(a) : F(a) \simeq f(a)$ for each $a : A$. This equivalence is obtained by composing the definition of the combinator with the definitions of the recursively generated functors (if any).

The algorithm may produce terms of the form $\mathsf{C}(\mathsf{B}', \ldots)$ or $\mathsf{C}(\mathsf{S}', \ldots)$. By the definitions of B and S, these can be replaced with $\mathsf{B}(\ldots)$ and $\mathsf{S}(\ldots)$, respectively.

Proposition 4.11. Let $\mathcal{U}$ be a universe with embedded functors and (at least) linear functor operations, and $A_1, \ldots, A_n$ and B be types of $\mathcal{U}$. Consider the (attempted) functor definition

$$\begin{aligned} F &: A_1 \to \ldots \to A_n \to B \\ (a_1, \ldots, a_n) &\mapsto t_{a_1,\ldots,a_n} \end{aligned}$$

with a term[12] $t_{a_1,\ldots,a_n}$ that is one of the following:

- a constant independent of all a_k,

[12] Formally, the attempted functor definition is given by a function $f : S_{A_1} \times \ldots \times S_{A_n} \to S_B$, where S_A is the set of instances of A. However, the proposition is much more readable in its current informal form.

- one of the variables a_k, or
- a functor application $G(b)$ where both G and b are terms that recursively follow the same rules.

Then the functoriality algorithm produces a functor matching the definition of F under the following additional constraints:

- If $\mathcal{U}$ only has linear functor operations (but does not have affine functor operations), each variable a_k must occur exactly once in $t_{a_1,\ldots,a_n}$.
- If $\mathcal{U}$ only has affine functor operations (but does not have full functor operations), each variable a_k must occur at most once in $t_{a_1,\ldots,a_n}$.

Proof. The algorithm given above is total under these conditions. □

Example. Let us consider the simple case where we want to compose a bifunctor $F : A \to B \to C$ with a functor $G : C \to D$. This can be understood in two different ways: We can either construct this composition for fixed but arbitrary F and G, or we can define it as a functor taking F and G as arguments.

For fixed F and G, the functor we want to construct is

$$\begin{aligned} H_{FG} : A \to B \to D \\ (a, b) \mapsto G(F(a, b)). \end{aligned}$$

The term $G(F(a, b))$ only consists of functor applications, references to the constants F and G, and references to the variables a and b, so we know that the functoriality algorithm can produce a functor matching this definition. Since each variable occurs in this term exactly once, the definition is valid in every universe with linear functor operations. If we actually execute the algorithm, we find that it outputs

$$H_{FG} := \mathsf{B}_{BCD}(G) \circ F$$

and

$$\begin{aligned} \mathsf{def}_{H_{FG}}(a, b) &:= \mathsf{def}_{\mathsf{B}_{BCD}}(G, F(a), b) \circ \mathsf{def}_{\mathsf{B'}_{A(B\to C)(B\to D)}}(F, \mathsf{B}_{BCD}(G), a)(b) \\ &: H_{FG}(a, b) \simeq G(F(a, b)). \end{aligned}$$

We can now interpret this construction in specific universes, for example:

- In the universe of categories:
  If $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}$ are categories and $F : \mathcal{A} \to \mathcal{C}^{\mathcal{B}}$ and $G : \mathcal{C} \to \mathcal{D}$ are functors, then we have a functor $H : \mathcal{A} \to \mathcal{D}^{\mathcal{B}}$ such that $H(a)(b)$ is isomorphic to $G(F(a)(b))$ for objects a of $\mathcal{A}$ and b of $\mathcal{B}$.
- In the universe of vector spaces over a field:
  If V, W, X, Y are vector spaces over K, f is a linear map from V to the space of linear maps from W to X, and g is a linear map from X to Y, then we have a linear map h from V to the space of linear maps from W to Y such that $h(v)(w) = g(f(v)(w))$ for vectors v of V and w of W.

If instead we want to construct a functor that takes F and G as arguments, we can either execute the functoriality algorithm directly for

$$\begin{aligned} H : (A \to B \to C) \to (C \to D) \to A \to B \to D \\ (F, G, a, b) \mapsto G(F(a, b)), \end{aligned}$$

or we can use the previous result and construct

$$\begin{aligned} H : (A \to B \to C) \to (C \to D) \to (A \to B \to D) \\ (F, G) \mapsto H_{FG} = \mathsf{B}_{BCD}(G) \circ F. \end{aligned}$$

The algorithm (always) produces the same result in both cases, which is

$$H := \mathsf{B'}_{(C\to D)((B\to C)\to(B\to D))(A\to B\to D)}(\mathsf{B}_{BCD}) \circ \mathsf{B'}_{A(B\to C)(B\to D)}.$$

Remark. The utility of the functoriality algorithm can be a bit subtle, especially when dealing with concrete universes such as the universe of categories. As a rule of thumb, it can be used to prove functoriality of terms where all of the objects that appear in the term are already functorial. Or, from a reverse point of view, it eliminates the need to compose and otherwise manipulate functors explicitly – instead, functors may simply be written as expressions that specify how to map objects, leaving the rest implicit.

4.3 Functor universe

In the universes we covered so far, the relationship between a type and the set of its instances was always very straightforward (although they are truly equal only in $\mathsf{Set}_{\mathcal{C}}$). In the universe that we are going to define now, we will need to be more careful about the distinction: Recall that a type is just a member of an index set, and its instances are actually defined by a family of sets that are indexed by types.

Definition 4.12. Let $\mathcal{U}$ and $\mathcal{V} = (I, (S_B)_{B \in I})$ and $\mathcal{W} = (J, (T_C)_{C \in J})$ be universes such that we have functors from $\mathcal{U}$ to $\mathcal{V}$ in $\mathcal{W}$, and let $A \in \mathcal{U}$. We define the *functor universe* $\mathcal{V}^A$ to be the pair

$$\mathcal{V}^A := (I, (T_{A \to B})_{B \in I}).$$

That is,

- the types (or type indices, to be explicit) of $\mathcal{V}^A$ are the same as the types of $\mathcal{V}$, but
- the instances in $\mathcal{V}^A$ of a type $B \in \mathcal{V}$ are actually functors from A to B.

We let $\mathcal{V}^A$ inherit instance equivalences from $\mathcal{W}$.

If types $B, C, \ldots \in \mathcal{V}$ are also types of $\mathcal{V}^A$ and vice versa, we need to be more explicit about the meaning of instance declarations $b : B$ and also of functor types $B \to C$. To avoid having to annotate all symbols with universes, we adopt the purely notational convention that B^A is technically defined to be equal to $B \in \mathcal{V}$ but should be understood as a type in $\mathcal{V}^A$.

So "$b : B$" is always an abbreviation for "$b :_{\mathcal{V}} B$," whereas "$F : B^A$" is an abbreviation for "$F :_{\mathcal{V}^A} B$," so that F is a functor from A to B even though $B^A = B$.

Full functor operations

We will first concentrate on the case where we have a single universe $\mathcal{U}$ with embedded functors and full functor operations, and a type $A \in \mathcal{U}$.

Proposition 4.13. In this case, $\mathcal{U}^A$ also has embedded functors.

Proof. For $B, C \in \mathcal{U}$ we define $(B^A \to C^A) := (B \to C)^A \in \mathcal{U}^A$. For a functor $G : B^A \to C^A$, which is also an instance of the type $A \to B \to C$, and an instance $F : B^A$, which is also an instance of $A \to B$, we define their functor application in $\mathcal{U}^A$ by $G(F) := \mathsf{S}_{ABC}(G, F)$. This definition respects instance equivalences because it is an application of the functor $\mathsf{S}_{ABC}(G)$. □

Proposition 4.14. $\mathcal{U}$ embeds into $\mathcal{U}^A$: For each type $B \in \mathcal{U}$ and instance $b : B$, we have a corresponding instance $b^A := \mathsf{K}_{AB}(b) : B^A$. This embedding respects instance equivalences and functor application, up to instance equivalence.

Proof. As a functor, K_{AB} respects instance equivalences. Moreover, from $\mathsf{rightConst}$ and dupK we can derive an equivalence

$$\mathsf{embedMap}(F, b) : F^A(b^A) \simeq (F(b))^A$$

for every $F : B \to C$ and $b : B$ with $B, C \in \mathcal{U}$. (Note that the functor application on the left side is defined as $F^A(b^A) = \mathsf{S}_{ABC}(\mathsf{K}_{A(B \to C)}(F), \mathsf{K}_{AB}(b))$.) □

Proposition 4.15. $\mathcal{U}^A$ has full functor operations.

Proof. TODO □

Thus, the functor universe $\mathcal{U}^A$ inherits all properties from $\mathcal{U}$, but it additionally comes with a specific instance $\mathsf{I}_A : A^A$. Intuitively, this instance can be understood as a free variable $a : A$ that acts like any other term of type A. So we can regard $\mathcal{U}^A$ as the universe in which the 'body' t_a of a functor definition

$$\begin{aligned} F : A &\to B \\ a &\mapsto t_a \end{aligned}$$

lives: In the previous section, we regarded $t_a : B$ as a term in $\mathcal{U}$ that depends on $a : A$. But we can also lift t_a to a term $T_G : B^A$ depending on a $G : A^A$, which gives a functor definition for the embedded functor

$$\begin{aligned} F^A : A^A &\to B^A \\ G &\mapsto T_G. \end{aligned}$$

Note that if t_a is a functor application (in $\mathcal{U}$), then T_G is a functor application in $\mathcal{U}^A$, which is given by S. Since (assuming full functor operations), all functor applications can be handled by S in the functoriality algorithm, we can regard the lifting operation and the functoriality algorithm as two different descriptions of the same principle.

If we set $G := \mathsf{I}_A$, then T_G is equivalent to F.[13] More succinctly, we have the following

Proposition 4.16. For every $B \in \mathcal{U}$ and $F : A \to B$ we have an equivalence

$$\mathsf{embedId}(F) : F^A(\mathsf{I}_A) \simeq F.$$

Proof. By rightConst, dupK, and rightId. □

Linear and affine functor operations

TODO

4.4 Extensionality

Theorem 4.17. Functors in a universe $\mathcal{U}$ with full functor operations are "extensional in practice:" If, for types $A, B \in \mathcal{U}$ and functors $F, G : A \to B$, we can derive an equivalence $e(a) : F(a) \simeq G(a)$ purely from generic operations on equivalences and the axioms given in section 4.1, for a fixed but arbitrary $a : A$, then we also have an equivalence $e : F \simeq G$.

Proof. If the term $e(a)$ is composed purely of the generic operations and axioms that we defined, then it can be lifted to an equivalence $e^A(H) : F^A(H) \simeq G^A(H)$ in the functor universe $\mathcal{U}^A$, dependent on an arbitrary functor $H : A^A$. Now set

$$\begin{aligned} e &:= \mathsf{embedId}(G) \circ e^A(\mathsf{I}_A) \circ (\mathsf{embedId}(F))^{-1} \\ &: F \simeq G. \end{aligned}$$

□

Conjecture 4.18. The above theorem extends to universes with linear or affine functor operations.

Due to the way in which the extensionality axioms are split between the three types of functor operations, this conjecture is almost guaranteed to be true. However, without full functor operations we cannot simply embed everything into the functor universe. A possible alternative might be to work with the sum universe $\mathcal{U} \uplus \mathcal{U}^A \uplus \mathsf{Set}_{\{0\}}$.

Remarks. * Efficiency * More direct algorithm * Further structure * Significance regarding natural isomorphisms

A list of axioms that ensure extensionality in SKI combinator calculus is given in [**?**].

Corollary 4.19. Whenever the functoriality algorithm offers multiple alternatives, there are instance equivalences between the resulting functors (in a universe with full functor operations, or unconditionally assuming conjecture 4.18).

[13] This is quite similar to how evaluating a polynomial $f \in R[X]$ at X yields f itself.

Remark. The following alternatives exist. (Brackets indicate that an alternative is redundant because it can also be regarded as an alternative of one or more other alternatives.)

Case		Alternatives
$F : A \to C$ $a \mapsto G(b)$	for $b : B$ and $G : B \to C$	$\mathsf{K}_{AC}(G(b))$ $\mathsf{B}'_{ABC}(\mathsf{K}_{AB}(b), G)$ $\mathsf{C}_{ABC}(\mathsf{K}_{A(B\to C)}(G), b)$ $\left[\mathsf{S}'_{ABC}(\mathsf{K}_{AB}(b), \mathsf{K}_{A(B\to C)}(G))\right]$
$F : A \to B$ $a \mapsto G(a)$	for $G : A \to B$	G $\mathsf{B}'_{AAB}(\mathsf{I}_A, G)$ $\mathsf{W}_{AB}(\mathsf{K}_{A(A\to B)}(G))$ $\left[\mathsf{S}'_{AAB}(\mathsf{I}_A, \mathsf{K}_{A(A\to B)}(G))\right]$
$F : A \to C$ $a \mapsto G(b_a)$	for $G : B \to C$ and $b_a : B$	$\mathsf{B}'_{ABC}(H, G)$ $\mathsf{S}'_{ABC}(H, \mathsf{K}_{B(B\to C)}(G))$ with $H : A \to B$, $a \mapsto b_a$
$F : (B \to C) \to C$ $G \mapsto G(b)$	for $b : B$	$\mathsf{T}_{BC}(b)$ $\mathsf{C}_{(B\to C)BC}(\mathsf{I}_{B\to C}, b)$ $\left[\mathsf{S}'_{(B\to C)BC}(\mathsf{K}_{(B\to C)B}(b), \mathsf{I}_{B\to C})\right]$
$F : A \to C$ $a \mapsto G_a(b)$	for $b : B$ and $G_a : B \to C$	$\mathsf{C}_{ABC}(G, b)$ $\mathsf{S}'_{ABC}(\mathsf{K}_{AB}(b), G)$ with $G : A \to (B \to C)$, $a \mapsto G_a$
$F : A \to B$ $a \mapsto G_a(a)$	for $G_a : A \to B$	$\mathsf{W}_{AB}(G)$ $\mathsf{S}'_{AAB}(\mathsf{I}_A, G)$ with $G : A \to (A \to B)$, $a \mapsto G_a$

Equivalences between all alternatives can be obtained directly from the axioms $\mathsf{rightId}, \mathsf{leftId}, \ldots$ without invoking the functor universe. However, this alone would not be sufficient to ensure that there is an equivalence between the corresponding final results of the functoriality algorithm.

4.5 Functorial meta-relations

5 Singletons

6 Products

7 Equivalences

8 Properties and relations

9 Dependent functors

10 Dependent products