Universe Abstractions[*]

Sebastian Reichelt

December 20, 2021

1 Introduction

In this document, we describe a mathematical framework to automate proofs that follow from structural properties. Such automation can be implemented in theorem provers, and in fact this document is largely intended as documentation for the accompanying formalization in Lean 4 [3]. At the same time, the algorithms are simple enough to be carried out by hand, and quite often the existence of an algorithm can be used to prove a result without actually executing the algorithm, so our results can be useful for informal mathematics as well.

The framework is built around the realization that categories and functors form a simply-typed lambda calculus [8] when objects are regarded up to isomorphism, because functors corresponding to the S and K combinators exist (theorem 4.8).[1] So it is possible to prove that a function between categories is functorial just by observing that it is a term in simply-typed lambda calculus, or, as we state it, its definition is built from functor applications (proposition 4.11). We then generalize this result by including further categorical structures, which corresponds to enriching the lambda calculus with more types.

Functors constructed in this way way are extensional, which means that if for two functors $F, G : \mathcal{C} \to \mathcal{D}$ we have an isomorphism between $F(a)$ and $G(a)$ for a sufficiently 'generic' a (which we make precise), then we can algorithmically construct a natural isomorphism between F and G (theorem 4.20).

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

The framework is influenced by ideas from Homotopy Type Theory [4], but it is built on conventional mathematical foundations: in this document, we work in Zermelo-Fraenkel (ZF) set theory with universes, whereas the Lean formalization is directly based on the Calculus of Inductive Constructions implemented in Lean, without assuming any additional axioms. When we use type-theoretic notation and vocabulary, it always refers to a specific definition that we give in this document.

2 Universes

Although we specifically work in Zermelo-Fraenkel (ZF) set theory with (a finite number of) Grothendieck universes, we will ignore all issues related to the sizes of collections. Explicit universe constraints are given in the Lean formalization. Since the logic implemented in Lean has been proved to be equiconsistent with ZF plus choice and a finite number of inaccessible cardinals [1], it is safe to assume that analogous constraints are sufficient in the latter.

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

[*]Still searching for a better title.

[1]This is different from the interpretation of simply-typed lambda calculus as the internal logic of a Cartesian closed category, mainly due to the different interpretation of equality. In particular, our result applies to *any* category, or more precisely to the collection of categories.

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 and vocabulary, we would like to stress that we are not defining a type theory; our definitions are just *informed* by type theory.[2] The words 'universe', 'type', and 'instance' do not carry any meaning beyond what is defined above.

It is important to distinguish between "$a : A$" and "$a \in A$" because A may in fact be a set that is different from S_A.[3] This difference can be regarded as an indirection, similarly to Tarski-style universes in type theory [9].

Since the definition of a universe does not contain any axioms about types and instances, there are a lot of examples of such universes. Fix a Grothendieck universe U.

- 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 size issues).
- For every collection $\mathcal{C}$ of algebraic 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 (i.e. informally a member 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 [7], so that 0 is an empty type and 1 is a type with a single instance $\varnothing : 1$.[4]
- $\mathsf{Unit} := \mathsf{Set}_{\{1\}}$.

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

[2] One could also say that we are directly working with *models* of typed lambda calculi. In any case, we avoid defining the syntax of lambda calculus.

[3] A is always a set in ZF in theory, but in our case it may also be a set in practice.

[4] 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.

- *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 $g \circ f : a \prec c$.

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 [4], section 2.1.)

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.[5] 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 simple algebraic structures (groups, rings, vector spaces, etc.) have the same instance equivalences as Set. More specifically, universes of algebraic structures inherit their instance equivalences from Set if their elements do not have any internal structure that suggests a different definition of instance equivalence. This 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. Therefore, the universe of categories has instance equivalences in Set instead of Bool.

[5]When formalizing this theory in HoTT, it should be possible to assume that all of these instance equivalences are actually equalities.

(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.
- 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.[6]

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 $\mathsf{apply}_{ABF} : S_A \to T_B$. Given $a : A$, we will abbreviate "$\mathsf{apply}_{ABF}(a)$" to "$F(a)$."
- Moreover, apply_{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 $F(a) \simeq F(b)$.[7] 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 *internal 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 internal 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 internal 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 internal 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

[6]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.

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

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 internal functors will have to do:

  – commutative semigroups, monoids, and groups
  – modules over a ring
  – vector spaces over a field

- Continuous functions between topological spaces can be regarded as internal functors of a universe of topological spaces, by fixing a topology on them.
- 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 apply is completely defined by $\mathsf{apply}_{11\varnothing}(\varnothing) := \varnothing$, and wd is completely defined by $\mathsf{wd}_{11\varnothing\varnothing\varnothing}(\varnothing) := \varnothing$.
- 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.[8] Since 0 has no instances and 1 only has $\varnothing$, we need to define three functions $\mathsf{apply}_{00\varnothing}$, $\mathsf{apply}_{01\varnothing}$, and $\mathsf{apply}_{11\varnothing}$. The first two have empty domains, and the third is again completely defined by $\mathsf{apply}_{11\varnothing}(\varnothing) := \varnothing$.

Definition 4.2. For universes $\mathcal{U}$ and $\mathcal{V}$ with functors in $\mathcal{W}$, types $A \in \mathcal{U}$ and $B \in \mathcal{V}$, and a family $(t_a)_{a:A}$ of instances $t_a : B$ (i.e. a function from the set of instances of A to the set of instances of B) we define the notation

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

to mean that F is a functor from A to B, and that for each $a : A$ we have an instance equivalence

$$\mathsf{def}_F(a) : F(a) \simeq t_a.$$

We call def_F the *definition of* F. If $\mathsf{def}_F(a) = \mathsf{id}_B$ for all $a : A$, we call the definition *strict*.

We extend this notation to bifunctors: Given appropriate universes and types and a family $(t_{ab})_{a:A,b:B}$, we define

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

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 $\mathsf{def}_F(a, b) : F(a, b) \simeq t_{ab}$.

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

Note that not every family of instances 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 internal functors has *linear functor operations*[9] if we have the following three functors and six instance equivalences for all types $A, B, C, D \in \mathcal{U}$.

[8] This can be regarded as a degenerate case of the Curry-Howard correspondence.

[9] The words "linear" and "affine" refer to linear and affine logic. The theory can be modified to incorporate relevant logic as well.

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

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

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

Before stating the required equivalences, we derive two additional functors from T and $\mathsf{B'}$. To improve readability, we want to use the symbol "$\circ$" when $\mathsf{B'}$ 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

- 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.

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

(For the meaning of these equivalences, see lemma 4.7 and proposition 4.15.)

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.[10]

We will usually avoid spelling out the details of such constructions,[11] but for demonstration purposes we will explicitly construct $\mathsf{def}_{\mathsf{C}}$ now.

Recall that "$\circ$" is just a shorthand for B' in reverse order. So for a given $F : A \to B \to C$, the term $\mathsf{C}(F) = (\mathsf{B}'_{B[[B\to C]\to C][A\to C]}(\mathsf{T}_{BC}) \circ \mathsf{B}'_{A[B\to C]C})(F)$ 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.[12]

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 [5] in a simply-typed lambda calculus [8], so we use established names for these combinators as much as possible.

[10]In terms of type theory, we are simply doing proof-relevant mathematics.
[11]Explicit constructions of all equivalences can be found in the Lean formalization.
[12]Due to a minor technical detail, B' leads to a slightly more convenient definition of C.

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

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

$$\begin{aligned} \mathsf{rightConst}_{ABC} &: \mathsf{B'}_{ABC} \circ \mathsf{K}_{AB} \simeq \mathsf{B}_{[B\to C]C[A\to C]}(\mathsf{K}_{AC}) \circ \mathsf{T}_{BC} \\ \mathsf{leftConst}_{ABC} &: \mathsf{B'}_{C[B\to C][A\to C]}(\mathsf{K}_{BC}) \circ \mathsf{B'}_{ABC} \simeq \mathsf{K}_{[A\to B][C\to A\to C]}(\mathsf{K}_{AC}) \end{aligned}$$

Definition 4.6. We say that a universe $\mathcal{U}$ with internal 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

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

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.

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

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 for $\mathsf{rightId}$. 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. □

Alternative proof. The functoriality algorithm presented in the next section does not depend on these axioms. Therefore we may state them in the alternative form given in proposition 4.15, from which lemma 4.7 follows immediately. ☐

Theorem 4.8. The universes with internal 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 [6].
  Since Bool has instance equivalences in Unit, those are trivially satisfied.
- The functors of Set are just functions. Since functions are extensional,[13] 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 strictly to d. That is, we show that the expression defining the functor is indeed functorial in all arguments starting from the last.[14]

  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:

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

[13] 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.

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

W_F is indeed a functor: We have

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

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 to ∞-groupoids and to $(\infty, 1)$-categories.

Conjecture 4.10. Topological spaces satisfying some mild conditions (compactly generated Hausdorff?) 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 [5]. 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 in the following proposition is sufficient.

Proposition 4.11. Let $\mathcal{U}$ be a universe with internal functors and (at least) linear functor operations, $A_1, \ldots, A_n$ and B be types in $\mathcal{U}$, and $(t_{a_1 \ldots a_n})_{a_k : A_k}$ be a family of instances $t_{a_1 \ldots a_n} : B$ that are one of the following:

- a constant independent of all a_k,
- one of the variables a_k, or
- a functor application $G_{a_1 \ldots a_n}(b_{a_1 \ldots a_n})$ such that both $G_{a_1 \ldots a_n}$ and $b_{a_1 \ldots a_n}$ recursively follow the same rules.[15]

[15]Formally, the rules generate a family $(T_B)_{B \in \mathcal{U}}$ of sets $T_B \subseteq F_B$, where F_B is the set of functions from $S_{A_1} \times \cdots \times S_{A_n}$ to S_B, and S_A is the set of instances of A for each $A \in \mathcal{U}$.

Then we have a functor

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

if the following additional constraints are satisfied.

- 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. 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. To obtain a bifunctor

$$\begin{aligned} F : A_1 \to A_2 \to B \\ (a_1, a_2) \mapsto t_{a_1 a_2}, \end{aligned}$$

first recursively obtain the functor

$$\begin{aligned} F_{a_1} : A_2 \to B \\ a_2 \mapsto t_{a_1 a_2} \end{aligned}$$

for constant $a_1 : A_1$. Then recursively obtain the functor

$$\begin{aligned} F' : A_1 \to [A_2 \to B] \\ a_1 \mapsto F_{a_1}. \end{aligned}$$

Finally set $F := F'$, apply proposition 4.4 to $\mathsf{def}_{F'}$ to obtain an equivalence $\mathsf{def}_{F'}(a_1)(a_2) : F(a_1, a_2) \simeq F_{a_1}(a_2)$, and set $\mathsf{def}_F(a_1, a_2) := \mathsf{def}_{F_{a_1}}(a_2) \circ \mathsf{def}_{F'}(a_1)(a_2)$.

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

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

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

Desired result		F
$F : A \to B$ $a \mapsto b$	for $b : B$ (constant with respect to a)	$\mathsf{K}_{AB}(b)$
$F : A \to A$ $a \mapsto a$		I_A
$F : A \to B$ $a \mapsto G(a)$	for $G : A \to B$	G
$F : A \to C$ $a \mapsto G(b_a)$	for $b_a : B$ and $G : B \to C$	$\mathsf{B'}_{ABC}(H, G)$ with $\begin{aligned} H : A \to B \\ a \mapsto b_a \end{aligned}$
$F : [B \to C] \to C$ $G \mapsto G(b)$	for $b : B$	$\mathsf{T}_{BC}(b)$
$F : A \to C$ $a \mapsto G_a(b)$	for $b : B$ and $G_a : B \to C$	$\mathsf{C}_{ABC}(G, b)$ with $\begin{aligned} G : A \to [B \to C] \\ a \mapsto G_a \end{aligned}$
$F : A \to B$ $a \mapsto G_a(a)$	for $G_a : A \to B$	$\mathsf{W}_{AB}(G)$ with $\begin{aligned} G : A \to [A \to B] \\ a \mapsto G_a \end{aligned}$
$F : A \to C$ $a \mapsto G_a(b_a)$	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, t_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 instance equivalence $\mathsf{def}_F(a) : F(a) \simeq t_a$ for each $a : A$. This equivalence is obtained by composing the definition of the combinator with the definitions of the recursively obtained functors that are passed to the combinator as arguments (if any). □

Remark. 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.

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 done 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.

The algorithm becomes even more useful by incorporating further structure that can be described abstractly in terms of universes and functors. Then, similarly to the example above, we can algorithmically construct functors that involve such structure, and the resulting construction will be valid in many universes. Whenever something is "obviously functorial," it probably has a universe-based interpretation so that functoriality indeed does not need to be proved.

Proposition 4.12. Applying the functoriality algorithm to the definition of any of the combinators $\mathsf{K}_{AB}, \mathsf{I}_A, \mathsf{B'}_{ABC}, \mathsf{B}_{ABC}, \mathsf{T}_{AB}, \mathsf{C}_{ABC}, \mathsf{W}_{AB}, \mathsf{S'}_{ABC}, \mathsf{S}_{ABC}$ results in exactly that combinator.

Proof. This is a simple exercise in executing the functoriality algorithm. □

Proposition 4.13. In a universe with linear/affine/full functor operations, the functoriality algorithm is surjective in the sense that it outputs all functors that can be built from the combinators that are valid in that universe.

Proof. For a functor $F : A_1 \to \cdots \to A_n \to B$, applying the functoriality algorithm to

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

produces $F' = F$ by repeated application of the third case in the algorithm. This is valid in any universe because each a_k is used exactly once, and of course F can be a combinator. □

Although the proof is trivial, the proposition is important because we will often establish something "for all families of terms that satisfy the constraints of proposition 4.11," and in those case a suitably modified claim also holds "for all functors that can be built from combinators."

4.3 Extensionality/naturality algorithm

In this section, we will introduce an algorithm to prove equivalences of functors given suitable equivalences of their values, i.e. to prove extensionality (which, in the universe of categories, corresponds to naturality). First, we introduce a notation for such equivalences.

Definition 4.14. Let $A_1 \ldots, A_n, B$ be types in (potentially different) universes with functors such that we have a functor type $[A_1 \to \cdots \to A_n \to B]$, and $(s_{a_1 \ldots a_n})_{a_k : A_k}$ and $(t_{a_1 \ldots a_n})_{a_k : A_k}$ be two families of instances $s_{a_1 \ldots a_n}, t_{a_1 \ldots a_n} : B$. Then we define the notation

$$\begin{aligned} &(A_1 \to \cdots \to A_n \to B) \\ &\quad e(a_1, \ldots, a_n) \ : \ s_{a_1 \ldots a_n} \simeq t_{a_1 \ldots a_n} \end{aligned}$$

to mean that we have an equivalence $e : F_s \simeq F_t$ between the two functors F_s and F_t that are obtained by applying the functoriality algorithm to $(s_{a_1 \ldots a_n})_{a_k : A_k}$ and $(t_{a_1 \ldots a_n})_{a_k : A_k}$, respectively, taking the first possible alternative at each step that is nondeterministic.

We say that an equivalence $e_{a_1 \ldots a_n} : s_{a_1 \ldots a_n} \simeq t_{a_1 \ldots a_n}$ is *extensional* in $a_1, \ldots, a_n$ if we can obtain an equivalence e as defined above.

Remark. In particular, if F is the functor obtained for $(t_{a_1 \ldots a_n})_{a_k : A_k}$, we have

$$\begin{aligned} &(A_1 \to \cdots \to A_n \to B) \\ &\quad \mathsf{id}_F(a_1, \ldots, a_n) \ : \ F(a_1, \ldots, a_n) \simeq t_{a_1 \ldots a_n}, \end{aligned}$$

i.e. the definition def_F produced by the functoriality algorithm is extensional.

Proposition 4.15. Using this notation, we can understand the axioms introduced in section 4.1 as asserting the extensionality of certain equivalences that hold by definition.

For linear functor operations:

$$([A \to B] \to A \to B)$$
$$\mathsf{rightId}_{AB}(F, a) \;:\; F(\mathsf{I}_A(a)) \simeq F(a)$$

$$([A \to B] \to [A \to B])$$
$$\mathsf{rightId}_{AB}(F) \;:\; F \circ \mathsf{I}_A \simeq F \quad \text{(derived)}$$

$$([A \to B] \to A \to B)$$
$$\mathsf{leftId}_{AB}(F, a) \;:\; \mathsf{I}_B(F(a)) \simeq F(a)$$

$$([A \to B] \to [A \to B])$$
$$\mathsf{leftId}_{AB}(F) \;:\; \mathsf{I}_B \circ F \simeq F \quad \text{(derived)}$$

$$([A \to B] \to A \to B)$$
$$\mathsf{swapT}_{AB}(F, a) \;:\; \mathsf{T}_{AB}(a, F) \simeq F(a)$$

$$([A \to B] \to A \to [B \to C] \to C)$$
$$\mathsf{swapB'}_{ABC}(F, a, G) \;:\; (G \circ F)(a) \simeq G(F(a))$$

$$([B \to C] \to A \to [A \to B] \to C)$$
$$\mathsf{swapB}_{ABC}(G, a, F) \;:\; (G \circ F)(a) \simeq G(F(a))$$

$$([A \to B] \to [B \to C] \to [C \to D] \to A \to D)$$
$$\mathsf{assoc}_{ABCD}(F, G, H, a) \;:\; (H \circ G)(F(a)) \simeq H(G(F(a)))$$

$$([A \to B] \to [B \to C] \to [C \to D] \to [A \to D])$$
$$\mathsf{assoc}_{ABCD}(F, G, H) \;:\; (H \circ G) \circ F \simeq H \circ (G \circ F) \quad \text{(derived)}$$

For affine functor operations:

$$(B \to [B \to C] \to A \to C)$$
$$\mathsf{rightConst}_{ABC}(b, G, a) \;:\; G(\mathsf{K}_{AB}(b, a)) \simeq G(b)$$

$$(B \to [B \to C] \to [A \to C])$$
$$\mathsf{rightConst}_{ABC}(b, G) \;:\; G \circ \mathsf{K}_{AB}(b) \simeq \mathsf{K}_{AC}(G(b)) \quad \text{(derived)}$$

$$([A \to B] \to C \to A \to C)$$
$$\mathsf{leftConst}_{ABC}(F, c, a) \;:\; \mathsf{K}_{BC}(c, F(a)) \simeq c$$

$$([A \to B] \to C \to [A \to C])$$
$$\mathsf{leftConst}_{ABC}(F, c) \;:\; \mathsf{K}_{BC}(c) \circ F \simeq \mathsf{K}_{AC}(c) \quad \text{(derived)}$$

For full functor operations:

$$([A \to A \to B] \to A \to B)$$
$$\mathsf{dupC}_{AB}(F, a) \;:\; \mathsf{C}(F, a, a) \simeq F(a, a)$$

$$([A \to B] \to A \to B)$$
$$\mathsf{dupK}_{AB}(F, a) \;:\; \mathsf{K}_{A[A\to B]}(F, a, a) \simeq F(a)$$

$$([A \to A \to B] \to [B \to C] \to A \to C)$$
$$\mathsf{rightDup}_{ABC}(F, G, a) \;:\; G(\mathsf{W}_{AB}(F, a)) \simeq G(F(a, a))$$

$$([A \to B] \to [A \to B \to B \to C] \to A \to C)$$
$$\mathsf{leftDup}_{ABC}(F, G, a) \;:\; \mathsf{W}_{BC}(G(a), F(a)) \simeq G(a, F(a), F(a))$$

In the rest of this section, we will show that these axioms imply the extensionality of *all* such equivalences.

Proposition 4.16. The following equivalences can be derived from the extensionality axioms.

1. $\mathsf{swapSwap}(F) : \mathsf{C}(\mathsf{C}(F)) \simeq F$
   for $F : A \to B \to C$
2. $\mathsf{dupLeftC}(F, G) : \mathsf{W}_{AC}(\mathsf{C}(G) \circ F) \simeq \mathsf{S'}_{ABC}(F, G)$
   for $F : A \to B$ and $G : A \to B \to C$
3. $\mathsf{dupW}(F) : \mathsf{W}_{A[A\to B]}(\mathsf{W}_{AB} \circ F) \simeq \mathsf{W}_{AB}(\mathsf{W}_{A[A\to B]}(F))$
   for $F : A \to A \to A \to B$
4. $\mathsf{assocS}(F, G, H) : \mathsf{S'}_{ABD}(F, \mathsf{S'}_{A[B\to C][B\to D]}(G, \mathsf{B}_{BCD} \circ H)) \simeq \mathsf{S'}_{ACD}(\mathsf{S'}_{ABC}(F, G), H)$
   for $F : A \to B$, $G : A \to B \to C$, and $H : A \to C \to D$

Proof. We will only list the axioms that play a central role in the proof.

1. By swapT.
2. By dupC and $\mathsf{swapSwap}$.
3. By $\mathsf{leftDup}$.
4. By assoc and dupW.

Full proofs (and a lot of further equivalences) are contained in the Lean formalization. □

Lemma 4.17. A family $(e_{a_1,a_2})_{a_k:A_k}$ of equivalences is extensional in (a_1, a_2) if it is extensional in a_2 for fixed a_1 and the resulting family of equivalences is extensional in a_2.

Proof. This follows directly from the construction of the functoriality algorithm. □

Therefore, in the following lemmas, we will restrict ourselves to a single parameter $a : A$.

Lemma 4.18. When a single step in the functoriality algorithm is nondeterministic, the resulting functors are equivalent.

Proof. 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 the alternatives given on the right are easily derived from the extensionality axioms. □

Lemma 4.19. For each combinator F that is defined axiomatically in section 4.1, the applied definition

$$\mathsf{def}_F(t_a, t'_a, \ldots)$$

is extensional in $a : A$ if the families $(t_a)_{a:A}, (t'_a)_{a:A}, \ldots$ satisfy the conditions of proposition 4.11. Specifically, we have the following equivalences for suitable families $(b_a), (c_a), (G_a), (H_a)$.

1. $$\mathsf{extI}_{AB}(a) \ \overset{(A \to B)}{:} \ \mathsf{I}_B(b_a) \simeq b_a$$
2. $$\mathsf{extT}_{ABC}(a) \ \overset{(A \to C)}{:} \ \mathsf{T}_{BC}(b_a, G_a) \simeq G_a(b_a)$$
3. $$\mathsf{extB'}_{ABCD}(a) \ \overset{(A \to D)}{:} \ \mathsf{B'}_{BCD}(G_a, H_a, b_a) \simeq H_a(G_a(b_a))$$
4. $$\mathsf{extK}_{ABC}(a) \ \overset{(A \to C)}{:} \ \mathsf{K}_{BC}(c_a, b_a) \simeq c_a$$
5. $$\mathsf{extW}_{ABC}(a) \ \overset{(A \to C)}{:} \ \mathsf{W}_{BC}(G_a, b_a) \simeq G_a(b_a, b_a)$$

Proof. According to definition 4.14, for each of the five combinators we need to execute the functoriality algorithm on both sides of the equivalence given above, taking the first alternative if the algorithm is nondeterministic, and construct an equivalence between the resulting functors. Thus, the required constructions are different depending on whether each argument is constant, exactly a, or dependent on a.

Fortunately, the previous lemma allows us to combine many of these cases,[16] except that in a universe with only linear or affine functor operations, we need to provide an individual construction for each case where exactly one of the arguments depends on a.

This leaves us with the following list. In each case, for a family $(t_a)_{a:A}$ of terms $t_a : B$, let $F_t : A \to B$ denote the functor constructed for $(t_a)_{a:A}$ under the assumption that t_a depends on a but is not exactly a.

1. $\mathsf{extI}_{AB} := \mathsf{leftId}_{AB}(F_b)$.
2. If b_a is a constant $b : B$, then both sides of extT_{ABC} are $\mathsf{C}(F_G, b)$.
   If G_a is a constant $G : B \to C$, we construct an equivalence $\mathsf{extT}_{ABC} : \mathsf{C}(\mathsf{T}_{BC} \circ F_b, G) \simeq G \circ F_b$ from swapT.
   If both b_a and G_a depend on a, we construct an equivalence $\mathsf{extT}_{ABC} : \mathsf{S'}(F_G, \mathsf{T}_{BC} \circ F_b) \simeq \mathsf{S'}(F_b, F_G)$ from $\mathsf{dupLeftC}$.
3. If G_a and H_a are constants $G : B \to C$ and $H : C \to D$, the required equivalence is $\mathsf{extB'}_{ABCD} := \mathsf{assoc}_{ABCD}(F_b, G, H)$.
   If G_a and b_a are constants $G : B \to C$ and $b : B$, we construct an equivalence $\mathsf{extB'}_{ABCD} : \mathsf{C}(\mathsf{B'}_{BCD}(G) \circ F_H, b) \simeq \mathsf{C}(F_H, G(b))$ from $\mathsf{swapB'}$.
   If H_a and b_a are constants $H : C \to D$ and $b : B$, we construct an equivalence $\mathsf{extB'}_{ABCD} : \mathsf{C}(\mathsf{B}_{BCD}(H) \circ F_G, b) \simeq H \circ \mathsf{C}(F_G, b)$ from swapB.
   If more than one term depends on a, the required equivalence is $\mathsf{extB'}_{ABCD} := \mathsf{assocS}(F_b, F_G, F_H)$.
4. If c_a is a constant $c : C$, the required equivalence is $\mathsf{extK}_{ABC} := \mathsf{leftConst}_{ABC}(F_b, c)$.
   If b_a is a constant $b : B$, we construct an equivalence $\mathsf{extK}_{ABC} : \mathsf{C}_{ABC}(\mathsf{K}_{BC} \circ F_c, b) \simeq F_c$ from $\mathsf{rightConst}$.
   If both b_a and c_a depend on a, we construct an equivalence $\mathsf{extK}_{ABC} : \mathsf{S'}_{ABC}(F_b, \mathsf{K}_{BC} \circ F_c) \simeq F_c$ from $\mathsf{rightConst}$, $\mathsf{leftConst}$, dupK, and dupC.
5. The construction of extW_{ABC} is only valid in a universe with full functor operations. Therefore we can assume that both G_a and b_a depend on a and set $\mathsf{extW}_{ABC} := \mathsf{leftDup}_{ABC}(F_b, F_G)$. □

[16] A practical implementation of the algorithm may still want to handle each case individually to produce shorter terms.

Theorem 4.20. All equivalences that can be constructed from the axioms defined in section 4.1 are extensional in all variables (in the sense of definition 4.14).

Specifically, let $\mathcal{U}$ be a universe, $A_1, \ldots, A_n, B$ be types in $\mathcal{U}$, and $(s_{a_1 \ldots a_n})_{a_k : A_k}$ and $(t_{a_1 \ldots a_n})_{a_k : A_k}$ be families of instances $s_{a_1 \ldots a_n}, t_{a_1 \ldots a_n} \in B$, and $(e_{a_1 \ldots a_n})_{a_k : A_k}$ be a family of instance equivalences $e_{a_1 \ldots a_n} : s_{a_1 \ldots a_n} \simeq t_{a_1 \ldots a_n}$ that are one of the following:

- a constant independent of all a_k (implying that $s_{a_1 \ldots a_n}$ and $t_{a_1 \ldots a_n}$ are also constant),
- $\mathsf{id}_{s_{a_1 \ldots a_n}}$ (implying $s_{a_1 \ldots a_n} = t_{a_1 \ldots a_n}$),
- $f^{-1}_{a_1 \ldots a_n}$ for an equivalence $f_{a_1 \ldots a_n}$ that recursively follows the same rules,
- $g_{a_1 \ldots a_n} \circ f_{a_1 \ldots a_n}$ for equivalences $f_{a_1 \ldots a_n}$ and $g_{a_1 \ldots a_n}$ that recursively follow the same rules,
- $G_{a_1 \ldots a_n}(f_{a_1 \ldots a_n})$ for a functor $G_{a_1 \ldots a_n}$ that satisfies the constraint of proposition 4.11 and an equivalence $f_{a_1 \ldots a_n}$ that recursively satisfies the constraints of this theorem,
- an application of a functor definition $\mathsf{def}_G(x_{a_1 \ldots a_n}, x'_{a_1 \ldots a_n}, \ldots)$, where G is one of the combinators that were defined axiomatically in section 4.1, such that $x_{a_1 \ldots a_n}, x'_{a_1 \ldots a_n}, \ldots$ satisfy the constraint of proposition 4.11.

Then we can construct an equivalence $e : F_s \simeq F_t$ between the two functors F_s and F_t that are obtained by applying the functoriality algorithm to $(s_{a_1 \ldots a_n})_{a_k : A_k}$ and $(t_{a_1 \ldots a_n})_{a_k : A_k}$, respectively. I.e. using the notation introduced in definition 4.14, we have an equivalence

$$\begin{aligned} &(A_1 \to \cdots \to A_n \to B) \\ &\quad e(a_1, \ldots, a_n) \ : \ s_{a_1 \ldots a_n} \simeq t_{a_1 \ldots a_n}. \end{aligned}$$

Proof. As in the proof of proposition 4.11, we can restrict ourselves to the case that $n = 1$, i.e. we replace $a_1, \ldots, a_n$ with a single $a : A$. We recursively construct e based on the different cases.

- If s_a and t_a are constants $s, t : B$, then $F_s = \mathsf{K}_{AB}(s)$ and $F_t = \mathsf{K}_{AB}(t)$. Therefore, if e_a is also a constant f, we can set $e := \mathsf{K}_{AB}(f)$.
- If $e_a = id_{s_a}$ implying $s_a = t_a$ for all a, then $F_s = F_t$, and we can set $e := \mathsf{id}_{A \to B}$.
- If $e_a = f_a^{-1}$, set $e := f^{-1}$, where f is the equivalence obtained recursively for f_a.
- Likewise for $e_a = g_a \circ f_a$.
- If $e_a = G_a(f_a) : G_a(u_a) \simeq G_a(v_a)$ for an appropriate functor $G_a : C \to B$ and equivalence $f_a : u_a \simeq v_a$ with $u_a, v_a : C$, then the definitions of F_s and F_t depend on whether G_a, u_a, and v_a are constant, exactly a, or dependent on a. If u_a and v_a differ in this respect, first obtain equivalent alternatives for F_s and F_t according to lemma 4.18, so that F_s and F_t are both applications of the same combinator H. Then e is obtained by applying H to f in the appropriate way, where f is the equivalence obtained recursively for f_a.
- If $e_a = \mathsf{def}_G(x_a, x'_a, \ldots)$ for a combinator G, set e according to lemma 4.19. □

Corollary 4.21. If $F, G : A_1 \to \cdots \to A_n \to B$ are functors and $(e_{a_1 \ldots a_n})_{a_k : A_k}$ is a family of instance equivalences $e_{a_1 \ldots a_n} : F(a_1, \ldots, a_n) \simeq G(a_1, \ldots, a_n)$ that satisfy the constraints of theorem 4.20, then we have an equivalence $e : F \simeq G$.

Corollary 4.22. Whenever the functoriality algorithm is nondeterministic, there are equivalences between the resulting functors.
(This strengthens lemma 4.18, as an equivalence in each step does not necessarily translate to an equivalence in the final result.)

Remarks. The triviality or nontriviality of this theorem depends on the amount of structure that instance equivalences of $\mathcal{U}$ have. In the universe of categories, it is already quite significant: If for two functors $F, G : \mathcal{C} \to \mathcal{D}$ we can derive a family $(e_c)_{c \in \mathcal{C}}$ of isomorphisms from the axioms given in the previous sections, then F and G are naturally isomorphic.

As with the functoriality algorithm, the usefulness of the theorem increases with additional structure that we define on universes. When such structure is equipped with appropriate functors and equivalences, theorem 4.20 applies to that additional structure as well.

The result can also be interpreted as an extensionality theorem in lambda calculus, or more specifically in combinatory logic. A proof that extensionality in SKI combinator calculus follows from five axioms is given in [2], theorem 8.14. Our result is similar, the key differences being that we also incorporate linear and affine logic, and that our theorem is restricted to simply-typed lambda calculus.

4.4 Functor universe

The functoriality and extensionality algorithms have a metamathematical character. In this section, we present a purely mathematical approach that offers alternative proofs of proposition 4.11 and theorem 4.20.

Specifically, we introduce a *functor universe* with the following two properties.

- The existence of internal functors in the functor universe corresponds to the existence of the S combinator in the base universe, and implies proposition 4.11.
- Functor operations (including the existence of S) in the functor universe correspond to extensionality in the base universe, and imply theorem 4.20.

First, a word of warning: In the concrete 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 $\mathsf{Set}_{\mathcal{C}}$). However, in the functor universe 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 4.23. 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 those 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, then the symbols ":" and "$\to$" become ambiguous. To avoid having to annotate each symbol with a universe, we adopt the purely notational convention that for each type $B \in \mathcal{V}$ we define $B^A := B$ except that B^A should be understood as a type in $\mathcal{V}^A$. (Note that we do *not* extend this convention to instances of types; instead we will use the same notation for an embedding operation.)

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, not an instance of B.

Full functor operations

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

Proposition 4.24. *In this case, $\mathcal{U}^A$ also has internal functors, defined by $[B^A \to C^A] := [B \to C]^A$ for $B, C \in \mathcal{U}$.*

Proof. 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 4.25. *$\mathcal{U}$ 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, for $G : B \to C$ and $b : B$ with $B, C \in \mathcal{U}$ we have an equivalence

$$\mathsf{embedMap}(G, b) : G^A(b^A) \overset{\text{def}}{=} \mathsf{S}_{ABC}(\mathsf{K}_{A[B\to C]}(G), \mathsf{K}_{AB}(b)) \simeq \mathsf{K}_{AC}(G(b)) \overset{\text{def}}{=} (G(b))^A$$

by lemma 4.18. □

Proposition 4.26. $\mathcal{U}^A$ has full functor operations, defined by $\mathsf{I}_{B^A} := (\mathsf{I}_B)^A$, $\mathsf{T}_{(B^A)(C^A)} := (\mathsf{T}_{BC})^A, \ldots$ for $B, C \in \mathcal{U}$.

Proof. Clearly these instances are functors of the correct type, and by $\mathsf{embedMap}$ they map embedded instances of types in $\mathcal{U}$ to the correct values. However, we need to verify that they also map all other instances of $\mathcal{U}^A$ to the values specified by their definition, up to equivalence. E.g. for I_{B^A} we need to provide an equivalence

$$\mathsf{def}_{\mathsf{I}_{B^A}}(F) : \mathsf{I}_{B^A}(F) \overset{\text{def}}{=} \mathsf{S}_{ABC}(\mathsf{K}_{AB}(\mathsf{I}_B), F) \simeq F$$

for $F : B^A$, and likewise for the other four combinators.

These equivalences can be obtained either via the algorithm given in theorem 4.20, or directly from the lemmas leading up to that theorem.

Instance equivalences $\mathsf{rightId}, \mathsf{leftId}, \ldots$ in $\mathcal{U}^A$ are trivially obtained from the corresponding equivalences in $\mathcal{U}$ by repeated application of $\mathsf{embedMap}$. □

Proposition 4.27. If $G : B \to C$ $(B, C \in \mathcal{U})$ is a functor with definition

$$\begin{aligned} G : B &\to C \\ b &\mapsto t_b \end{aligned}$$

where t_b follows the constraints in proposition 4.11, then we can lift the family $(t_b)_{b:B}$ to a family $(T_F)_{F:B^A}$ of instances of C^A, which gives a definition

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

for the embedded functor G^A.

Proof. We recursively define T_F based on the three possibilities for t_b.

- If t_b is a constant c independent of b, we set $T_F := c^A$.
- If $t_b = b$, we set $T_F := F$.
- If t_b is a functor application, we set T_F to the application of the lifted functor to the lifted term.

Then, executing the functoriality algorithm for $(t_b)_{b:B}$ and $(T_F)_{F:B^A}$ gives results $G' : B \to C$ and $G'' : B^A \to C^A$ that only consist of functor applications of constant terms, such that each term x in G' corresponds exactly to x^A in G''. Thus by repeated application of $\mathsf{embedMap}$ we obtain an equivalence between G'^A and G''. Composing this equivalence with the definitions of G, G', and G'' yields the required equivalence $\mathsf{def}_{G^A}(F) : G^A(F) \simeq T_F$ for $F : B^A$. □

We can use this proposition to obtain an alternative proof of proposition 4.11 as follows. Given a family $(t_a)_{a:A}$ of terms $t_a : B$ that satisfy the constraints of proposition 4.11, instead of executing the functoriality algorithm we may lift (t_a) to a family $(T_G)_{G:A^A}$ of terms $T_G : B^A$, and set the result to T_{I_A}.

This functor is equivalent to the functor F constructed in the proof of proposition 4.11 because according to proposition 4.27, $F^A(G)$ is equivalent to T_G for each $G : A^A$, and due to the following

Proposition 4.28. 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 $\mathsf{rightId}$. □

Remark. This shows that the functor universe $\mathcal{U}^A$ can be regarded as $\mathcal{U}$ with an adjoined element $\mathsf{I}_A : A^A$, which behaves similarly to the symbol X in a polynomial ring $R[X]$.

Proposition 4.29. Let $B, C \in \mathcal{U}$, $(s_b)_{b:B}$ and $(t_b)_{b:B}$ be families of instances $s_b, t_b : C$, and $(e_b)_{b:B}$ be a family of instance equivalences $e_b : s_b \simeq t_b$ that satisfy the constraints of theorem 4.20.

Then $(e_b)_{b:B}$ lifts to a family $(E_F)_{F:B^A}$ of equivalences $E_F : S_F \simeq T_F$, where (S_F) and (T_F) are lifted from (s_b) and (t_b) according to proposition 4.27.

Proof. We recursively define E_F based on the six possibilities for e_b.

- If e_b is a constant $e : c \simeq c'$, set $E_F := \mathsf{K}_{AC}(e) : c^A \simeq c'^A$.
- If $e_b = \mathsf{id}_{t_b}$, set $E_F := \mathsf{id}_{T_F}$.
- If e_b is f_b^{-1} obtain E_F by recursion.
- Likewise for $e_b = g_b \circ f_b$.
- If $e_b = G_b(f_b)$, set $E_F := I_F(H_F)$, where H_F is obtained recursively from f_b, and I_F is lifted from G_b.
- If $e_b = \mathsf{def}_G(x_b, x'_b, \ldots)$ for a combinator G, set $E_F := \mathsf{def}_{G^A}(X_F, X'_F, \ldots)$, where X_F is lifted from x_b, etc. □

This gives rise to an alternative proof of theorem 4.20 as follows. Given a family $(e_a)_{a:A}$ of equivalences $e_a : s_a \simeq t_a$ as in theorem 4.20, lift (e_a) to a family $(E_H)_{H:A^A}$ of equivalences $E_H : S_H \simeq T_H$ in the functor universe $\mathcal{U}^A$. Since S_{I_A} and T_{I_A} are equivalent to the functors that the functoriality algorithm produces for (s_a) and (t_a), respectively, the result can be obtained from E_{I_A}.

Linear and affine functor operations

Now let $\mathcal{U}$ be any universe with at least linear functor operations. If $\mathcal{U}$ does not have full functor operations, then $\mathcal{U}^A$ does not have internal functors, but we can either apply a constant functor (i.e. in $\mathcal{U}$) to a variable argument (i.e. in $\mathcal{U}^A$) or vice versa.

Proposition 4.30. We have functors from $\mathcal{U}^A$ to $\mathcal{U}^A$ in $\mathcal{U}$, defined by $[B^A \to C^A] := [B \to C]$ for $B, C \in \mathcal{U}$. Moreover, we have functors from $\mathcal{U}$ to $\mathcal{U}^A$ in $\mathcal{U}^A$, defined by $[B \to C^A] := [B \to C]^A$.

Proof. For a functor $G : B^A \to C^A$, which by definition is just an instance of $[B \to C]$, and an instance $F : B^A$, we define their functor application by $G(F) := G \circ F$. For a functor $G : B \to C^A$, which is an instance of $[A \to B \to C]$, and an instance $b : B$, we define $G(b) := \mathsf{C}_{ABC}(G, b)$. □

We can unify these functors within a sum universe that additionally includes an empty type.

Definition 4.31. For a universes $\mathcal{U}, \mathcal{V}, \mathcal{W}$ such that we have functors from $\mathcal{U}$ to $\mathcal{V}$ in $\mathcal{W}$, and a type $A : \mathcal{U}$, we define the *optional functor universe* $\mathcal{V}^{A?}$ to be

$$\mathcal{V}^{A?} := \mathcal{V} \uplus \mathcal{V}^A \uplus \mathsf{Set}_{\{0\}}.$$

That is, each type in $\mathcal{V}^{A?}$ is either

- a type $B \in \mathcal{V}$,
- a functor type $B^A \in \mathcal{V}^A$, or
- the empty type 0.

Proposition 4.32. For a universe $\mathcal{U}$ with internal functors and at least linear functor operations, $\mathcal{U}^{A?}$ has internal functors as follows. For $B, C \in \mathcal{U}$, we set

- $[B \to C]$ in $\mathcal{U}^{A?}$ to be the same as $[B \to C]$ in $\mathcal{U}$,
- $[B \to C^A] := [B \to C]^A$,
- $[B^A \to C^A] := \begin{cases} [B \to C]^A & \text{if } \mathcal{U} \text{ has full functor operations} \\ [B \to C] & \text{otherwise,} \end{cases}$

- $[X \to Y] := 0$ for all $X, Y \in \mathcal{V}^{A?}$ not covered above.

Functor application is defined as in propositions 4.24 and 4.30.

Definition 4.33. For $B, C \in \mathcal{U}$ and $G : B \to C$, we define $G^{A?} : B^A \to C^A$ by

$$G^{A?} := \begin{cases} G^A & \text{if } \mathcal{U} \text{ has full functor operations} \\ G & \text{otherwise.} \end{cases}$$

Proposition 4.34. $\mathsf{embedMap}$ in $\mathcal{U}^A$ generalizes to

$$\mathsf{embedMap}^?(G, b) : G^{A?}(b^A) \simeq (G(b))^A.$$

Conjecture 4.35. If $\mathcal{U}$ has linear/affine/full functor operations, then so does $\mathcal{U}^{A?}$.

Proposition 4.36. For every $B \in \mathcal{U}$ and $F : A \to B$ we have an equivalence

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

4.5 Functorial meta-relations

5 Singletons

6 Products

7 Equivalences

8 Properties and relations

9 Dependent functors

10 Dependent products

References

[1] Mario Carneiro. The type theory of Lean. https://github.com/digama0/lean-type-theory/releases/download/v1.0/main.pdf, 2019. Accessed: 2021-21-15.

[2] J. Roger Hindley and Jonathan P. Seldin. *Lambda-Calculus and Combinators, an Introduction.* Cambridge University Press, 2008.

[3] Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction – CADE 28*, pages 625–635, Cham, 2021. Springer International Publishing.

[4] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics.* https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

[5] Wikipedia. Combinatory logic. https://en.wikipedia.org/wiki/Combinatory_logic. Accessed: 2021-21-15.

[6] Wikipedia. Curry–Howard correspondence. https://en.wikipedia.org/wiki/Curry%E2%80%93Howard_correspondence. Accessed: 2021-21-15.

[7] Wikipedia. Set-theoretic definition of natural numbers. https://en.wikipedia.org/wiki/Set-theoretic_definition_of_natural_numbers. Accessed: 2021-21-15.

[8] Wikipedia. Simply typed lambda calculus. https://en.wikipedia.org/wiki/Simply_typed_lambda_calculus. Accessed: 2021-21-15.

[9] Wikipedia. Universe in type theory. https://en.wikipedia.org/wiki/Universe_(mathematics)#In_type_theory. Accessed: 2021-21-15.