Dependent Combinators

Sebastian Reichelt

May 7, 2022

1 Introduction

This project, which is still work in progress, is an attempt to define structures axiomatically in such a way that these axioms have models in all conceivable collections of higher categories, including bicategories and hopefully any of the different notions of ∞-categories (and also, of course, 1-categories and sets). Although somewhat inspired by Homotopy Type Theory (HoTT) [3], the entire theory is compatible with standard (and, to a large extend, weaker) foundations, including Zermelo-Fraenkel (ZF) set theory and the dependent type theory implemented by the Lean 4 [2] theorem prover. A primary objective is to automate abstract but mathematically trivial proofs in Lean, and this document is intended to serve as documentation of that formalization.

The guiding principle (which somewhat matches categorical thinking outside of the HoTT realm) is to never assume equality in any axiom, as that equality becomes a more general equivalence at the next higher level. In particular, this means that 1-categories are not the right starting point for a completely generic formalization. Instead, the axioms have a type-theoretic flavor, with categories being just one of many models.

We do not, however, assume any knowledge of type theory – except in the following overview, which is specifically written with readers coming from a type theory background in mind. The rest of this document, as well as the formalization in Lean 4, are self-contained except for the descriptions of category-theoretic models of the axioms.

2 Overview

This section aims to give just enough of an overview so that readers with some knowledge of type theory can get an idea what to expect and not to expect.

One particular thing *not* to expect is a definition of a specific type theory, at least not in the usual sense, even though this documentation uses type-theoretic notation throughout. Instead, all definitions should be read primarily as notation for axioms defined in set theory – though it does make sense to treat each model of these axioms as a model of a type theory. This axiomatic approach allows us to avoid any discussion (and Lean formalization) of syntax, which would just distract from the goal of automating proofs *outside* of the type theory.

The axioms are organized in "layers." Although a strong connection between layers and type-theoretical universe levels (or homotopy levels in HoTT) exists, there is one important difference: Each model tends to satisfy the axioms of *all* layers. If a model has too little internal structure for a particular layer, then it satisfies the axioms of that layer trivially instead of not at all.

For example, the axioms of the first layer can loosely be interpreted as defining *propositions*, but those same axioms are *also* part of all other layers. One can think of layer n as defining constraints governing each span of n successive universe levels.

- **Layer 1** starts from a very simple definition of a Tarski-style *universe*: an index set I (of *types*) together with a function that assigns a set of *instances* (each written as $a : A$) to each type $A \in I$. (Note that this is a self-contained set-theoretic definition and not tied to any particular type theory.) Moreover:

- For any two types A and B, we assume the existence of a type $[A \to B]$ such that given an instance $F : A \to B$ and an instance $a : A$ we can obtain an instance $F(a) : B$. We call F a *functor* due to its additional structure introduced in layers 2 and 3. (In layer 1, we might call it an 'implication' instead.)
- We axiomatically assert the existence of certain functors, which in this layer correspond to logical axioms, and in general are combinators [4]. Thus in order to build functors, we can use the well-known algorithm to convert arbitrary lambda terms into applications of combinators.
- We can optionally introduce further types such as *units*, *counits*, *products*, and *coproducts*. Functorial introduction and elimination rules enable their integration into the above algorithm.
- Lastly, for types A and B we introduce an *equivalence type* $[A \leftrightarrow B]$ and assert the existence of certain instances of this type, in particular corresponding to reflexivity, symmetry, and transitivity. We also assert that there is an instance of $[[A \leftrightarrow B] \to [A \to B]]$. (Importantly, equivalence is *not* simply defined to be the product of two functors, as that definition would not be reusable at higher layers.)

- A **layer 2** universe is layer 1 universe $\mathcal{U}$ along with another layer 1 universe $\mathcal{V}$, which can be thought of as the next-lower-level universe. We introduce equivalences between instances as well as a limited notion of dependent types.
  - For every type $A : \mathcal{U}$ and instances $a, b : A$, we demand the existence of a type $[a \simeq b]$ in $\mathcal{V}$ that satisfies the axioms of a (functorial $\mathcal{V}$-valued) equivalence relation.
  - For each of the concrete functors defined in layer 1, we introduce an axiom that specifies the behavior of the functor up to equivalence.
  - For two functors $F : A \to B$ and $G : A \to C$, we introduce a type $[F \overset{\simeq}{\rightsquigarrow} G] : \mathcal{V}$ with axioms that let us interpret it as

    $$\prod_{a,b:A} [F(a) \simeq F(b) \to G(a) \simeq G(b)].$$

    Similarly to the combinators we use to build concrete functors, we define *dependent combinators* to convert lambda terms into instances of this type. The combinators also imply

    $$\mathsf{congrArg}_F : \prod_{a,b:A} [a \simeq b \to F(a) \simeq F(b)].$$

  - Additional dependent combinators guarantee *extensionality* of functors with respect to equivalence. This allows us to extend the Π notation by simply defining

    $$\prod_{a:A} [F(a) \simeq G(a)]$$

    as notation for $[F \simeq G]$, as we can automatically translate any lambda term implementing this Π type to a proof of $F \simeq G$. (In particular, this automates some tedious category theory proofs.)
  - Similarly to Π types in $\mathcal{V}$, we introduce Σ types in $\mathcal{U}$, i.e. subtypes.
  - All types are required to respect instance equivalences in specific ways, so that the translation of lambda terms into dependent combinators also works for terms involving those types. In particular, the two functors of an equivalence are required to be inverse to each other up to instance equivalence.
  - We can axiomatically embed $\mathcal{V}$ in $\mathcal{U}$, obtaining a notion of *truncation*.
  - If $\mathcal{V}$ appears as a type in $\mathcal{U}$, so that we have types of the form $[A \to \mathcal{V}]$ and $[A \to A \to \mathcal{V}]$, we can define a more generic notion of dependent types that implies the above axioms. Moreover, we can demand that instance equivalences of $\mathcal{V}$ are the same as type equivalences. (This tends to fail if $\mathcal{V}$ is too large.)

- **Layer 3** is derived from layer 2 by strengthening the universe $\mathcal{V}$ to a layer 2 universe. This means that we have equivalences of equivalences in a third (layer 1) universe $\mathcal{W}$, suggesting an interpretation of the higher-level equivalences as isomorphisms.
  - Types and instance equivalences in layer 3 are required to form a (weak) groupoid, functors are required to be groupoid functors, and so on.
  - In particular, equivalences are required to be adjoint equivalences.
  - Optionally, we may add a concept of morphisms between types of $\mathcal{U}$, by stating axioms that ensure compatibility with instance equivalences when those are regarded as isomorphisms.
  - If $\mathcal{V}$ is a type in $\mathcal{U}$, we can demand that morphisms of $\mathcal{V}$ are the same as functors.

3 Layer 1

This section introduces the 'propositional' axioms that all layers are based on, without any prerequisites.

3.1 Universes

Definition 3.1. We define a *layer 1 universe* to be a pair $(I, (S_A)_{A \in I})$ of

- an index set (or more general collection) 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.

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

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

(So this definition allows us to use type-theoretic notation without actually defining a type theory with concrete syntax.)

There are obviously a lot of examples of universes as defined above. Some concrete universes that we will refer to throughout the rest of the document are:

- A universe Unit with a single type $*$ that has a single instance $\star : *$.
- A universe Bool with an empty type $\bot$ and a type $\top$ with a single instance $\star : \top$.[1]
- A universe Set of sets, such that $S_A := A$ for each set A.[2]
- Universes of algebraic data structures, such that I is the collection of groups, vector spaces over a given field, etc., and S_A is the carrier set of the structure A.
- A universe Cat of categories, such that S_A is the set of objects of the category A.
- Generalization from Cat to higher categories should be straightforward. (Ideally, we would like to *define* ∞-categories based on the universe axioms, but this is still work in progress.)

3.2 Functors

Definition 3.2. We say that a layer 1 universe $\mathcal{U} = (I, (S_A)_{A \in I})$ has *functors* if for all types $A, B : \mathcal{U}$ we have a *functor type* $[A \to B] : \mathcal{U}$ and a function $\mathsf{apply}_{AB} : S_{[A \to B]} \times S_A \to S_B$.

Given instances $F : A \to B$ and $a : A$, we write "$F(a)$" for "$\mathsf{apply}_{AB}(F, a)$." As usual, we let "$\to$" 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 write "$F(a, b)$" for "$F(a)(b)$." (In section 3.4, we will identify $A \to B \to C$ with $A \times B \to C$, where $A \times B$ is a product type.)

[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.

[2] It is possible to use a universe of setoids in place of the universe of sets. In the Lean formalization, this universe is the simplest nontrivial universe in a certain sense.

Most of the universes defined above have functor types:

- In Unit, there is obviously only one choice $[* \to *] := *$ and $\star(\star) := \star$.
- In Bool, functors can be defined as logical implication:

$$[A \to B] := \begin{cases} \bot & \text{if } A = \top \text{ and } B = \bot, \\ \top & \text{otherwise} \end{cases}$$

  and $\star(\star) := \star$.
- Functions in Set are functors.
- In Cat, we can let $[A \to B]$ be the functor category from A to B.
- Universes of some algebraic structures have functors, depending on whether morphisms are again instances of the same structure. For example, this is the case for commutative semigroups and for vector spaces.

Definition 3.3. For a set S and a universe $\mathcal{U} = (I, \dots)$ with functors, we define a *$\mathcal{U}$-valued prerelation* on S to be a function $(\prec) : S \times S \to I$, which we write in infix form. 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 $a, b \in S$ we have a functor $(-)^{-1} : a \prec b \to b \prec a$, and
- *transitive* if for every $a, b, c \in S$ we have a bifunctor $(- \circ -) : a \prec b \to b \prec c \to a \prec c$.

(See also [3], section 2.1.)

Note in particular that the functor arrow $(\to)$ is a $\mathcal{U}$-valued prerelation on I.

Definition 3.4. A layer 1 universe $\mathcal{U}$ with functors has *linear logic* if for all types $A, B, C : \mathcal{U}$ there are functors

- $\mathsf{I}_A : A \to A$,
- $\mathsf{T}_{AB} : A \to [A \to B] \to B$, and
- $\mathsf{B}'_{ABC} : [A \to B] \to [B \to C] \to [A \to C]$.

It has *affine logic* if there is additionally a functor

- $\mathsf{K}_{AB} : B \to [A \to B]$,

and *full logic* if there is additionally[3]

- $\mathsf{W}_{AB} : [A \to A \to B] \to [A \to B]$.

We generally omit type subscripts when the types are fully determined by arguments. Moreover, if $\mathcal{U}$ has (at least) linear logic, then the prerelation $(\to)$ is reflexive and transitive with

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

Proposition 3.5. From the functors defined above we can derive the following functors with given types.

- $\mathsf{C}_{ABC} := \mathsf{B}'_{B[[B\to C]\to C][A\to C]}(\mathsf{T}_{BC}) \circ \mathsf{B}'_{A[B\to C]C} : [A \to B \to C] \to [B \to A \to C]$,
- $\mathsf{B}_{ABC} := \mathsf{C}(\mathsf{B}'_{ABC}) : [B \to C] \to [A \to B] \to [A \to C]$, and, assuming full logic,
- $\mathsf{O}_{AB} := \mathsf{W}_{[A\to B]B} \circ \mathsf{B}'_{[A\to B]AB} : [[A \to B] \to A] \to [A \to B] \to B$,
- $\mathsf{S}'_{ABC} := \mathsf{B}_{[A\to B\to C][A\to A\to C][A\to C]}(\mathsf{W}_{AC}) \circ \mathsf{B}_{A[B\to C][A\to C]} \circ \mathsf{B}'_{ABC} :$
  $[A \to B] \to [A \to B \to C] \to [A \to C]$,
- $\mathsf{S}_{ABC} := \mathsf{C}(\mathsf{S}'_{ABC}) : [A \to B \to C] \to [A \to B] \to [A \to C]$.

[3] We ignore *relevant logic* because it would add more complexity to definition 4.6.

From the choice of names, it may be obvious that these functors can be interpreted as combinators [4] in a simply-typed lambda calculus [6]. Therefore, we can construct functors using the standard algorithm to convert lambda terms to combinator applications.

Instead of lambda notation, we will write such constructed functors as *functor descriptions*

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

and state the algorithm to construct a functor corresponding to a given description in a slightly informal way as follows.

Proposition 3.6. Let $\mathcal{U}$ be a universe with (at least) linear logic, $A_1, \ldots, A_n$ 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.[4]

Then we can construct 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 $\mathcal{U}$ has linear but not affine logic, each variable a_k must occur exactly once in $t_{a_1 \ldots a_n}$.
- If $\mathcal{U}$ has affine but not full logic, each variable a_k must occur at most once in $t_{a_1 \ldots a_n}$.

Proof. First, note that the case $n > 1$ can be reduced to two recursive applications of the same algorithm with lower n as follows. For constant $a_1 : A_1$, construct the functor

$$\begin{aligned} F_{a_1} : A_2 \to \cdots \to A_n \to B \\ (a_2, \ldots, a_n) \mapsto t_{a_1 \ldots a_n}. \end{aligned}$$

Then the result is given by

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

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-unique) case split on t_a according to table 1.

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 logic, all other cases of functor applications may be regarded as mere optimizations. □

Remarks. 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.

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.

[4] More formally, the rules generate a family $(T_B)_{B:\mathcal{U}}$ 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 $A : \mathcal{U}$.

Definition		Functor
$F : A \to B$ $a \mapsto b$	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 $G : B \to C$	$\mathsf{B}'(H, G)$ with $H : A \to B$ $a \mapsto b_a$
$F : [B \to C] \to C$ $G \mapsto G(b)$	for $b : B$	$\mathsf{T}_{BC}(b)$
$F : A \to C$ $a \mapsto G_a(b)$	for $b : B$ and $G_a : B \to C$	$\mathsf{C}(G, b)$ with $G : A \to [B \to C]$ $a \mapsto G_a$
$F : A \to B$ $a \mapsto G_a(a)$	for $G_a : A \to B$	$\mathsf{W}(G)$ with $G : A \to [A \to B]$ $a \mapsto G_a$
$F : [B \to C] \to C$ $G \mapsto G(b_G)$	for $b_G : B$	$\mathsf{O}(H)$ with $H : [B \to C] \to B$ $G \mapsto b_G$
$F : A \to C$ $a \mapsto G_a(b_a)$	for $b_a : B$ and $G_a : B \to C$	$\mathsf{S}'(H, G)$ with $H : A \to B$ and $G : A \to [B \to C]$ $a \mapsto b_a$ and $a \mapsto G_a$

Table 1: Conversion of functor description to combinator application

Proposition 3.7. The algorithm applied to each of the following functor descriptions outputs exactly the given combinator, when taking the topmost possible alternative in case of nondeterminism.

- $\mathsf{K}_{AB} : B \to A \to B$
  $(b, a) \mapsto b$
- $\mathsf{I}_A : A \to A$
  $a \mapsto a$
- $\mathsf{B}'_{ABC} : [A \to B] \to [B \to C] \to A \to C$
  $(F, G, a) \mapsto G(F(a))$
- $\mathsf{B}_{ABC} : [B \to C] \to [A \to B] \to A \to C$
  $(G, F, a) \mapsto G(F(a))$
- $\mathsf{T}_{AB} : A \to [A \to B] \to B$
  $(a, F) \mapsto F(a)$
- $\mathsf{C}_{ABC} : [A \to B \to C] \to B \to A \to C$
  $(F, b, a) \mapsto F(a, b)$
- $\mathsf{W}_{AB} : [A \to A \to B] \to A \to B$
  $(F, a) \mapsto F(a, a)$
- $\mathsf{O}_{AB} : [[A \to B] \to A] \to [A \to B] \to B$
  $(F, G) \mapsto G(F(G))$
- $\mathsf{S}'_{ABC} : [A \to B] \to [A \to B \to C] \to A \to C$
  $(F, G, a) \mapsto G(a, F(a))$

- $\mathsf{S}_{ABC} : [A \to B \to C] \to [A \to B] \to A \to C$
$$(G, F, a) \mapsto G(a, F(a))$$

Remark. Note, however, that this does *not* imply that functors map instances exactly as specified by the definition, as in layer 1 there are no axioms about specific instances. However, for the concrete universes listed earlier this in fact the case.

Proposition 3.8. Of the universes listed earlier,

- Unit, Bool, Set, and Cat have full logic, and
- the universes of commutative semigroups and of vector spaces have linear logic.

Proof. The required functors can be defined easily. A strategy that works for all universes is to algorithmically construct the combinators from smaller functors as shown in proposition 3.7. E.g. for T, the algorithm specifies that one first needs to construct a functor

$$T_a : [A \to B] \to B$$
$$F \mapsto F(a)$$

and then to construct

$$\mathsf{T}_{AB} : A \to [[A \to B] \to B]$$
$$a \mapsto T_a,$$

both of which are either trivial or simple depending on the universe.

Alternatively, a universe has full logic if its types form a Cartesian closed category, and linear logic if its types form a closed monoidal category. (Note that the converse may not necessarily hold, due to the lack of axioms on instances.) □

Remark. Specifically for Cat, this means that we can construct functors between categories by simply stating their functor description, without having to verify manually that this definition is functorial. Similarly for linear maps between vector spaces, though in that case we need to ensure that each variable is used exactly once.

3.3 Singletons

Definition 3.9. A layer 1 universe $\mathcal{U}$ with functors may have a designated *unit type* $\top : \mathcal{U}$ which must satisfy the following two axioms.

- It must have an instance $\star : \top$.
- For each type $A : \mathcal{U}$ there is a functor $\mathsf{elim}_A : A \to \top \to A$ (which matches $\mathsf{K}_{\top A}$ but should exist even in the absence of affine logic, encoding the idea that a unit instance contains no data).

Definition 3.10. Similarly, $\mathcal{U}$ may have an *counit type* $\bot : \mathcal{U}$, which must satisfy the following axiom.

- For each type $A : \mathcal{U}$ there is a functor $\mathsf{elim}_A : \bot \to A$.

Remark. Note that the above axioms do not imply that $\top$ has a single instance or that $\bot$ has no instances. In particular the type $* : \mathsf{Unit}$ is both a unit type and a counit type.

3.4 Products and coproducts

Definition 3.11. A layer 1 universe $\mathcal{U}$ with functors has *products* if for all $A, B : \mathcal{U}$ we have a *product type* $[A \times B] : \mathcal{U}$ with the following functors.

- $\mathsf{intro}_{AB} : A \to B \to A \times B$ (taking the symbol "$\times$" to bind more strongly than "$\to$")
- $\mathsf{elim}_{ABC} : [A \to B \to C] \to [A \times B \to C]$ for each $C : \mathcal{U}$

For $a : A$ and $b : B$ we define their pair as

$$(a, b) := \mathsf{intro}(a, b)$$

and extend the functor description notation such that

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

is defined as $F := \mathsf{elim}(F')$ with

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

Remark. This implies that projections from $[A \times B]$ to A or B are only available under the assumption of (at least) affine logic, except if the other side is $\top$. However, linear logic gives us, for example,

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

and

$$\begin{aligned} \mathsf{assoc}_{ABC} : [A \times B] \times C &\to A \times [B \times C] \\ ((a, b), c) &\mapsto (a, (b, c)). \end{aligned}$$

Definition 3.12. $\mathcal{U}$ has *coproducts* if for all $A, B : \mathcal{U}$ we have a *coproduct type* $[A \oplus B] : \mathcal{U}$ with the following functors.

- $\mathsf{lintro}_{AB} : A \to A \oplus B$
- $\mathsf{rintro}_{AB} : B \to A \oplus B$
- $\mathsf{elim}_{ABC} : [A \to C] \to [B \to C] \to [A \oplus B \to C]$ for each $C : \mathcal{U}$

3.5 Equivalences

Definition 3.13. A layer 1 universe $\mathcal{U}$ with functors has *equivalences* if for all $A, B : \mathcal{U}$ we have an *equivalence type* $[A \leftrightarrow B] : \mathcal{U}$ such that $(\leftrightarrow)$ is a reflexive symmetric transitive prerelation and for $A, B : \mathcal{U}$ we have a functor

- $\mathsf{elim}_{AB} : [A \leftrightarrow B] \to [A \to B]$.

Due to the lack of an introduction functor, these axioms do not enable the construction of concrete equivalences between two different types. This is intentional because the correct notion of equivalence depends on the universe $\mathcal{U}$, for example equivalences in Set are bijections, whereas equivalences in Cat are (adjoint) equivalences of categories.

Therefore, we assert the existence of concrete equivalences axiomatically. In all of these cases, the corresponding functors in both directions can already be proven to exist regardless of the equivalence. In other words, the axioms do not add any logical strength, but they establish a concept of equivalence that can be referenced at higher layers.

Definition 3.14. We say that $\mathcal{U}$ has *standard equivalences* if it has (at least) linear logic and instances of the following types exist, for all $A, B, C : \mathcal{U}$.

- $[A \to B \to C] \leftrightarrow [B \to A \to C]$
- $[\top \to A] \leftrightarrow A$ if $\mathcal{U}$ has a unit type
- $[A \to \bot] \leftrightarrow [A \leftrightarrow \bot]$ if $\mathcal{U}$ has a counit type
- $[A \times B \to C] \leftrightarrow [A \to B \to C]$ if $\mathcal{U}$ has products
- $A \times B \leftrightarrow B \times A$ if $\mathcal{U}$ has products
- $[A \times B] \times C \leftrightarrow A \times [B \times C]$ if $\mathcal{U}$ has products
- $[A \to B \times C] \leftrightarrow [A \to B] \times [A \to C]$ if $\mathcal{U}$ has products and full logic
- $\top \times A \leftrightarrow A$ if $\mathcal{U}$ has products and a unit type

- $\bot \times A \leftrightarrow \bot$ if $\mathcal{U}$ has products and a counit type
- $A \oplus B \leftrightarrow B \oplus A$ if $\mathcal{U}$ has coproducts
- $[A \oplus B] \oplus C \leftrightarrow A \oplus [B \oplus C]$ if $\mathcal{U}$ has coproducts
- $[A \oplus B \to C] \leftrightarrow [A \to C] \times [B \to C]$ if $\mathcal{U}$ has coproducts, products, and full logic
- $\bot \oplus A \leftrightarrow A$ if $\mathcal{U}$ has coproducts and a counit type
- $[A \leftrightarrow B] \leftrightarrow [B \leftrightarrow A]$
- functors of the form $[A \leftrightarrow B] \to [\Phi(A) \leftrightarrow \Phi(B)]$, where $\Phi : \mathcal{U} \to \mathcal{U}$ is a function that constructs types

3.6 Functor universe

In this section, we construct a universe $\mathcal{U}^A$ that can be understood as "$\mathcal{U}$ within a context containing an instance of $A : \mathcal{U}$." If we regard the types of $\mathcal{U}$ as propositions (via the Curry-Howard correspondence [5]), then in order to prove an implication $[A \to B] : \mathcal{U}$, we can instead "prove B under the assumption A" by constructing an instance of $B^A : \mathcal{U}^A$.

We therefore want $\mathcal{U}^A$ to inherit all structure of $\mathcal{U}$, and whether or not it does so is a good coherence test of the axioms, particularly at higher layers.

Definition 3.15. Let $\mathcal{U}$ be a universe with full logic, and $A : \mathcal{U}$ be a type. We define the *functor universe* $\mathcal{U}^A$ as follows.

- The types of $\mathcal{U}^A$, written as $B^A : \mathcal{U}^A$ for $B : \mathcal{U}$, are in 1:1 correspondence with the types of $\mathcal{U}$. (In fact, we can technically define the set of types of $\mathcal{U}^A$ to be exactly the set of types of $\mathcal{U}$, but for clarity we will assume the types to be distinct in this description.)
- The set of instances of $B^A : \mathcal{U}^A$ is the set of instances of $[A \to B] : \mathcal{U}$.

Then $\mathcal{U}^A$ has, among others, the following instances.

- $\mathsf{I}_A : A^A$ (which acts as the "instance of A in the context"), and
- for each $B : \mathcal{U}$ and $b : B$, an "embedded" instance $b^A := \mathsf{K}_{AB}(b) : B^A$.

Proposition 3.16. For $B, C : \mathcal{U}$, the type $[B \to C]^A : \mathcal{U}^A$ is a functor type from $B^A : \mathcal{U}^A$ to $C^A : \mathcal{U}^A$. Under this definition of functors, $\mathcal{U}^A$ has full logic.

Proof. For $G : [B \to C]^A$ and $F : B^A$ we can define $G(F) := \mathsf{S}'_{ABC}(F, G) : C^A$.

For a functor $H : B \to C$, its embedded instance $H^A : [B \to C]^A$ is a functor from B^A to C^A. Thus, we can define the five primitive combinators by embedding the corresponding combinators of $\mathcal{U}$: $\mathsf{I}_{B^A} := (\mathsf{I}_B)^A$, $\mathsf{T}_{B^A C^A} := (\mathsf{T}_{BC})^A$, ... □

Proposition 3.17. For types $B, C : U$, consider the functor description

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

If t_b follows the constraints of proposition 3.6. we can lift the family $(t_b)_{b:B}$ to a family $(T_F)_{F:B^A}$ of instances of C^A such that the functor description

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

also follows these constraints.

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. □

We can use this proposition to obtain an alternative proof of proposition 3.6 as follows (assuming full logic). Given a family $(t_a)_{a:A}$ of terms $t_a : B$ that satisfy the constraints of proposition 3.6, we may lift (t_a) to a family $(T_F)_{F:A^A}$ of terms $T_F : B^A$. Then $T_{\mathsf{I}_A} : B^A$ is a functor from A to B.

Proposition 3.18. If $\mathcal{U}$ has a unit type $\top : \mathcal{U}$, then $\top^A$ is a unit type of $\mathcal{U}^A$.

Proof. Obviously we have $\star^A : \top^A$. For $B : \mathcal{U}$ we can take $\mathsf{elim}_{B^A} := (\mathsf{elim}_B)^A$. □

Counits, products, coproducts, and equivalences embed analogously.

4 Layer 2

4.1 Universes

Definition 4.1. A *layer 2 universe* $(\mathcal{U}, \mathcal{V}, (\simeq_A)_{A:\mathcal{U}})$ is a triple of

- a layer 1 universe $\mathcal{U}$,
- a layer 1 universe $\mathcal{V}$ with (at least) linear logic, and
- for each type $A : \mathcal{U}$, a reflexive symmetric transitive $\mathcal{V}$-valued prerelation $(\simeq_A)$ on the set of instances of A. We call $[a \simeq b] : \mathcal{V}$ the type of *instance equivalences* between $a : A$ and $b : A$.

We will often write "$\mathcal{U}$" instead of "$(\mathcal{U}, \mathcal{V}, (\simeq_A)_{A:\mathcal{U}})$," leaving $\mathcal{V}$ and $(\simeq_A)_{A:\mathcal{U}}$ implicit.

In particular, we will consider the following layer 2 universes.

- $(\mathsf{Unit}, \mathsf{Unit}, (\star, \star) \mapsto *)$
- $(\mathsf{Bool}, \mathsf{Unit}, (\star, \star) \mapsto *)$
- $(\mathsf{Set}, \mathsf{Bool}, (=))$
- $(\mathsf{Cat}, \mathsf{Set}, \mathrm{Iso})$, where $\mathrm{Iso}(a, b)$ is the set of isomorphisms from a to b. (This requires special consideration of size-related issues, which we will ignore here because they are adequately addressed by the formalization in Lean.)

4.2 Functors

Definition 4.2. A layer 2 universe $(\mathcal{U}, \mathcal{V}, (\simeq))$ has *functors* if $\mathcal{U}$ has functors (as a layer 1 universe) and additionally for all types $A, B, C : \mathcal{U}$ and functors $F : A \to B$ and $G : A \to C$ we have a type $[F \overset{\simeq}{\rightsquigarrow} G] : \mathcal{V}$ that satisfies the following axioms.

- For $a, b : A$ there is a functor
  $\mathsf{elim}_{FGab} : [F \overset{\simeq}{\rightsquigarrow} G] \to [F(a) \simeq F(b) \to G(a) \simeq G(b)]$.
- For $A, B : \mathcal{U}$ and $F, G : A \to B$ there is a functor
  $\mathsf{eq}_{FG} : F \simeq G \to [F \overset{\simeq}{\rightsquigarrow} G]$.
- For $A, B, C, D : \mathcal{U}$, $F : A \to B$, $G : A \to C$, and $H : A \to D$ there is a bifunctor
  $\mathsf{trans}_{FGH} : [F \overset{\simeq}{\rightsquigarrow} G] \to [G \overset{\simeq}{\rightsquigarrow} H] \to [F \overset{\simeq}{\rightsquigarrow} H]$.
- If $\mathcal{U}$ has (at least) linear logic, then for $A, B, C : \mathcal{U}$, $F : A \to B$, and $G : B \to C$ there is an instance
  $\mathsf{rBr}_{FG} : [F \overset{\simeq}{\rightsquigarrow} G \circ F]$.
- If $\mathcal{U}$ has (at least) linear logic, then for $A, B, C, D : \mathcal{U}$, $F : A \to B$, $G : B \to C$, and $H : A \to D$ there is a functor
  $\mathsf{rBl}_{FGH} : [G \circ F \overset{\simeq}{\rightsquigarrow} H] \to [F \overset{\simeq}{\rightsquigarrow} H]$.
- If $\mathcal{U}$ has (at least) affine logic, then for $A, B, C : \mathcal{U}$, $F : A \to B$, and $c : C$ there is an instance
  $\mathsf{rK}_{Fc} : F \overset{\simeq}{\rightsquigarrow} \mathsf{K}_{AC}(c)$.
- If $\mathcal{U}$ has full logic, then for $A, B, C, D : \mathcal{U}$, $F : A \to B$, and $G : A \to B \to C$ there is a functor
  $\mathsf{rSr}_{FG} : [F \overset{\simeq}{\rightsquigarrow} G] \to [F \overset{\simeq}{\rightsquigarrow} \mathsf{S}'(F, G)]$.

- If $\mathcal{U}$ has full logic, then for $A, B, C, D : \mathcal{U}$, $F : A \to B$, $G : A \to B \to C$, and $H : A \to D$ there is a functor
  $\mathsf{rSl}_{FGH} : [\mathsf{S}'(F, G) \overset{\simeq}{\rightsquigarrow} H] \to [G \overset{\simeq}{\rightsquigarrow} H] \to [F \overset{\simeq}{\rightsquigarrow} H]$.

In the concrete universes we consider, this type is implemented as follows.

- In Unit and Bool, obviously $[F \overset{\simeq}{\rightsquigarrow} G] := * : \mathsf{Unit}$ for all functors F and G.
- For $A, B, C : \mathsf{Set}$, $F : A \to B$, and $G : A \to C$, we define

$$[F \overset{\simeq}{\rightsquigarrow} G] := \begin{cases} \top & \text{if for all } a, b \in A, F(a) = F(b) \text{ implies } G(a) = G(b), \\ \bot & \text{otherwise.} \end{cases}$$

- For $A, B, C : \mathsf{Cat}$, $F : A \to B$, and $G : A \to C$, we define $[F \overset{\simeq}{\rightsquigarrow} G]$ to be a set of *functorial maps*, which are collections of functions from $\mathrm{Hom}(F(a), F(b))$ to $\mathrm{Hom}(G(a), G(b))$, depending on two objects $a, b : A$, that respect identity and composition of morphisms.

Definition 4.3. In a layer 2 universe $\mathcal{U}$ with functors, when we say that a functor F matches a given functor description

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

we require that there is an instance equivalence $\mathsf{def}_F(a_1, \ldots, a_n) : F(a_1, \ldots, a_n) \simeq t_{a_1 \ldots a_n}$ for $a_k : A_k$.

For $\mathcal{U}$ to have linear, affine, or full logic, we will require each of the five primitive combinators to match their description as given in proposition 3.7.

Remark. By rBr, this implies that that if $\mathcal{U}$ has (at least) linear logic, then

- for all $A, B : \mathcal{U}$, $F : A \to B$, and $a, b : A$ there is an instance
  $\mathsf{congrArg}_{Fab} : a \simeq b \to F(a) \simeq F(b)$, and
- for all $A, B : \mathcal{U}$, $F, G : A \to B$, and $a : A$ there is an instance
  $\mathsf{congrFun}_{FGa} : F \simeq G \to F(a) \simeq G(a)$.

Proposition 4.4. If the five primitive combinators match their functor descriptions as defined above, then the algorithm to construct a functor for a given functor description, as described in the proof of proposition 3.6, always produces functors that match the functor description that is given as input to the algorithm.

Proof. The required equivalence can be constructed by composing, in each step of the algorithm, the definition of the combinator used in that step with the definitions of its arguments, using $\mathsf{congrArg}$. □

4.3 Dependent functors

Definition 4.5. For a layer 2 universe layer 2 universe $(\mathcal{U}, \mathcal{V}, (\simeq))$ with functors, we introduce a *dependent functor type* notation

$$\left[\prod_{a_k : A_k} p_{a_1 \ldots a_n} \right] : \mathcal{V}$$

for some specific terms $p_{a_1 \ldots a_n} : \mathcal{V}$ with $A_1, \ldots, A_n : \mathcal{U}$. (A more generic definition that only works for some universes is given later, starting in section 4.10.)

The common requirement on this notation is that given a term

$$f : \prod_{a_k : A_k} p_{a_1 \ldots a_n}$$

and concrete $a_k : A_k$, we have a *dependent functor application*

$$f(a_1, \ldots, a_n) : p_{a_1 \ldots a_n}.$$

- For the case that $p_{a_1 \ldots a_n} : \mathcal{V}$ is an instance equivalence, we simply define

$$\left[\prod_{a_k : A_k} [s_{a_1 \ldots a_n} \simeq t_{a_1 \ldots a_n}]\right] := [F \simeq G]$$

with $B : \mathcal{U}$ such that $s_{a_1 \ldots a_n}, t_{a_1 \ldots a_n} : B$ and

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

To avoid nondeterminism within the specification of the dependent functor type, the functors F and G must be obtained from $(s_{a_1 \ldots a_n})$ and $(t_{a_1 \ldots a_n})$ by always taking the topmost possible alternative within table 1.

Note that the algorithm to construct F and G ensures that

$$\left[\prod_{a_1 : A_1, \ldots, a_n : A_n} [s_{a_1 \ldots a_n} \simeq t_{a_1 \ldots a_n}]\right] \stackrel{\text{def}}{=} \left[\prod_{a_1 : A_1, \ldots, a_{n-1} : A_{n-1}} \left[\prod_{a_n : A_n} [s_{a_1 \ldots a_n} \simeq t_{a_1 \ldots a_n}]\right]\right]$$

if $n > 1$.

- A specific case of implication is given by

$$\left[\prod_{a,b : A} [s_a \simeq s_b \to t_a \simeq t_b]\right] := [F \stackrel{\simeq}{\rightsquigarrow} G]$$

with $B, C : \mathcal{U}$ such that $s_a : B$ and $t_a : C$ and

$$\begin{aligned} F : A &\to B \\ a &\mapsto s_a, \\ G : A &\to C \\ a &\mapsto t_a. \end{aligned}$$

- Quantification over products can be defined by distribution:

$$\left[\prod_{a_k : A_k} [p_{a_1 \ldots a_n} \times q_{a_1 \ldots a_n}]\right] := \left[\left[\prod_{a_k : A_k} p_{a_1 \ldots a_n}\right] \times \left[\prod_{a_k : A_k} q_{a_1 \ldots a_n}\right]\right]$$

- Quantification over a unit type is defined to yield a unit type:

$$\left[\prod_{a_k : A_k} \top\right] := \top$$

As for functors, we wish to define instances $f : \prod_{a_k : A_k} p_{a_1 \ldots a_n}$ via *dependent functor descriptions*

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

so we will list specific conditions that enable this.

Definition 4.6. A layer 2 universe $(\mathcal{U}, \mathcal{V}, (\simeq))$ with functors has *linear logic* if the layer 1 universe $\mathcal{U}$ has linear logic, the definitions of I, T, and B' match their descriptions as given in proposition 3.7, and for all types $A, B, C, D : \mathcal{U}$ there are instances

- $\mathsf{rightId}_{AB} : \prod_{F:A\to B,a:A} [F(\mathsf{I}(a)) \simeq F(a)]$
$\stackrel{\text{def}}{=} \prod_{F:A\to B} [F \circ \mathsf{I}_A \simeq F]$
$\stackrel{\text{def}}{=} [\mathsf{B}'_{AAB}(\mathsf{I}_A) \simeq \mathsf{I}_{A\to B}]$,
- $\mathsf{leftId}_{AB} : \prod_{F:A\to B,a:A} [\mathsf{I}(F(a)) \simeq F(a)]$
$\stackrel{\text{def}}{=} \prod_{F:A\to B} [\mathsf{I}_B \circ F \simeq F]$
$\stackrel{\text{def}}{=} [\mathsf{B}_{ABB}(\mathsf{I}_B) \simeq \mathsf{I}_{A\to B}]$,
- $\mathsf{swapT}_{AB} : \prod_{F:A\to B,a:A} [\mathsf{T}(a,F) \simeq F(a)]$
$\stackrel{\text{def}}{=} \prod_{F:A\to B} [\mathsf{C}(\mathsf{T}_{AB},F) \simeq F]$
$\stackrel{\text{def}}{=} [\mathsf{C}(\mathsf{T}_{AB}) \simeq \mathsf{I}_{A\to B}]$,
- $\mathsf{swapB}'_{ABC} : \prod_{F:A\to B,a:A,G:B\to C} [(G\circ F)(a) \simeq G(F(a))]$
$\stackrel{\text{def}}{=} \prod_{F:A\to B} [\mathsf{C}(\mathsf{B}'_{ABC}(F)) \simeq \mathsf{T}_{BC} \circ F]$
$\stackrel{\text{def}}{=} [\mathsf{C}_{[B\to C]AC} \circ \mathsf{B}'_{ABC} \simeq \mathsf{B}_{AB[[B\to C]\to C]}(\mathsf{T}_{BC})]$,
- $\mathsf{swapB}_{ABC} : \prod_{G:B\to C,a:A,F:A\to B} [(G\circ F)(a) \simeq G(F(a))]$
$\stackrel{\text{def}}{=} \prod_{G:B\to C} [\mathsf{C}(\mathsf{B}_{ABC}(G)) \simeq \mathsf{B}_{[A\to B]BC}(G) \circ \mathsf{T}_{AB}]$
$\stackrel{\text{def}}{=} [\mathsf{C}_{[A\to B]AC} \circ \mathsf{B}_{ABC} \simeq \mathsf{B}_{A[[A\to B]\to B][[A\to B]\to C]}(\mathsf{T}_{AB}) \circ \mathsf{B}_{[A\to B]BC}]$,
- $\mathsf{assoc}_{ABCD} : \prod_{F:A\to B,G:B\to C,H:C\to D,a:A} [(H\circ G)(F(a)) \simeq H(G(F(a)))]$
$\stackrel{\text{def}}{=} \prod_{F:A\to B,G:B\to C,H:C\to D} [(H\circ G)\circ F \simeq H\circ(G\circ F)]$
$\stackrel{\text{def}}{=} [\mathsf{B}'_{[B\to C][[C\to D]\to[B\to D]][[C\to D]\to[A\to D]]}(\mathsf{B}'_{BCD}) \circ \mathsf{B}_{[C\to D][B\to D][A\to D]} \circ \mathsf{B}'_{ABD} \simeq$
$\mathsf{B}_{[B\to C][A\to C][[C\to D]\to[A\to D]]}(\mathsf{B}'_{ACD}) \circ \mathsf{B}'_{ABC}]$.

$(\mathcal{U},\mathcal{V},(\simeq))$ has *affine logic* it it has linear logic, the layer 1 universes $\mathcal{U}$ and $\mathcal{V}$ have affine logic, the definition of K matches its description as given in proposition 3.7, and for all types $A,B,C:\mathcal{U}$ there are instances

- $\mathsf{rightConst}_{ABC} : \prod_{b:B,G:B\to C,a:A} [G(\mathsf{K}(b,a)) \simeq G(b)]$
$\stackrel{\text{def}}{=} \prod_{b:B,G:B\to C} [G\circ \mathsf{K}_{AB}(b) \simeq \mathsf{K}_{AC}(G(b))]$
$\stackrel{\text{def}}{=} [\mathsf{B}'_{ABC} \circ \mathsf{K}_{AB} \simeq \mathsf{B}_{[B\to C]C[A\to C]}(\mathsf{K}_{AC}) \circ \mathsf{T}_{BC}]$,
- $\mathsf{leftConst}_{ABC} : \prod_{F:A\to B,c:C,a:A} [\mathsf{K}(c,F(a)) \simeq c]$
$\stackrel{\text{def}}{=} \prod_{F:A\to B,c:C} [\mathsf{K}_{BC}(c) \circ F \simeq \mathsf{K}_{AC}(c)]$
$\stackrel{\text{def}}{=} [\mathsf{B}'_{C[B\to C][A\to C]}(\mathsf{K}_{BC}) \circ \mathsf{B}'_{ABC} \simeq \mathsf{K}_{[A\to B][C\to A\to C]}(\mathsf{K}_{AC})]$.

$(\mathcal{U}, \mathcal{V}, (\simeq))$ has *full logic* if it has affine logic, the layer 1 universes $\mathcal{U}$ and $\mathcal{V}$ have full logic, the definition of W matches its description as given in proposition 3.7, and for all types $A, B, C : \mathcal{U}$ there are instances

- $\mathsf{dupC}_{AB} : \prod_{F:A\to A\to B, a:A} [\mathsf{C}(F,a,a) \simeq F(a,a)]$
  $\stackrel{\text{def}}{=} \prod_{F:A\to A\to B} [\mathsf{W}(\mathsf{C}(F)) \simeq \mathsf{W}(F)]$
  $\stackrel{\text{def}}{=} [\mathsf{W}_{AB} \circ \mathsf{C}_{AAB} \simeq \mathsf{W}_{AB}]$,
- $\mathsf{dupK}_{AB} : \prod_{F:A\to B, a:A} [\mathsf{K}(F,a,a) \simeq F(a)]$
  $\stackrel{\text{def}}{=} \prod_{F:A\to B} [\mathsf{W}(\mathsf{K}_{A[A\to B]}(F)) \simeq F]$
  $\stackrel{\text{def}}{=} [\mathsf{W}_{AB} \circ \mathsf{K}_{A[A\to B]} \simeq \mathsf{I}_{A\to B}]$.
- $\mathsf{rightDup}_{ABC} : \prod_{F:A\to A\to B, G:B\to C, a:A} [G(\mathsf{W}(F,a)) \simeq G(F(a,a))]$
  $\stackrel{\text{def}}{=} \prod_{F:A\to A\to B, G:B\to C} [G \circ \mathsf{W}(F) \simeq \mathsf{W}(\mathsf{B}_{ABC}(G) \circ F)]$
  $\stackrel{\text{def}}{=} [\mathsf{B}'_{ABC} \circ \mathsf{W}_{AB} \simeq$
  $\mathsf{B}_{[B\to C][A\to A\to C][A\to C]}(\mathsf{W}_{AC}) \circ \mathsf{B}'_{[B\to C][[A\to B]\to[A\to C]][A\to A\to C]}(\mathsf{B}_{ABC}) \circ$
  $\mathsf{B}'_{A[A\to B][A\to C]}]$,
- $\mathsf{leftDup}_{ABC} : \prod_{F:A\to B, G:A\to B\to B\to C, a:A} [\mathsf{W}(G(a), F(a)) \simeq G(a, F(a), F(a))]$
  $\stackrel{\text{def}}{=} \prod_{F:A\to B, G:A\to B\to B\to C} [\mathsf{S}'(F, \mathsf{W}_{BC} \circ G) \simeq \mathsf{S}'(F, \mathsf{S}'(F, G))]$
  $\stackrel{\text{def}}{=} [\mathsf{B}'_{[A\to B\to B\to C][A\to B\to C][A\to C]}(\mathsf{B}_{A[B\to B\to C][B\to C]}(\mathsf{W}_{BC})) \circ \mathsf{S}'_{ABC} \simeq$
  $\mathsf{S}'(\mathsf{S}'_{ABC}, \mathsf{B}'_{[A\to B\to B\to C][A\to B\to C][A\to C]} \circ \mathsf{S}'_{AB[B\to C]})]$.

Proposition 4.7. The following instances can be derived from the axioms above, for $A, B, C, D : \mathcal{U}$, assuming linear or full logic as appropriate.

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

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

1. By swapT.
2. By dupC and swapSwap.
3. By leftDup.
4. By assoc and dupW.

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

Lemma 4.8. When a single step in the functoriality algorithm is nondeterministic according to table 1, the resulting functors are equivalent.

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

Table 2: Nondeterminism in the algorithm to construct functors

Proof. The specific cases of nondeterminism are listed in table 2. (Grayed-out alternatives are redundant because they can also be regarded as alternatives of other alternatives.)

Equivalences between the alternatives given on the right are easily derived from the axioms and proposition 4.7. □

Lemma 4.9. For each primitive combinator F with functor description

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

there is an instance of

$$\prod_{x:X} [F(a_{1x}, \ldots, a_{nx}) \simeq t_{a_{1x} \ldots a_{nx}}]$$

if the families $(a_{1x})_{x:X}, \ldots, (a_{nx})_{x:X}$ satisfy the conditions of proposition 3.6.

Specifically, we have the following dependent functors for types $X, A, B, C : \mathcal{U}$ and suitable families $(a_x), (b_x), (F_x), (G_x)$.

1. $\mathsf{extI}_{XA}((a_x)_{x:X}) : \prod_{x:X} [\mathsf{I}(a_x) \simeq a_x]$
   with $a_x : A$,
2. $\mathsf{extT}_{XAB}((a_x)_{x:X}, (F_x)_{x:X}) : \prod_{x:X} [\mathsf{T}(a_x, F_x) \simeq F_x(a_x)]$
   with $a_x : A$ and $F_x : A \to B$,
3. $\mathsf{extB}'_{XABC}((F_x)_{x:X}, (G_x)_{x:X}, (a_x)_{x:X}) : \prod_{x:X} [\mathsf{B}'(F_x, G_x, a_x) \simeq G_x(F_x(a_x))]$
   with $F_x : A \to B$, $G_x : B \to C$, and $a_x : A$,
4. $\mathsf{extK}_{XAB}((b_x)_{x:X}, (a_x)_{x:X}) : \prod_{x:X} [\mathsf{K}(b_x, a_x) \simeq b_x]$
   with $b_x : B$ and $a_x : A$,

5. $\mathsf{extW}_{XAB}((F_x)_{x:X}, (a_x)_{x:X}) : \prod_{x:X} [\mathsf{W}(F_x, a_x) \simeq F_x(a_x, a_x)]$
   with $F_x : A \to A \to B$ and $a_x : A$.

Proof. According to definition 4.5, for each of the five combinators we need to execute the algorithm to construct a functor on both sides of each equivalence, taking the first alternative whenever 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 x, or dependent on x.

Fortunately, lemma 4.8 allows us to combine these three cases whenever the requirement on linear vs. affine vs. full logic is the same.[5] In order to handle linear and affine logic, we need to provide an individual construction for each case where exactly one of the arguments depends on x, but we do not need to consider the case where an argument is exactly x. In situations where full logic is required, we can limit ourselves to the case that *all* arguments depend on x.

This leaves us with the following list, where dependent arguments are written as "(t_x)" and constant arguments are written as "(t)."
In each case, for a family $(t_x)_{x:X}$ of terms $t_x : T$, let $\varphi(t_x) : X \to T$ denote the functor constructed for $(t_x)_{x:X}$ under the assumption that t_x depends on x.

1. $\mathsf{extI}_{XA}((a_x)) := \mathsf{leftId}(\varphi(a_x))$.
2. $\mathsf{extT}_{XAB}((a), (F_x)) := \mathsf{id}_{\mathsf{C}(\varphi(F_x), a)}$.
   $\mathsf{extT}_{XAB}((a_x), (F)) : \mathsf{C}(\mathsf{T}_{AB} \circ \varphi(a_x), F) \simeq F \circ \varphi(a_x)$ is constructed from swapT.
   $\mathsf{extT}_{XAB}((a_x), (F_x)) : \mathsf{S}'(\varphi(F_x), \mathsf{T}_{AB} \circ \varphi(a_x)) \simeq \mathsf{S}'(\varphi(a_x), \varphi(F_x))$ is constructed from $\mathsf{dupLeftC}$.
3. $\mathsf{extB}'_{XABC}((F), (G), (a_x)) := \mathsf{assoc}(\varphi(a_x), F, G)$.
   $\mathsf{extB}'_{XABC}((F), (G_x), (a)) : \mathsf{C}(\mathsf{B}'_{ABC}(F) \circ \varphi(G_x), a) \simeq \mathsf{C}(\varphi(G_x), F(a))$ is constructed from swapB'.
   $\mathsf{extB}'_{XABC}((F_x), (G), (a)) : \mathsf{C}(\mathsf{B}_{ABC}(G) \circ \varphi(F_x), a) \simeq G \circ \mathsf{C}(\varphi(F_x), a)$ is constructed from swapB.
   $\mathsf{extB}'_{XABC}((F_x), (G_x), (a_x)) := \mathsf{assocS}(\varphi(a_x), \varphi(F_x), \varphi(G_x))$.
4. $\mathsf{extK}_{XAB}((b), (a_x)) := \mathsf{leftConst}(\varphi(a_x), b)$.
   $\mathsf{extK}_{XAB}((b_x), (a)) : \mathsf{C}(\mathsf{K}_{AB} \circ \varphi(b_x), a) \simeq \varphi(b_x)$ is constructed from $\mathsf{rightConst}$.
   $\mathsf{extK}_{XAB}((b_x), (a_x)) : \mathsf{S}'(\varphi(a_x), \mathsf{K}_{AB} \circ \varphi(b_x)) \simeq \varphi(b_x)$ is constructed from $\mathsf{rightConst}$, $\mathsf{leftConst}$, dupK, and dupC.
5. extW requires full logic; therefore everything can be handled by
   $\mathsf{extW}_{XAB}((F_x), (a_x)) := \mathsf{leftDup}(\varphi(a_x), \varphi(F_x))$. □

Theorem 4.10. Let $\mathcal{U}$ be a layer 2 universe with functors, $A_1, \ldots, A_n, B$ be types in $\mathcal{U}$, $(s_{a_1 \ldots a_n})_{a_k:A_k}$ 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,
- $\mathsf{congrArg}_{G_{a_1 \ldots a_n}}(f_{a_1 \ldots a_n})$ for a functor $G_{a_1 \ldots a_n}$ that satisfies the constraint of proposition 3.6 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 a primitive combinator, such that $x_{a_1 \ldots a_n}, x'_{a_1 \ldots a_n}, \ldots$ satisfy the constraint of proposition 3.6.

Then we can construct an equivalence $e : F_s \simeq F_t$ between functors F_s and F_t, corresponding to the dependent functor description

$$e : A_1 \to \cdots \to A_n \to e_{a_1 \ldots a_n}$$
$$(a_1, \ldots, a_n) \mapsto s_{a_1 \ldots a_n} \simeq t_{a_1 \ldots a_n}.$$

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

Proof. As in the proof of proposition 3.6, we can restrict ourselves to the case that $n = 1$, i.e. we replace $a_1, \dots, a_n$ 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 $e := \mathsf{congrArg}_{\mathsf{K}_{AB}}(f)$.
- If $e_a = \mathsf{id}_{s_a}$ 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 $e_a = \mathsf{congrArg}_{G_a, u_a, v_a}(f_a)$ 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 4.8, 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 $e_a = \mathsf{def}_G(x_a, x'_a, \dots)$ for a combinator G, set e according to lemma 4.9. □

Corollary 4.11. If $F, G : A_1 \to \cdots \to A_n \to B$ are functors and $(e_{a_1 \dots a_n})_{a_k : A_k}$ is a family of instance equivalences $e_{a_1 \dots a_n} : F(a_1, \dots, a_n) \simeq G(a_1, \dots, a_n)$ that satisfy the constraints of theorem 4.10, then we have an equivalence $e : F \simeq G$.

Corollary 4.12. Whenever the algorithm to construct a functor is nondeterministic, there are equivalences between the resulting functors.
(This strengthens lemma 4.8, 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 4.10 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 [1], theorem 8.14. Our result is similar, the key differences being that on the one hand we also incorporate linear and affine logic, while on the other hand our theorem is restricted to simply-typed lambda calculus.

4.4 Categorical prerelations

Note that the axioms rightId, leftId, and assoc give every universe with (at least) linear logic the shape of a weak category, which we can define as follows.

Definition 4.13. For a layer 2 universe $\mathcal{U}$ with functors, a $\mathcal{U}$-valued reflexive and transitive prerelation $(\prec)$ on a set S is defined to be *categorical* if we have the following three instances for $a, b, c, d \in S$.

- $\mathsf{rightId}_{ab} : \prod_{f : a \prec b} [f \circ \mathsf{id}_a \simeq f]$
  $\stackrel{\text{def}}{=} [- \circ \mathsf{id}_a \simeq \mathsf{I}_{a \prec b}]$,
- $\mathsf{leftId}_{ab} : \prod_{f : a \prec b} [\mathsf{id}_b \circ f \simeq f]$
  $\stackrel{\text{def}}{=} [\mathsf{id}_b \circ - \simeq \mathsf{I}_{a \prec b}]$,

- $\mathsf{assoc}_{abcd} : \prod_{f:a\prec b, g:b\prec c, h:c\prec d} [(h \circ g) \circ f \simeq h \circ (g \circ f)]$
$\stackrel{\text{def}}{=} [\mathsf{B}'_{[b\prec c][[c\prec d]\to[b\prec d]][[c\prec d]\to[a\prec d]]}(- \circ -) \circ \mathsf{B}_{[c\prec d][b\prec d][a\prec d]} \circ (- \circ -) \simeq$
$\mathsf{B}_{[b\prec c][a\prec c][[c\prec d]\to[a\prec d]]}(- \circ -) \circ (- \circ -)]$.

In $U = \mathsf{Set}$, these axioms reduce to the axioms of a 1-category. For $\mathcal{U} = \mathsf{Cat}$, the three equivalences are precisely the two unitors and the associator in the definition of a bicategory; adding the triangle and pentagon identities will be part of layer 3.

4.5 Rewriting

4.6 Singletons

4.7 Products and coproducts

4.8 Equivalences

4.9 Functor universe

4.10 Universe type

5 Layer 3

References

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

[2] 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.

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

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

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

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