Higher Lenses

Paolo Capriotti
TU Darmstadt

Nils Anders Danielsson
University of Gothenburg

Andrea Vezzosi
IT University Copenhagen

***Abstract*—We show that total, very well-behaved lenses are not very well-behaved when treated proof-relevantly in the setting of homotopy type theory/univalent foundations. In their place we propose something more well-behaved: higher lenses. Such a lens contains an equivalence between the lens's source type and the product of its view type and a remainder type, plus a function from the remainder type to the propositional truncation of the view type. It can equivalently be formulated as a getter function and a proof that its family of fibres is coherently constant, i.e. factors through propositional truncation.**

We explore the properties of higher lenses. For instance, we prove that higher lenses are equivalent to traditional ones for types that satisfy the principle of uniqueness of identity proofs. We also prove that higher lenses are n-truncated for n-truncated types, using a coinductive characterisation of coherently constant functions.

I. Introduction

Lenses were originally proposed [1] as a modular approach to the *view update* problem [2], i.e., how to propagate changes made to a view back into the original structure. Modularity is achieved by combinators that build more complex lenses from simpler ones. The subset of *very well-behaved* lenses has since been adopted in functional programming as a way to simplify reading and writing components of larger structures, especially when deeply nested. Such lenses have been further extended to a larger family called *optics* [3], but we will focus on the original notion in this paper.

For a small example, consider a record type R_1 containing a field $\mathsf{f}_1 : R_2$, where the record type R_2 contains a field $\mathsf{f}_2 : R_3$, and the record type R_3 contains a boolean field f_3:

```
record R₁ : Type where        record R₃ : Type where
  field f₁ : R₂                 field f₃ : Bool

record R₂ : Type where
  field f₂ : R₃
```

Given a value $x : R_1$, how do you invert the boolean value contained within it? Manual code can easily become somewhat awkward:

```
record x { f₁ = record (x .R₁.f₁) { f₂ =
  record (x .R₁.f₁ .R₂.f₂) { f₃ =
    not (x .R₁.f₁ .R₂.f₂ .R₃.f₃) } } }
```

Consider also the additional complication if the field f_3 is a finite map from strings to *Bool*, and you want to update the boolean that the string `"foo"` maps to (if any).

With lenses these updates are easy. Assume that three lenses are given, $r_1 : \mathit{Lens}\ R_1\ R_2$, $r_2 : \mathit{Lens}\ R_2\ R_3$ and $r_3 : \mathit{Lens}\ R_3\ \mathit{Bool}$, corresponding to f_1, f_2 and f_3, respectively. Then one can perform the inversion of the boolean contained in the record value x by composing the three lenses: $\mathit{modify}\ (r_3 \circ r_2 \circ r_1)\ \mathit{not}\ x$.

Total, very well-behaved lenses consist of a getter and a setter satisfying three laws [1]. These laws are often treated proof-irrelevantly, but we are interested in the question of how lenses behave in homotopy type theory/univalent foundations [4], in which equality types can have a rich structure. For this reason we interpret "a getter and a setter satisfying three laws" in the following way, with the proofs as an explicit part of the definition ("T" stands for "traditional"):

```
record Lensᵀ (A : Type a) (B : Type b) :
             Type (a ⊔ b) where
  field get     : A → B
        set     : A → B → A                                (1)
        get-set : ∀ a b → get (set a b) ≡ b
        set-get : ∀ a → set a (get a) ≡ a
        set-set : ∀ a b₁ b₂ → set (set a b₁) b₂ ≡ set a b₂
```

Here $\mathit{Type}\ a$ is the type universe at level a in the universe hierarchy, and $a \sqcup b$ is the maximum of the levels a and b.

A lens (from now on we drop "total, very well-behaved") from A to B provides an updatable B-view of elements of type A: the getter returns the view, and given a (possibly new) value for the view one can update the value of type A using the setter. The *get-set* law states that the setter actually updates the view, the *set-get* law states that nothing happens if a value is updated with its own view, and the *set-set* law states that if one updates the view twice, then the first update has no effect.

We will see that in the setting of homotopy type theory this type is not entirely well-behaved (§III):

- The type $\mathit{Lens}^{\mathrm{T}}\ A\ \top$ of lenses to the unit type is not always equivalent to the unit type, for instance when A is the circle, $\mathbb{S}^1$.
- Lenses with equal setters have equal getters. However, lenses with equal setters can still be distinct, even if the view type is provably inhabited.
- Certain *coherence laws* do not in general hold for traditional lenses. Here is a simple example, defined using a canonical implementation of the "congruence" function $\mathit{cong} : (f : A \to B) \to x \equiv y \to f\ x \equiv f\ y$:

  $$\mathit{cong}\ \mathsf{get}\ (\mathsf{set\text{-}get}\ a) \equiv \mathsf{get\text{-}set}\ a\ (\mathsf{get}\ a)$$

  This is an equality between equality proofs. The two sides have the same type, so it makes sense to ask whether they are equal. The answer is not necessarily yes.

978-1-6654-4895-6/21/$31.00 ©2021 IEEE

These problems are irrelevant if one restricts attention to types that are *sets*, i.e. types A which satisfy the principle of uniqueness of identity proofs ($(x\ y : A) \to (p\ q : x \equiv y) \to p \equiv q$). Lenses are perhaps mainly used for "programs", rather than "proofs". Thus one may wonder if types that are not sets occur in programs. Let us give one small example. The following type represents terms, and is parametrised by the terms' types:

$$\begin{array}{l} \textbf{data}\ \mathit{Tm}\ (A : \mathit{Type}) : \mathit{Type}_1\ \textbf{where} \\ \quad \mathsf{literal} : A \to \mathit{Tm}\ A \\ \quad \mathsf{map}\ \ : \{B : \mathit{Type}\} \to \\ \qquad\qquad \mathit{Tm}\ (B \to A) \to \mathit{Tm}\ B \to \mathit{Tm}\ A \end{array} \tag{2}$$

(We use the notation $\{B : \mathit{Type}\}$ for *implicit* arguments, that do not need to be given explicitly if they can be inferred.) This kind of type can be used to represent a domain-specific embedded language, and one can prove that if A is inhabited, then $\mathit{Tm}\ A$ is not a set. One may object that the map constructor's type argument is unrestricted. However, we get the same result if we restrict it:

$$\begin{array}{l} \mathsf{map} : \{B : \mathit{Type}\} \to \mathit{Is\text{-}set}\ B \to \\ \qquad \mathit{Tm}\ (B \to A) \to \mathit{Tm}\ B \to \mathit{Tm}\ A \end{array} \tag{3}$$

As an alternative to traditional lenses we present *higher lenses*, which do not have the problems listed above:

- Higher lenses can be defined in several equivalent ways, one of which is small (§V, §XII and §XIII).
- Higher lenses satisfy coherence laws like the one above (§VI), and in fact infinite towers of coherence conditions, expressed coinductively (§XIII).
- The type of higher lenses from A to the unit type is equivalent to the unit type. Higher lenses also satisfy other equivalences (§VII).
- *Homotopy levels*, or h-levels, is a concept which is used to classify types (see Section II for a brief introduction). If there is a lens from A to B, where A is inhabited and has h-level n, then B also has h-level n (§IX). Furthermore the h-level of the type of higher lenses from A to B is n if A has positive h-level n (§XV).
- Traditional and higher lenses from sets are equivalent, with an equivalence that preserves getters and setters (§X), and they form equal categories (§XI). However, whereas certain "naive" categories of higher lenses between *types* are univalent, this is not the case for certain naive categories of traditional lenses (§XI).
- Higher lenses with equal setters are equal, given the assumption that the view type C of the resulting lens is *stable* in the sense that $\| C \|$ implies C (§VIII). Here $\| C \|$ is the *propositional truncation* of C, roughly speaking the quotient with the trivial relation [4]. Note that both inhabited types and empty types are stable.
- Composition is unique for stable view types, if the setter is required to be implemented in the canonical way (§XI). We present two implementations of composition: one works for lenses for which the universe of the source type is at least as large as the universe of the view type (§XI), and the other is unrestricted (§XIV). The first implementation is associative, with the identity lens as a left and right unit. For the second this holds when the view type of the resulting lens is stable.

The results have been formalised in Cubical Agda [5], and the code is available to download [6]. Everything with an "equation number" has a counterpart in the formalisation, and quite a few other things mentioned in the text have also been formalised. (Note that there are small differences between the code and this text.)

There are other "traditional" representations of very well-behaved lenses, notably the Van Laarhoven representation [7]. We have chosen to focus mostly on $\mathit{Lens}^{\mathrm{T}}$, but also briefly discuss a representation based on bijections [8], see Section IV. (Related work is discussed further in Section XVI.) We do not focus on questions of efficiency, but note that we have started looking into ensuring that "proofs" are not present at run-time by using explicit erasure annotations [9], [10].

We would like to point out that lenses of the kind discussed above do not work very well for dependent record types. One problem is that if one (say) modifies the set field of a traditional lens, then one might have to also modify the proofs of the lens laws. We prove that it is not in general possible to define a lens for the first projection from a Σ-type (§IX). Another problem is that the lens types discussed above have types like $\mathit{Type} \to \mathit{Type} \to \mathit{Type}$, not $(A : \mathit{Type}) \to (A \to \mathit{Type}) \to \mathit{Type}$. There is some preliminary work on *dependent lenses* [11], with type signatures of the latter kind (modulo size issues). We hope that the work we present here can inform further work on dependent lenses.

II. Homotopy Type Theory

This section contains a brief introduction to some concepts from homotopy type theory that are relevant for this work.

We use $\_\equiv\_$ to denote the equality (or identity) type, *refl* for the canonical proof of reflexivity of type $(x : A) \to x \equiv x$, and we use a canonical implementation of the "congruence" function *cong*, mentioned above, of type $(f : A \to B) \to x \equiv y \to f\ x \equiv f\ y$.

Our results have been formalised in Cubical Agda [5], in which equality of functions is extensional, and the univalence "axiom" can be proved. A consequence of the univalence axiom is that equivalent types (in the same universe) are equal. We use $A \simeq B$ to denote that A and B are equivalent, and define the concept in the following way [4]:

$$\begin{array}{l} \mathit{Is\text{-}equivalence} : \{A : \mathit{Type}\ a\}\ \{B : \mathit{Type}\ b\} \to \\ \qquad\qquad\qquad (A \to B) \to \mathit{Type}\ (a \sqcup b) \\ \mathit{Is\text{-}equivalence}\ \{A = A\}\ \{B = B\}\ f = \\ \quad (f^{-1} : B \to A)\ \times \\ \quad (f\text{-}f^{-1} : \forall\ x \to f\ (f^{-1}\ x) \equiv x)\ \times \\ \quad (f^{-1}\text{-}f : \forall\ x \to f^{-1}\ (f\ x) \equiv x)\ \times \\ \quad \forall\ x \to \mathit{cong}\ f\ (f^{-1}\text{-}f\ x) \equiv f\text{-}f^{-1}\ (f\ x) \end{array} \tag{4}$$

$$\begin{array}{l} \_\simeq\_ : \mathit{Type}\ a \to \mathit{Type}\ b \to \mathit{Type}\ (a \sqcup b) \\ A \simeq B = (f : A \to B) \times \mathit{Is\text{-}equivalence}\ f \end{array} \tag{5}$$

Here the syntax $\{A = X\}$ is used to bind the variable X to the implicit argument A, and $(x : A) \times P\ x$ is a Σ-type: a dependent pair type where the type of the second component can depend on the value of the first component. We use the notation $\mathit{to}\ \mathit{eq}$ for the left-to-right direction of the equivalence eq ("f"), and $\mathit{from}\ \mathit{eq}$ for the other direction ("f^{-1}"). The proof showing that $\mathit{to}\ \mathit{eq}$ is a left inverse of $\mathit{from}\ \mathit{eq}$ ("f-f^{-1}") is denoted by $\mathit{to\text{-}from}\ \mathit{eq}$, and the right inverse proof ("f^{-1}-f") by $\mathit{from\text{-}to}\ \mathit{eq}$.

Note the last component of *Is-equivalence*:

$$\begin{aligned}\forall\ x \to\ & \mathit{cong}\ (\mathit{to}\ \mathit{eq})\ (\mathit{from\text{-}to}\ \mathit{eq}\ x) \equiv \\ & \mathit{to\text{-}from}\ \mathit{eq}\ (\mathit{to}\ \mathit{eq}\ x)\end{aligned} \tag{6}$$

This coherence property, which relates two proofs of the same equality type, is not present in usual definitions of what it means for a function to be bijective. If there is a bijection (or a "function with a quasi-inverse") between two types, then one can prove that the types are equivalent [4]. However, the type of bijections between two types is not in general equivalent to the type of equivalences between those types [4]. Equivalences are better behaved, because $\mathit{Is\text{-}equivalence}\ f$ is necessarily a *proposition*, meaning that all inhabitants are equal [4].

The concept of a "proposition" is part of the hierarchy of *homotopy levels*, or h-levels, which can be used to classify types. We use the following definition of h-levels (note that the term *n-type* used in the HoTT book [4] corresponds to types with h-level $2 + n$):

$$\begin{aligned}&\mathit{H\text{-}level} : \mathbb{N} \to \mathit{Type}\ a \to \mathit{Type}\ a \\ &\mathit{H\text{-}level}\ \mathsf{zero} \qquad\qquad A = \mathit{Contractible}\ A \\ &\mathit{H\text{-}level}\ (\mathsf{suc}\ \mathsf{zero}) \quad\ \ A = (x\ y : A) \to x \equiv y \\ &\mathit{H\text{-}level}\ (\mathsf{suc}\ (\mathsf{suc}\ n))\ A = \\ &\quad (x\ y : A) \to \mathit{H\text{-}level}\ (\mathsf{suc}\ n)\ (x \equiv y)\end{aligned} \tag{7}$$

A type has h-level 0 if it is *contractible*, which is equivalent to stating that the type is equivalent to the unit type:

$$\begin{aligned}&\mathit{Contractible} : \mathit{Type}\ a \to \mathit{Type}\ a \\ &\mathit{Contractible}\ A = (x : A) \times ((y : A) \to x \equiv y)\end{aligned} \tag{8}$$

Types with h-level 1 are called propositions ($\mathit{Is\text{-}proposition} = \mathit{H\text{-}level}\ 1$). A type has h-level 2 if all of its equality types are propositions. In this case we call the type a *set* ($\mathit{Is\text{-}set} = \mathit{H\text{-}level}\ 2$). Every proposition is a set, because if a type has h-level n, then it also has all higher h-levels [4].

Cubical Agda comes with support for higher inductive types (HITs). We use a HIT called the circle [4]:

$$\begin{aligned}&\textbf{data}\ \mathbb{S}^1 : \mathit{Type}\ \textbf{where} \\ &\quad \mathsf{base} : \mathbb{S}^1 \\ &\quad \mathsf{loop}\ \ : \mathsf{base} \equiv \mathsf{base}\end{aligned} \tag{9}$$

HITs allow you to give not just ordinary ("point") constructors, but also *higher* constructors that can affect the meaning of equality: to a first approximation one can think of this as a means to define quotient types. The circle is a type with a point constructor base and a higher constructor loop that states that base is equal to itself. Using loop one can construct a function that is distinct from $\mathit{refl} : (x : \mathbb{S}^1) \to x \equiv x$:

$$\mathit{not\text{-}refl} : (x : \mathbb{S}^1) \to x \equiv x \tag{10}$$

In fact, the type $\mathsf{base} \equiv \mathsf{base}$ is equivalent to the set of integers $\mathbb{Z}$ [4], [12]. Thus the circle is not a set.

We also use some truncation operators. The *propositional* truncation operator $\|\_\|$ turns any type A into a proposition which is inhabited if A is and which is not inhabited if A is not inhabited. The *one-step* truncation [13] is discussed in Section XIII. Both can be defined as HITs.

III. Traditional Lenses

Let us now investigate traditional lenses in a higher setting. Using the first two lens laws one can prove that lenses with equal setters have equal getters. However, in the proof-relevant setting of this text there are lenses with equal setters that are not equal. We use the circle (9) to construct the counterexample:

$$\begin{aligned}&l : \mathit{Lens}^{\mathrm{T}}\ \mathbb{S}^1\ \mathbb{S}^1 \\ &l = \textbf{record}\ \{ \\ &\quad \mathsf{get} = \lambda\ x \to x;\ \mathsf{set} = \lambda\ \_\ x \to x; \\ &\quad \mathsf{get\text{-}set} = \lambda\ \_ \to \mathit{not\text{-}refl};\ \mathsf{set\text{-}get} = \mathit{refl}; \\ &\quad \mathsf{set\text{-}set} = \lambda\ \_\ \_ \to \mathit{not\text{-}refl}\ \}\end{aligned} \tag{11}$$

If *not-refl* (10) is replaced by *refl* in the definition of l, then we get the identity lens for $\mathbb{S}^1$. We can prove that the lens l fails to satisfy the coherence law given in the introduction:

$$\begin{aligned}\neg\ ((x : \mathbb{S}^1) \to\ & \mathit{cong}\ (\mathsf{get}\ l)\ (\mathsf{set\text{-}get}\ l\ x) \equiv \\ & \mathsf{get\text{-}set}\ l\ x\ (\mathsf{get}\ l\ x))\end{aligned} \tag{12}$$

We have the following sequence of implications (where $\bot : \mathit{Type}$ is an empty type; note that $\neg\ A$ is defined to be $A \to \bot$):

$$\begin{aligned}&((x : \mathbb{S}^1) \to \mathit{cong}\ (\mathsf{get}\ l)\ (\mathsf{set\text{-}get}\ l\ x) \equiv \\ &\qquad\qquad \mathsf{get\text{-}set}\ l\ x\ (\mathsf{get}\ l\ x)) \qquad \to \\ &((x : \mathbb{S}^1) \to \mathit{refl}\ x \equiv \mathit{not\text{-}refl}\ x) \qquad\ \to \\ &\bot\end{aligned}$$

The first step uses the fact that $\mathit{cong}\ f\ (\mathit{refl}\ x)$ is equal to $\mathit{refl}\ (f\ x)$. Because the identity lens satisfies this coherence law we get that the two lenses are not equal.

Using similar steps one can also prove that the lens l fails to satisfy another coherence law:

$$\begin{aligned}\neg\ ((x\ y\ z : \mathbb{S}^1) \to\ & \mathit{cong}\ (\mathsf{get}\ l)\ (\mathsf{set\text{-}set}\ l\ x\ y\ z) \equiv \\ & \mathit{trans}\ (\mathsf{get\text{-}set}\ l\ (\mathsf{set}\ l\ x\ y)\ z) \\ & \quad (\mathit{sym}\ (\mathsf{get\text{-}set}\ l\ x\ z)))\end{aligned} \tag{13}$$

Here *sym* and *trans* are the canonical proofs showing that equality is symmetric and transitive, respectively.

We have proved that if the view type is a proposition, then $\mathit{Lens}^{\mathrm{T}}\ A\ B$ is equivalent to the Cartesian product of the type of the getter and the type of the reflexivity proof for A:

$$\begin{aligned}&\mathit{Is\text{-}proposition}\ B \to \\ &\mathit{Lens}^{\mathrm{T}}\ A\ B \simeq (A \to B) \times ((a : A) \to a \equiv a)\end{aligned} \tag{14}$$

The proof is omitted due to lack of space. As noted above (§II) the type $(a : A) \to a \equiv a$ is not a proposition when A is $\mathbb{S}^1$. Thus, if we let A be the circle and B be the unit type, then we get a result mentioned in the introduction: $\mathit{Lens}^{\mathrm{T}}\ A\ \top$ is not necessarily equivalent to the unit type.

We end this section by discussing two equivalences between a lens's source type and the Cartesian product of some type based on the lens and the lens's view type. These equivalences are stated using the following proof-relevant variant of the notion of a preimage of a function:

$$\begin{aligned}&\_^{-1}\_ : \{A : \mathit{Type}\ a\}\ \{B : \mathit{Type}\ b\} \to \\ &\qquad (A \to B) \to B \to \mathit{Type}\ (a \sqcup b) \\ &f^{\ -1}\ y = \exists\ x \times f\ x \equiv y\end{aligned} \tag{15}$$

(We use the notation $\exists\ x \times P\ x$ for Σ-types when we do not want to give the type of the domain.) In the setting of homotopy type theory $f^{\ -1}\ y$ can be called the *fibre* of f over y [4].

The first equivalence is based on an isomorphism given by Johnson et al. [14, Corollary 13]:

$$\begin{aligned}&\mathit{Is\text{-}set}\ B \to (l : \mathit{Lens}^{\mathrm{T}}\ A\ B)\ (b : B) \to \\ &A \simeq (\mathsf{get}\ l^{\ -1}\ b \times B)\end{aligned} \tag{16}$$

Note that we require the view type to be a set. For higher lenses we can prove a corresponding equivalence without making this requirement (see Lemma 37). We can also do this for traditional lenses that satisfy the two coherence laws that were discussed above (given in negated form for a particular lens in Lemmas 12 and 13).

The second equivalence is based on an isomorphism given by Pierce and Schmitt [15, Theorem 2.3.9]:

$$\begin{aligned}&\mathit{Is\text{-}set}\ A \to (l : \mathit{Lens}^{\mathrm{T}}\ A\ B) \to \\ &A \simeq ((f : B \to A) \times \parallel \mathsf{set}\ l^{\ -1}\ f \parallel) \times B\end{aligned} \tag{17}$$

Here we instead assume that the *source* type A is a set. Note the use of the propositional truncation operator $\|\_\|$. We will use this equivalence to translate traditional lenses where the source type is a set to higher lenses (§X). The right-to-left direction of the equivalence just applies the function to the value of type B. The other direction maps a value a to the function $\mathsf{set}\ l\ a$, a proof of $\parallel \mathsf{set}\ l^{\ -1}\ \mathsf{set}\ l\ a \parallel$, and the value $\mathsf{get}\ l\ a$. The two directions can be proved to be inverses of each other using the lens laws, the assumption that A is a set, and a variant of Lemma 48 for traditional lenses. The equivalence does not necessarily hold if A is not a set (see the accompanying code for details).

IV. Lenses Based on Bijections

Very well-behaved lenses from A to B are sometimes defined to be bijections between A and the Cartesian product of B and some new type [8]. In our setting with different type universes one might formulate this in the following way, where $\_\leftrightarrow\_$ stands for the type of bijections, or functions with quasi-inverses (and lsuc is a successor function for levels):

$$\begin{aligned}&\mathit{Lens}^{\mathrm{B}} : \mathit{Type}\ a \to \mathit{Type}\ b \to \mathit{Type}\ (\mathit{lsuc}\ (a \sqcup b)) \\ &\mathit{Lens}^{\mathrm{B}}\ \{a = a\}\ \{b = b\}\ A\ B = \\ &\quad (R : \mathit{Type}\ (a \sqcup b)) \times (A \leftrightarrow R \times B)\end{aligned} \tag{18}$$

However, this type is somewhat ill-behaved, even if the source and view types are sets. When the source and view types are both empty we get that the definition above is equivalent to *Type* (the lowest universe): $\mathit{Lens}^{\mathrm{B}}\ \bot\ \bot \simeq \mathit{Type}$. For a well-behaved notion of lens one might expect that there should be no choice in how to define a lens from the empty type to the empty type. This is the fact for traditional lenses (it follows from Lemma 14).

V. Higher Lenses

We obtain our first definition of higher lenses by tweaking the definition from the previous section. First we note that the type of equivalences is better behaved than the type of functions with quasi-inverses, as discussed in Section II. We also note that when the view type is empty, then we would like to force the "remainder type" R to be empty. We can do this by requiring there to be a function from the remainder type to the propositional truncation of the view type. We end up with the following definition, where "E" stands for "equivalence":

$$\begin{aligned}&\textbf{record}\ \mathit{Lens}^{\mathrm{E}}\ (A : \mathit{Type}\ a)\ (B : \mathit{Type}\ b) : \\ &\qquad\qquad \mathit{Type}\ (\mathit{lsuc}\ (a \sqcup b))\ \textbf{where} \\ &\quad \textbf{field}\ \mathsf{R} \qquad\quad : \mathit{Type}\ (a \sqcup b) \\ &\qquad\quad \mathsf{equiv} \qquad : A \simeq \mathsf{R} \times B \\ &\qquad\quad \mathsf{inhabited} : \mathsf{R} \to \parallel B \parallel\end{aligned} \tag{19}$$

The final field is not much of a restriction in practice: given an equivalence between A and $R \times B$, where R lives in the "right" universe, we can construct a lens from A to B:

$$\begin{aligned}&\{A : \mathit{Type}\ a\}\ \{B : \mathit{Type}\ b\}\ \{R : \mathit{Type}\ (a \sqcup b)\} \to \\ &A \simeq R \times B \to \mathit{Lens}^{\mathrm{E}}\ A\ B\end{aligned} \tag{20}$$

We let the remainder type of the lens be $R \times \parallel B \parallel$, which makes it easy to implement the $\mathsf{inhabited}$ field, and define the equiv field in the following way:

$$\begin{aligned}&A \ \simeq\ R \times B \ \simeq\ R \times (\parallel B \parallel \times B) \ \simeq \\ &\quad (R \times \parallel B \parallel) \times B\end{aligned}$$

The first step uses the supplied equivalence, the final step makes use of the fact that $\_\times\_$ is associative up to $\_\simeq\_$, and the second step uses the following equivalence: $\parallel A \parallel \times A \simeq A$.

Given a higher lens it is easy to define a getter and a setter, following Van Laarhoven [8]. We also define a function from the source type to the remainder type (proj_1 is the first projection, and proj_2 the second):

$$\begin{aligned}&\mathit{get} : A \to B \\ &\mathit{get}\ a = \mathsf{proj}_2\ (\mathit{to}\ \mathsf{equiv}\ a)\end{aligned} \tag{21}$$

$$\begin{aligned}&\mathit{remainder} : A \to \mathsf{R} \\ &\mathit{remainder}\ a = \mathsf{proj}_1\ (\mathit{to}\ \mathsf{equiv}\ a)\end{aligned} \tag{22}$$

$$\begin{aligned}&\mathit{set} : A \to B \to A \\ &\mathit{set}\ a\ b = \mathit{from}\ \mathsf{equiv}\ (\mathit{remainder}\ a\ , b)\end{aligned} \tag{23}$$

The setter leaves the "remainder part" of the source value unchanged, but it replaces the "view part" with the new view value. Note that the $\mathsf{inhabited}$ field could equivalently have stated that the *remainder* function is surjective. (For simplicity we do not include the lens as an argument of the definitions above. After discussing lens laws below we will write things like $\mathit{get}\ l$ instead of get.)

Following Van Laarhoven [8] we can also prove the lens laws. Let us start with the *get-set* law. If we unfold the definitions of get and set, then we see that we should prove that $\mathsf{proj_2}\ (to\ \mathsf{equiv}\ (from\ \mathsf{equiv}\ (remainder\ a\ ,\ b)))$ is equal to b, which is equal to $\mathsf{proj_2}\ (remainder\ a\ ,\ b)$. This follows because equiv is an equivalence:

$$\begin{aligned}&get\text{-}set : \forall\ a\ b \to get\ (set\ a\ b) \equiv b\\ &get\text{-}set\ a\ b =\\ &\quad cong\ \mathsf{proj_2}\ (to\text{-}from\ \mathsf{equiv}\ (remainder\ a\ ,\ b))\end{aligned} \tag{24}$$

A similar argument shows that the setter leaves the remainder unchanged:

$$\begin{aligned}&remainder\text{-}set :\\ &\quad \forall\ a\ b \to remainder\ (set\ a\ b) \equiv remainder\ a\\ &remainder\text{-}set\ a\ b =\\ &\quad cong\ \mathsf{proj_1}\ (to\text{-}from\ \mathsf{equiv}\ (remainder\ a\ ,\ b))\end{aligned} \tag{25}$$

For the $set\text{-}get$ law we should prove that the application $from\ \mathsf{equiv}\ (to\ \mathsf{equiv}\ a)$ is equal to a. This also follows because equiv is an equivalence:

$$\begin{aligned}&set\text{-}get : \forall\ a \to set\ a\ (get\ a) \equiv a\\ &set\text{-}get\ a = from\text{-}to\ \mathsf{equiv}\ a\end{aligned} \tag{26}$$

Finally, for the $set\text{-}set$ law we should prove that the application $from\ \mathsf{equiv}\ (remainder\ (set\ a\ b_1)\ ,\ b_2)$ is equal to $from\ \mathsf{equiv}\ (remainder\ a\ ,\ b_2)$. This follows from the $remainder\text{-}set$ law:

$$\begin{aligned}&set\text{-}set : \forall\ a\ b_1\ b_2 \to set\ (set\ a\ b_1)\ b_2 \equiv set\ a\ b_2\\ &set\text{-}set\ a\ b_1\ b_2 =\\ &\quad cong\ (\lambda\ r \to from\ \mathsf{equiv}\ (r\ ,\ b_2))\\ &\qquad (remainder\text{-}set\ a\ b_1)\end{aligned} \tag{27}$$

With the lens laws proved we know that every higher lens can be translated to a traditional lens with the same getter and setter. We do not know if there is always such a translation in the other direction, but there is one when the source type is a set (§X). If the view type is stable, then a traditional lens that satisfies the two coherence laws discussed in the following section can also be translated to a higher lens with the same getter and setter (see the accompanying code for details).

VI. Coherence Laws

Higher lenses satisfy some coherence laws that do not necessarily hold for traditional lenses (as discussed in Section III). First consider the $get\text{-}set\text{-}get$ law:

$$cong\ get\ (set\text{-}get\ a) \equiv get\text{-}set\ a\ (get\ a) \tag{28}$$

We can prove this law in the following way:

$$\begin{aligned}&cong\ get\ (set\text{-}get\ a) &&\equiv\\ &cong\ (\mathsf{proj_2} \circ to\ \mathsf{equiv})\ (from\text{-}to\ \mathsf{equiv}\ a) &&\equiv\\ &cong\ \mathsf{proj_2}\ (cong\ (to\ \mathsf{equiv})\ (from\text{-}to\ \mathsf{equiv}\ a)) &&\equiv\\ &cong\ \mathsf{proj_2}\ (to\text{-}from\ \mathsf{equiv}\ (to\ \mathsf{equiv}\ a)) &&\equiv\\ &get\text{-}set\ a\ (get\ a)\end{aligned}$$

The first and last steps hold by definition. The second step uses the fact that $cong$ satisfies some functor-like laws. Finally the third step uses Lemma 6.

We can also prove the other coherence law mentioned above, $get\text{-}set\text{-}set$:

$$\begin{aligned}&cong\ get\ (set\text{-}set\ a\ b_1\ b_2) \equiv\\ &trans\ (get\text{-}set\ (set\ a\ b_1)\ b_2)\ (sym\ (get\text{-}set\ a\ b_2))\end{aligned} \tag{29}$$

Due to lack of space the proof is omitted.

One might wonder how far one can take this: do all conceivable coherence laws of this kind hold for higher lenses? We do not have an answer to this question. Instead we explore other properties of higher lenses, and return to questions about coherence below.

VII. Some Equivalences

Let us now discuss some equivalences satisfied by $Lens^{\mathrm{E}}$. We begin by proving that if the view type is a proposition, then $Lens^{\mathrm{E}}\ A\ B$ is equivalent to the type of its getter:

$$Is\text{-}proposition\ B \to Lens^{\mathrm{E}}\ A\ B \simeq (A \to B) \tag{30}$$

Note the simplification compared to the corresponding result for traditional lenses (14). We can build up the equivalence in the following way:

$$\begin{array}{lll}Lens^{\mathrm{E}}\ A\ B & \simeq & \text{(a)}\\ (\exists\ R \times (A \simeq R \times B) \times (R \to \parallel B \parallel)) & \simeq & \text{(b)}\\ (\exists\ R \times (A \simeq R \times B) \times (R \to B)) & \simeq & \text{(c)}\\ (\exists\ R \times (A \simeq R) \qquad \times (R \to B)) & \simeq & \text{(d)}\\ (\exists\ R \times (A \simeq R) \qquad \times (A \to B)) & \simeq & \text{(e)}\\ (\exists\ R \times (A \simeq R)) \qquad \times (A \to B) & \simeq & \text{(f)}\\ (A \to B) & & \end{array}$$

Step a uses the fact that the record type $Lens^{\mathrm{E}}\ A\ B$ can be expressed as a nested Σ-type. Step b uses the assumption that B is a proposition. Step c uses the fact that inhabited propositions are contractible, along with the assumptions that there is a function from R to B and a value in R (one can prove that $R \times B$ is equivalent to $R \times C$ by proving $R \to B \simeq C$). Step d uses the equivalence between A and R. Step e uses the fact that the Σ type former is "associative" up to $\_\simeq\_$. Finally step f uses the fact that types of the form $(x : A) \times x \equiv y$, *singleton types*, are contractible [4], and in the presence of univalence equality of types is equivalent to equivalence of types.

As a consequence of Lemma 30 we get the following results (the second one holds also for traditional lenses):

$$Lens^{\mathrm{E}}\ A\ \top \simeq \top \tag{31}$$

$$Lens^{\mathrm{E}}\ A\ \bot \simeq \neg\ A \tag{32}$$

We can also prove the following lemmas (both of which also hold for traditional lenses):

$$Lens^{\mathrm{E}}\ \top\ B \simeq Contractible\ B \tag{33}$$

$$Lens^{\mathrm{E}}\ \bot\ B \simeq \top \tag{34}$$

The getter of a lens is an equivalence if and only if the inhabited field is:

$$\begin{aligned}&Is\text{-}equivalence\ (get\ l) \simeq\\ &Is\text{-}equivalence\ (\mathsf{inhabited}\ l)\end{aligned} \tag{35}$$

We prove this in the following way (here B is the view type of l):

$$\begin{array}{ll} \mathit{Is\text{-}equivalence}\ (\mathit{get}\ l) & \simeq \\ \mathit{Is\text{-}equivalence}\ (\mathsf{proj_2} : \mathsf{R}\ l \times B \to B)) & \simeq \\ \mathit{Is\text{-}equivalence}\ (\mathsf{inhabited}\ l) & \end{array}$$

The first step uses the 2-out-of-3 property [4]: if two of the functions f, g and $f \circ g$ are equivalences, then the third one is also an equivalence. In this case one of the equivalences is *to* (equiv l). The second step uses a general property that can be proved by giving functions in both directions.

The type of equivalences between A and B can be expressed in terms of lenses for which the getter is an equivalence:

$$\begin{array}{l} (A \simeq B) \simeq \\ (l : \mathit{Lens}^{\mathrm{E}}\ A\ B) \times \mathit{Is\text{-}equivalence}\ (\mathit{get}\ l) \end{array} \tag{36}$$

We can prove this by the following chain of equivalences:

$$\begin{array}{ll} ((l : \mathit{Lens}^{\mathrm{E}}\ A\ B) \times \mathit{Is\text{-}equivalence}\ (\mathit{get}\ l)) & \simeq \\ ((l : \mathit{Lens}^{\mathrm{E}}\ A\ B) \times \mathit{Is\text{-}equivalence}\ (\mathsf{inhabited}\ l)) & \simeq \\ (((R\ ,\ \_) : \exists\ R \times (R \simeq \|\ B\ \|)) \times (A \simeq R \times B)) & \simeq \\ A \simeq \|\ B\ \| \times B & \simeq \\ A \simeq B & \end{array}$$

The first step uses Lemma 35, the second step rearranges the type, the third step removes a singleton type, and finally the last step uses the fact that $\|\ B\ \| \times B$ is equivalent to B. Our implementation of the right-to-left direction of Lemma 36 returns the getter of the lens and some proof (this holds by definition). Lemma 36 does not in general hold for traditional lenses (see the accompanying code for details).

We can also prove the following variant of Lemma 16, without assuming that the view type is a set:

$$(l : \mathit{Lens}^{\mathrm{E}}\ A\ B)\ (b : B) \to \mathsf{R}\ l \simeq \mathit{get}\ l^{\ -1}\ b \tag{37}$$

We have implemented the equivalence in the following way: The right-to-left direction throws away the equality proof and applies the *remainder* function: $\lambda\ (a\ ,\ \_) \to \mathit{remainder}\ l\ a$. The left-to-right direction maps r to *from* (equiv l) $(r\ ,\ b)$, along with the following proof:

$$\begin{array}{ll} \mathit{get}\ l\ (\mathit{from}\ (\mathsf{equiv}\ l)\ (r\ ,\ b)) & \equiv \\ \mathsf{proj_2}\ (\mathit{to}\ (\mathsf{equiv}\ l)\ (\mathit{from}\ (\mathsf{equiv}\ l)\ (r\ ,\ b))) & \equiv \\ b & \end{array}$$

The first step holds by definition, and the second step uses *to-from* (equiv l). We do not include our proofs showing that these two functions are inverses of each other, because one of them is fairly long. See the accompanying code for details.

VIII. Equality of Lenses With Equal Setters

When proving that two higher lenses are equal it sometimes suffices to prove that the setters are equal. This does not hold unconditionally, but the results we present below cover quite a few cases. As an example of how these results can be used, see Lemma 60, which states that every composition operator for which *set* is implemented in the canonical way is equal, assuming that the view type of the resulting lenses is stable.

We begin by stating a lemma that characterises equality of lenses:

$$\begin{array}{l} l_1 \equiv l_2 \simeq \\ \quad (\mathit{eq} : \mathsf{R}\ l_1 \simeq \mathsf{R}\ l_2) \times \\ \quad \forall\ a \to (\mathit{to}\ \mathit{eq}\ (\mathit{remainder}\ l_1\ a)\ ,\ \mathit{get}\ l_1\ a) \equiv \\ \qquad\quad \mathit{to}\ (\mathsf{equiv}\ l_2)\ a \end{array} \tag{38}$$

This result can be proved using univalence. As an aside, if the view type is inhabited, then we can use Lemma 37 to characterise equality of lenses without referring to remainder types:

$$\begin{array}{l} (l_1\ l_2 : \mathit{Lens}^{\mathrm{E}}\ A\ B)\ (b : B) \to \\ l_1 \equiv l_2 \simeq \\ \quad (\mathit{eq} : \mathit{get}\ l_1{}^{\ -1}\ b \simeq \mathit{get}\ l_2{}^{\ -1}\ b) \times \\ \quad (\forall\ a \to \mathit{to}\ \mathit{eq}\ (\mathit{set}\ l_1\ a\ b\ ,\ \mathit{get\text{-}set}\ l_1\ a\ b) \equiv \\ \qquad\quad (\mathit{set}\ l_2\ a\ b\ ,\ \mathit{get\text{-}set}\ l_2\ a\ b)) \times \\ \quad (\forall\ a \to \mathit{get}\ l_1\ a \equiv \mathit{get}\ l_2\ a) \end{array} \tag{39}$$

In the case where the view type is inhabited it is easy to prove that lenses with equal setters are equal:

$$\begin{array}{l} (l_1\ l_2 : \mathit{Lens}^{\mathrm{E}}\ A\ B) \to B \to \mathit{set}\ l_1 \equiv \mathit{set}\ l_2 \to \\ l_1 \equiv l_2 \end{array} \tag{40}$$

We use Lemma 38. Given a value $b : B$ we can prove that the two lenses' remainder types are equivalent:

$$\mathsf{R}\ l_1 \quad \simeq \quad \mathit{get}\ l_1{}^{\ -1}\ b \quad \simeq \quad \mathit{get}\ l_2{}^{\ -1}\ b \quad \simeq \quad \mathsf{R}\ l_2$$

The first and last steps use Lemma 37. The second step uses the fact that if the setters are equal, then the getters are equal (as mentioned above this follows from the lens laws). It remains to establish the equality in the statement of Lemma 38, but with our implementation of the equivalence between the remainder types this turns out to be straightforward.

What can we do if we do not have a witness showing that the view type is inhabited? We have proved the following results (for the proofs, see the accompanying code):

$$\begin{array}{l} (l_1\ l_2 : \mathit{Lens}^{\mathrm{E}}\ A\ B) \to (\mathsf{R}\ l_1 \to \|\ B\ \| \to B) \to \\ \quad \mathit{set}\ l_1 \equiv \mathit{set}\ l_2 \to l_1 \equiv l_2 \end{array} \tag{41}$$

$$\begin{array}{l} (l_1\ l_2 : \mathit{Lens}^{\mathrm{E}}\ A\ B) \to \mathit{Is\text{-}set}\ (\mathsf{R}\ l_2) \to \\ \quad \mathit{set}\ l_1 \equiv \mathit{set}\ l_2 \to l_1 \equiv l_2 \end{array} \tag{42}$$

Note that B is stable ($\|\ B\ \| \to B$) both when B is inhabited and when it is empty.

However, one can prove a *negation* to the statement that, for all types A and B in a given universe *Type a*, all lenses l_1 and l_2 from A to B with equal setters are equal:

$$\begin{array}{l} \neg\ ((A\ B : \mathit{Type}\ a)\ (l_1\ l_2 : \mathit{Lens}^{\mathrm{E}}\ A\ B) \to \\ \qquad \mathit{set}\ l_1 \equiv \mathit{set}\ l_2 \to l_1 \equiv l_2) \end{array} \tag{43}$$

This follows from the following result (the proof is omitted):

$$\begin{array}{l} \{A\ B\ C : \mathit{Type}\ a\} \to \\ ((l_1\ l_2 : \mathit{Lens}^{\mathrm{E}}\ (A \times C)\ C) \to \\ \quad \mathit{set}\ l_1 \equiv \mathit{set}\ l_2 \to l_1 \equiv l_2) \to \\ (f : C \to A \simeq B) \to \mathit{Constant}\ f \to \mathit{CC}\ f \end{array} \tag{44}$$

Constant f means that f is *weakly constant*, and *CC* f means that f is *coherently* or *conditionally* constant:

$$\begin{array}{l} \mathit{Constant} : \{A : \mathit{Type}\ a\}\ \{B : \mathit{Type}\ b\} \to \\ \qquad\qquad (A \to B) \to \mathit{Type}\ (a \sqcup b) \\ \mathit{Constant}\ f = \forall\ x\ y \to f\ x \equiv f\ y \end{array} \tag{45}$$

$$\begin{aligned} & \mathit{CC} : \{A : \mathit{Type}\ a\}\ \{B : \mathit{Type}\ b\} \to \\ & \qquad (A \to B) \to \mathit{Type}\ (a \sqcup b) \\ & \mathit{CC}\ \{A = A\}\ \{B = B\}\ f = \\ & \qquad (g : \|\ A\ \| \to B) \times f \equiv g \circ |\_| \end{aligned} \tag{46}$$

Here $|\_|$ is a constructor of the propositional truncation operator, of type $B \to \|\ B\ \|$. Every coherently constant function is weakly constant, and Kraus et al. [16] have proved that every weakly constant function with a stable domain is coherently constant. We can conclude the proof of Lemma 43 using the following result, which was proved by Christian Sattler (personal communication), building on work by Shulman [17]:

$$\begin{aligned} & \neg\ ((A\ B : \mathit{Type}\ a) \to \|\ A\ \| \to \\ & \qquad (f : A \to B \simeq B) \to \mathit{Constant}\ f \to \mathit{CC}\ f) \end{aligned} \tag{47}$$

One can prove Lemma 47 by letting "B" be the Eilenberg-MacLane space $K(\mathbb{Z}/2\mathbb{Z}, 1)$ [18] (suitably lifted).

Lemma 42 applies to lenses for which the remainder type is a set. One might wonder how the h-level of the remainder type relates to the h-level of the source type. Let us now consider questions of this kind.

IX. Homotopy Levels

If there is a lens from A to B, and A is an inhabited type with h-level n, then B has h-level n (this holds also for traditional lenses):

$$\mathit{Lens}^{\mathrm{E}}\ A\ B \to A \to \mathit{H\text{-}level}\ n\ A \to \mathit{H\text{-}level}\ n\ B \tag{48}$$

To prove this we can make use of the fact that if there is a split surjection (a function with a right inverse) from A to B, and A has h-level n, then B has h-level n [4]. The lens's getter gives us a function from A to B, the setter applied to the given element of A gives us a function in the other direction, and due to the get-set law the second function is a right inverse of the first one. Note that this result does not necessarily hold when A is empty: there is a lens from the empty type to the booleans, and the empty type is a proposition, but the type of booleans is not.

As a corollary of Lemma 48 we get that the view type of a lens with a contractible source type is contractible:

$$\mathit{Lens}^{\mathrm{E}}\ A\ B \to \mathit{Contractible}\ A \to \mathit{Contractible}\ B \tag{49}$$

This follows because contractible types are inhabited. Using this result we can prove that there is, in general, no lens from $(x : A) \times P\ x$ to A:

$$\neg\ \mathit{Lens}^{\mathrm{E}}\ ((b : \mathit{Bool}) \times b \equiv \mathsf{true})\ \mathit{Bool} \tag{50}$$

(Again this holds also for traditional lenses.) This follows because the singleton type $(b : \mathit{Bool}) \times b \equiv \mathsf{true}$ is contractible, but the type of booleans is not.

We can also prove that if the source type has h-level n, then the remainder type also has this h-level:

$$\begin{aligned} & (l : \mathit{Lens}^{\mathrm{E}}\ A\ B) \to \\ & \mathit{H\text{-}level}\ n\ A \to \mathit{H\text{-}level}\ n\ (\mathsf{R}\ l) \end{aligned} \tag{51}$$

If A has h-level n, then the equivalent type $\mathsf{R}\ l \times B$ also has this h-level. We also have the fact that if $\mathsf{R}\ l \times B$ has h-level n, then $\mathsf{R}\ l$ has this h-level, as long as there is a function from $\mathsf{R}\ l$ to $\|\ B\ \|$.

If a lens's getter is an equivalence, then the remainder type is propositional:

$$\mathit{Is\text{-}equivalence}\ (\mathit{get}\ l) \to \mathit{Is\text{-}proposition}\ (\mathsf{R}\ l) \tag{52}$$

If the getter is an equivalence, then Lemma 35 implies that the remainder type is equivalent to the propositional truncation of the view type, and thus the remainder type is a proposition.

We also have the following two properties, which follow from Lemma 30:

$$\mathit{Contractible}\quad B \to \mathit{Contractible}\quad (\mathit{Lens}^{\mathrm{E}}\ A\ B) \tag{53}$$

$$\mathit{Is\text{-}proposition}\ B \to \mathit{Is\text{-}proposition}\ (\mathit{Lens}^{\mathrm{E}}\ A\ B) \tag{54}$$

Our main result about h-levels is deferred to Section XV, where we use a different representation of lenses to prove that, if A has positive h-level n, then the type of higher lenses from A to B has h-level n.

X. Higher and Traditional Lenses are Equivalent for Sets

Let us now investigate under what circumstances $\mathit{Lens}^{\mathrm{E}}\ A\ B$ and $\mathit{Lens}^{\mathrm{T}}\ A\ B$ are equivalent.

First note that in the general case there might not even be a split surjection from $\mathit{Lens}^{\mathrm{E}}\ A\ B$ to $\mathit{Lens}^{\mathrm{T}}\ A\ B$:

$$\neg\ (\mathit{Lens}^{\mathrm{E}}\ \mathbb{S}^1\ \top \twoheadrightarrow \mathit{Lens}^{\mathrm{T}}\ \mathbb{S}^1\ \top) \tag{55}$$

This follows because $\mathit{Is\text{-}proposition}$ respects split surjections: $\mathit{Lens}^{\mathrm{E}}\ \mathbb{S}^1\ \top$ is a proposition, but $\mathit{Lens}^{\mathrm{T}}\ \mathbb{S}^1\ \top$ is not.

However, when the source type is a set there is an equivalence:

$$\mathit{Is\text{-}set}\ A \to \mathit{Lens}^{\mathrm{E}}\ A\ B \simeq \mathit{Lens}^{\mathrm{T}}\ A\ B \tag{56}$$

The forward direction of the equivalence returns a lens where the getter, the setter and the lens laws are defined as in Section V. This direction does not make use of the assumption that A is a set. However, the other direction does: we use Lemma 17 to construct an equivalence between A and $((f : B \to A) \times \|\ \mathsf{set}\ l\ ^{-1}\ f\ \|) \times B$, and turn this into a higher lens using Lemma 20.

Our implementations of both translations preserve getters and setters by definition (see the accompanying code for details), so one can prove that the round-trip from a traditional lens to a higher one and back yields the original lens by proving that the proofs of the lens laws are pointwise equal. The set-get and set-set laws return equalities between values of type A, and because A is a set these equality types are propositions. The get-set law returns equalities between values of type B. Because we have a lens from A to B we get by Lemma 48 (for traditional lenses) that B is a set whenever we have a witness showing that A is inhabited, and we can use the first argument of the get-set law as the witness.

Our proof of the other round-trip property is more complicated. We use Lemma 38, so we start by proving that the remainder type of the resulting lens is equivalent to the remainder type of the original one:

$$\begin{array}{ll}((f : B \to A) \times \| \, set \; l \;^{-1} \; f \, \|) \times \| \, B \, \| & \simeq \\ (\| \, B \, \| \to \mathsf{R} \; l) \times \| \, B \, \| & \simeq \\ \mathsf{R} \; l & \end{array}$$

The last step is proved by defining functions in both directions (one uses the inhabited field of l) and proving that they are inverses of each other. The first step uses the following equivalence, which is proved under the assumption that $\| \, B \, \|$ is inhabited:

$$\begin{array}{lr}((f : B \to A) \times \| \, set \; l \;^{-1} \; f \, \|) & \simeq \text{ (a)} \\ ((f : B \to A) \times \forall \; b \; b' \to set \; l \; (f \; b) \; b' \equiv f \; b') & \simeq \text{ (b)} \\ ((f : B \to \mathsf{R} \; l \times B) \times & \\ \quad \forall \; b \; b' \to (\mathsf{proj}_1 \; (f \; b) \; , \; b') \equiv f \; b') & \simeq \text{ (c)} \\ (((f \; , \; g) : (B \to \mathsf{R} \; l) \times (B \to B)) \times & \\ \quad \forall \; b \; b' \to f \; b \equiv f \; b' \times b' \equiv g \; b') & \simeq \text{ (d)} \\ ((f : B \to \mathsf{R} \; l) \times Constant \; f) \times & \\ \quad ((g : B \to B) \times (B \to \forall \; b \to b \equiv g \; b)) & \simeq \text{ (e)} \\ ((f : B \to \mathsf{R} \; l) \times Constant \; f) \times & \\ \quad ((g : B \to B) \times \forall \; b \to b \equiv g \; b) & \simeq \text{ (f)} \\ ((f : B \to \mathsf{R} \; l) \times Constant \; f) \times & \\ \quad ((g : B \to B) \times id \equiv g) & \simeq \text{ (g)} \\ ((f : B \to \mathsf{R} \; l) \times Constant \; f) & \simeq \text{ (h)} \\ (\| \, B \, \| \to \mathsf{R} \; l) & \end{array}$$

$Constant$ is defined above (45). Step a is proved by defining functions in both directions (the right-to-left direction uses the assumption that $\| \, B \, \|$ is inhabited) and proving that they are inverses of each other. Step b uses the equivalence equiv l. Steps c and d rearrange the types. Step e drops an argument. This is possible because there is another argument of the same type and the result type is a proposition (because A is a set and there is a higher lens from A to B Lemma 48 implies that B is a set if A is inhabited, and we can assume that there is a function from B to $\mathsf{R} \; l$ as well as a value of type B). Step f rearranges types again, and step g drops a contractible type. Finally step h uses the fact that the type of functions from $\| \, B \, \|$ to $\mathsf{R} \; l$ is equivalent to the type of weakly constant functions from $\| \, B \, \|$ to $\mathsf{R} \; l$ if $\mathsf{R} \; l$ is a set [19], and $\mathsf{R} \; l$ is a set because A is a set (51). Our use of Lemma 38 requires that we also prove an equality, but with our implementation of the equivalence that is easy.

XI. Identity and Composition

It is straightforward to define an identity lens:

$$id : Lens^{\mathrm{E}} \; A \; A \tag{57}$$

Furthermore any two lenses from A to A with the identity function id as their getter are equal:

$$\begin{array}{l}(l_1 \; l_2 : Lens^{\mathrm{E}} \; A \; A) \to \\ get \; l_1 \equiv id \to get \; l_2 \equiv id \to l_1 \equiv l_2\end{array} \tag{58}$$

This follows from Lemma 36 (with its stated computational behaviour).

We can also define a composition operator. For simplicity we give it for types in the same universe:

$$\begin{array}{l}\_\circ\_ : \{A \; B \; C : Type \; a\} \to \\ \quad Lens^{\mathrm{E}} \; B \; C \to Lens^{\mathrm{E}} \; A \; B \to Lens^{\mathrm{E}} \; A \; C\end{array} \tag{59}$$

We let the remainder type of $l_1 \circ l_2$ be $\mathsf{R} \; l_2 \times \mathsf{R} \; l_1$. It is easy to define the inhabited field. We define the equiv field in the following way:

$$\begin{array}{l}A \;\; \simeq \;\; \mathsf{R} \; l_2 \times B \;\; \simeq \;\; \mathsf{R} \; l_2 \times (\mathsf{R} \; l_1 \times C) \;\; \simeq \\ \quad (\mathsf{R} \; l_2 \times \mathsf{R} \; l_1) \times C\end{array}$$

The first two steps use the equiv fields of the two lenses, and the last step uses associativity of $\_\times\_$.

Our implementation of composition also works for lenses for which the universe of the source type is at least as large as the universe of the view type. However, in the case where A and C belong to a smaller universe and B to a larger one we do not know how to make this approach work. The problem is that $\mathsf{R} \; l_2 \times \mathsf{R} \; l_1$ is too large. Below we use coinductive higher lenses to define a composition operator that works for types in arbitrary universes (95). (One might wonder if there are any lenses from A to B where B is in a larger universe than A. If A and B are equivalent, then one can construct such a lens.)

It is easy to prove that composition is associative, with id as a left and right unit. Our proofs use Lemma 38. As an aside we note that analogous properties hold unconditionally for at least one implementation of composition for traditional lenses. See the accompanying code for details.

When the view type of the resulting lenses is stable every composition operator for which set is implemented in the canonical way is equal:

$$\begin{array}{l}(\| \, C \, \| \to C) \to \\ (c_1 \; c_2 : Lens^{\mathrm{E}} \; B \; C \to Lens^{\mathrm{E}} \; A \; B \to Lens^{\mathrm{E}} \; A \; C) \to \\ (\forall \; l_1 \; l_2 \; a \; c \to set \; (c_1 \; l_1 \; l_2) \; a \; c \equiv \\ \qquad\qquad set \; l_2 \; a \; (set \; l_1 \; (get \; l_2 \; a) \; c)) \to \\ (\forall \; l_1 \; l_2 \; a \; c \to set \; (c_2 \; l_1 \; l_2) \; a \; c \equiv \\ \qquad\qquad set \; l_2 \; a \; (set \; l_1 \; (get \; l_2 \; a) \; c)) \to \\ c_1 \equiv c_2\end{array} \tag{60}$$

This follows from Lemma 41. One can check that the composition operator defined above produces lenses for which set satisfies this kind of equality.

A lens (between types in the same universe) is bi-invertible if it has a left inverse, and also a right inverse:

$$\begin{array}{l}Is\text{-}bi\text{-}invertible : \{A \; B : Type \; a\} \to \\ \qquad\qquad Lens^{\mathrm{E}} \; A \; B \to Type \; (lsuc \; a) \\ Is\text{-}bi\text{-}invertible \; l = \\ \quad (\exists \; l' \times l' \circ l \equiv id) \times (\exists \; l' \times l \circ l' \equiv id)\end{array} \tag{61}$$

We have proved the following two equivalences:

$$(A \simeq B) \simeq ((l : Lens^{\mathrm{E}} \; A \; B) \times Is\text{-}bi\text{-}invertible \; l) \tag{62}$$
$$Is\text{-}equivalence \; (get \; l) \simeq Is\text{-}bi\text{-}invertible \; l \tag{63}$$

The first of the equivalences maps a bi-invertible lens to an equivalence for which the forward direction is the lens's getter. For traditional lenses—with certain definitions of identity and composition—we have proved an analogue of Lemma 63. However, Lemma 62 does not in general hold for traditional lenses, although there is always a split surjection from the right-hand side to the left-hand side. See the accompanying code for details.

Using the implementations of identity and composition discussed above we have defined univalent categories [4] of both higher and traditional lenses between sets with a fixed universe level. For each universe level the category of higher lenses is equal to a lifted variant of the category of traditional ones (note that $\mathit{Lens}^{\mathrm{E}}\ A\ B$ is large). As part of the proof of this fact we established that composition of traditional lenses from sets (where all source and view types have the same universe level) can be expressed in terms of composition of higher lenses and the equivalence between higher and traditional lenses (56). Again, see the accompanying code for details.

We conclude this section with some results that are based on questions asked by an anonymous reviewer. Let us use the term *naive category* for the concept that we obtain if we take the definition of a (not necessarily univalent) precategory [4] and drop the requirement that the morphism types must be sets. Let us also call such a category *univalent* if the canonical function from "A is equal to B" to "there is a bi-invertible morphism from A to B" is an equivalence for all objects A and B. We have proved that, for each universe level a, our definition of a naive category of higher lenses between *types* in $\mathit{Type}\ a$ is univalent (this follows from Lemma 62), whereas our definition of a naive category of traditional lenses between types in $\mathit{Type}\ a$ is not univalent. The latter fact can be used to prove that Lemmas 36 and 62 do not in general hold for traditional lenses. See the accompanying code for details.

XII. Coherently Constant Families of Fibres

In this section we present another variant of higher lenses, consisting of a getter and a proof that the family of fibres of the getter is coherently constant. "F" stands for "fibres":

$$\begin{aligned}&\mathit{Lens}^{\mathrm{F}} : \mathit{Type}\ a \to \mathit{Type}\ b \to \mathit{Type}\ (\mathit{lsuc}\ (a \sqcup b))\\&\mathit{Lens}^{\mathrm{F}}\ A\ B = (\mathit{get} : A \to B) \times \mathit{CC}\ (\mathit{get}\ ^{-1}\_)\end{aligned} \tag{64}$$

This type is used to prove that $\mathit{Lens}^{\mathrm{E}}$ is equivalent to $\mathit{Lens}^{\mathrm{C}}$, which is defined below (90). $\mathit{Lens}^{\mathrm{C}}$ is defined in the same way as $\mathit{Lens}^{\mathrm{F}}$, but with CC^{S} (88) instead of CC (46).

The type $\mathit{CC}\ (\mathit{get}\ ^{-1}\_)$ in the definition of $\mathit{Lens}^{\mathrm{F}}$ is equal to $(G : \parallel B \parallel \to \mathit{Type}\ (a \sqcup b)) \times (\mathit{get}\ ^{-1}\_) \equiv G \circ |\_|$. Because coherently constant functions are weakly constant we get that the family of fibres of get is weakly constant:

$$\begin{aligned}&\mathit{get}^{-1}\text{-}\mathit{constant} :\\&\quad (b_1\ b_2 : B) \to \mathit{get}\ ^{-1}\ b_1 \simeq \mathit{get}\ ^{-1}\ b_2\end{aligned} \tag{65}$$

Using $\mathit{get}^{-1}\text{-}\mathit{constant}$ we can define a setter:

$$\begin{aligned}&\mathit{set} : A \to B \to A\\&\mathit{set}\ a\ b = \mathsf{proj}_1\ (\mathit{to}\ (\mathit{get}^{-1}\text{-}\mathit{constant}\ (\mathit{get}\ a)\ b)\\&\qquad\qquad\qquad (a\ ,\ \mathit{refl}\ (\mathit{get}\ a)))\end{aligned} \tag{66}$$

We do not proceed to prove the lens laws, instead we prove that there is an equivalence between $\mathit{Lens}^{\mathrm{F}}\ A\ B$ and $\mathit{Lens}^{\mathrm{E}}\ A\ B$ that preserves getters and setters:

$$\mathit{Lens}^{\mathrm{F}}\ A\ B \simeq \mathit{Lens}^{\mathrm{E}}\ A\ B \tag{67}$$

The right-to-left direction of the equivalence takes a lens l to a triple $(\mathit{get}\ l\ ,\ (\lambda\ \_ \to \mathsf{R}\ l)\ ,\ \mathit{eq})$, where the equality eq is defined using univalence and Lemma 37. The left-to-right direction takes a triple $(\mathit{get}\ ,\ G\ ,\ \mathit{eq})$ to a lens where the R field is $(b : \parallel B \parallel) \times G\ b$. This makes it easy to define the $\mathsf{inhabited}$ field. The equiv field is defined via the following chain of equivalences:

$$\begin{array}{ll}A & \simeq \text{(a)}\\((a : A) \times (b : B) \times \mathit{get}\ a \equiv b) & \simeq \text{(b)}\\((b : B) \times \mathit{get}\ ^{-1}\ b) & \simeq \text{(c)}\\((b : B) \times G \mid b \mid) & \simeq \text{(d)}\\(((b\ ,\ \_) : \parallel B \parallel \times B) \times G\ b) & \simeq \text{(e)}\\((b : \parallel B \parallel) \times G\ b) \times B & \end{array}$$

Step a introduces a singleton type, and step b rearranges things a little. Step c uses the equality eq. Step d uses the fact that $\parallel B \parallel \times B$ is equivalent to B, and step e rearranges things a little again. See the accompanying code for proofs showing that the left-to-right and right-to-left directions of the equivalence are inverses of each other, and that they preserve getters and setters.

XIII. Coinductive Higher Lenses

In this section we present yet another variant of higher lenses. Unlike the ones presented above this variant is small. This variant will be used to prove some results about the h-level of a lens (102–103) and to define an unrestricted composition operator (95).

As shown by Kraus [19] the propositional truncation of a type can be obtained as the colimit of a certain semi-simplicial diagram (its Čech nerve). For us this means that we could in principle express coherent constancy of the family of fibres of get as an infinite tower of coherence conditions, the first of which corresponds to $\mathit{get}^{-1}\text{-}\mathit{constant}$ (65). However, it is currently not known whether semi-simplicial diagrams, and towers of coherence conditions based on them, can be encoded in homotopy type theory. Fortunately there are alternative characterisations of propositional truncation as a colimit which *can* be encoded in type theory [13], [20], [21].

Let us use the construction due to Van Doorn [13]. First the *one-step truncation* is defined as a higher inductive type:

$$\begin{aligned}&\textbf{data}\ \|\_\|^1\ (A : \mathit{Type}\ a) : \mathit{Type}\ a\ \textbf{where}\\&\quad |\_| \qquad\qquad : A \to \parallel A \parallel^1\\&\quad ||\text{-}\mathsf{constant} : \mathit{Constant}\ |\_|\end{aligned} \tag{68}$$

$\mathit{Constant}$ is defined above (45). The one-step truncation comes with a non-dependent eliminator:

$$\mathit{rec} : (f : A \to B) \to \mathit{Constant}\ f \to \parallel A \parallel^1 \to B \tag{69}$$

The following equivalence is established by Van Doorn:

$$\begin{aligned}&(\parallel A \parallel \to B) \simeq\\&\quad (f : \forall\ n \to \parallel A \parallel^1\text{-}\mathit{out}\ n \to B) \times\\&\quad \forall\ n\ x \to f\ (1 + n) \mid x \mid \equiv f\ n\ x\end{aligned} \tag{70}$$

Here the type $\parallel A \parallel^1\text{-}\mathit{out}\ n$ consists of n applications of $\|\_\|^1$ to A:

$$\begin{array}{l} \|\_\|^1\text{-}out : Type\ a \to \mathbb{N} \to Type\ a \\ \|\ A\ \|^1\text{-}out\ \mathsf{zero} \quad = A \\ \|\ A\ \|^1\text{-}out\ (\mathsf{suc}\ n) = \|\ \|\ A\ \|^1\text{-}out\ n\ \|^1 \end{array} \tag{71}$$

We will also use the following variant, where the final application of $\|\_\|^1$ is on the inside instead of on the outside:

$$\begin{array}{l} \|\_\|^1\text{-}in : Type\ a \to \mathbb{N} \to Type\ a \\ \|\ A\ \|^1\text{-}in\ \mathsf{zero} \quad = A \\ \|\ A\ \|^1\text{-}in\ (\mathsf{suc}\ n) = \|\ \|\ A\ \|^1\ \|^1\text{-}in\ n \end{array} \tag{72}$$

We use this variant in addition to the other one because we have found that in some cases one is more convenient, in some cases the other. For the first one the constructor $|\_|$ takes values from $\|\ A\ \|^1\text{-}out\ n$ to $\|\ A\ \|^1\text{-}out\ (1+n)$. For the second one we instead use the following function:

$$\begin{array}{l} |\_,\_|\text{-}in : \forall\ n \to \|\ A\ \|^1\text{-}in\ n \to \|\ A\ \|^1\text{-}in\ (1+n) \\ |\ \mathsf{zero}\ \ ,\ x\ |\text{-}in = |\ x\ | \\ |\ \mathsf{suc}\ n\ ,\ x\ |\text{-}in = |\ n\ ,\ x\ |\text{-}in \end{array} \tag{73}$$

The two definitions are equivalent, and the equivalence relates $|\_|$ and $|\_,\_|\text{-}in$:

$$out{\simeq}in : \forall\ n \to \|\ A\ \|^1\text{-}out\ n \simeq \|\ A\ \|^1\text{-}in\ n \tag{74}$$

$$\begin{array}{l} to\ (out{\simeq}in\ (1+n))\ |\ x\ | \equiv \\ \quad |\ n\ ,\ to\ (out{\simeq}in\ n)\ x\ |\text{-}in \end{array} \tag{75}$$

Let us now introduce a coinductive definition which will be used to capture the notion of “coherently constant”. The type $Coherently\ P\ step\ f$ means that we have $p_0 : P\ f$, $p_1 : P\ (step\ f\ p_0)$, $p_2 : P\ (step\ (step\ f\ p_0)\ p_1)$, and so on:[1]

$$\begin{array}{l} \textbf{record}\ Coherently\ \{A : Type\ a\}\ \{B : Type\ b\} \\ \quad (P : \{A : Type\ a\} \to (A \to B) \to Type\ p) \\ \quad (step : \{A : Type\ a\}\ (f : A \to B) \to P\ f \to \\ \qquad\qquad \|\ A\ \|^1 \to B) \\ \quad (f : A \to B) : Type\ p\ \textbf{where} \\ \quad \textbf{coinductive} \\ \quad \textbf{field}\ \mathsf{property}\ : P\ f \\ \qquad\quad \mathsf{coherent} : Coherently\ P\ step\ (step\ f\ \mathsf{property}) \end{array} \tag{76}$$

We can define “coherently constant” in the following way, using rec (69):

$$\begin{array}{l} CC^{\mathrm{C}} : \{A : Type\ a\}\ \{B : Type\ b\} \to \\ \qquad (A \to B) \to Type\ (a \sqcup b) \\ CC^{\mathrm{C}} = Coherently\ Constant\ rec \end{array} \tag{77}$$

We will also use CC^{C1}, a variant of CC^{C}:

$$\begin{array}{l} Constant^1 : \{A : Type\ a\}\ \{B : Type\ b\} \to \\ \qquad\qquad (A \to B) \to Type\ (a \sqcup b) \\ Constant^1\ \{A = A\}\ \{B = B\}\ f = \\ \quad (g : \|\ A\ \|^1 \to B) \times \forall\ x \to g\ |\ x\ | \equiv f\ x \end{array} \tag{78}$$

$$\begin{array}{l} CC^{\mathrm{C1}} : \{A : Type\ a\}\ \{B : Type\ b\} \to \\ \qquad (A \to B) \to Type\ (a \sqcup b) \\ CC^{\mathrm{C1}} = Coherently\ Constant^1\ (\lambda\ \_\ (g\ ,\ \_) \to g) \end{array} \tag{79}$$

[1]An equivalent type can be defined without using coinduction, following Ahrens et al. [22], but if we do this in an “obvious” way, then the type’s universe is $Type\ (lsuc\ a \sqcup b \sqcup p)$ instead of $Type\ p$. If $Coherently$ was in this universe then the type $Lens^{\mathrm{C}}$ below (90) would not be small.

The predicate $Constant^1$ is a variant of CC that is defined using the one-step truncation operator. It is equivalent to $Constant$:

$$Constant\ f \simeq Constant^1\ f \tag{80}$$

This can be proved by defining functions in both directions, and proving that they are inverses of each other (in one case using the dependent elimination principle for the one-step truncation). We used this result (and univalence) to prove that CC^{C} and CC^{C1} are also equivalent:

$$CC^{\mathrm{C}}\ f \simeq CC^{\mathrm{C1}}\ f \tag{81}$$

Let us now prove the following result:

$$(\|\ A\ \| \to B)\ \simeq\ (f : A \to B) \times CC^{\mathrm{C}}\ f \tag{82}$$

We can calculate in the following way:

$$\begin{array}{ll} (\|\ A\ \| \to B) & \simeq \\ ((f : \forall\ n \to \|\ A\ \|^1\text{-}out\ n \to B)\ \times & \\ \quad \forall\ n\ x \to f\ (1+n)\ |\ x\ | \equiv f\ n\ x) & \simeq \\ ((f : \forall\ n \to \|\ A\ \|^1\text{-}in\ n \to B)\ \times & \\ \quad \forall\ n\ x \to f\ (1+n)\ |\ n\ ,\ x\ |\text{-}in \equiv f\ n\ x) & \simeq \\ ((f : A \to B)\ \times\ CC^{\mathrm{C1}}\ f) & \simeq \\ ((f : A \to B)\ \times\ CC^{\mathrm{C}}\ f) & \end{array}$$

The first step is Lemma 70, the last step follows from Lemma 81, and the second step can be proved using Lemmas 74 and 75, as well as the following preservation lemmas:

$$\begin{array}{l} (eq : A \simeq B) \to (\forall\ x \to P\ x \simeq Q\ (to\ eq\ x)) \to \\ \quad ((x : A) \times P\ x) \simeq ((x : B) \times Q\ x) \end{array} \tag{83}$$

$$\begin{array}{l} (eq : B \simeq A) \to (\forall\ x \to P\ (to\ eq\ x) \simeq Q\ x) \to \\ \quad ((x : A) \to P\ x) \simeq ((x : B) \to Q\ x) \end{array} \tag{84}$$

We proved the penultimate step by defining functions in both directions and proving that they are inverses. We found this to be quite a bit easier using the formulation with $\|\_\|^1\text{-}in$ and $|\_,\_|\text{-}in$ than the one with $\|\_\|^1\text{-}out$. As an aside we can also mention that we proved the following equivalence:

$$\begin{array}{l} CC^{\mathrm{C1}}\ f \simeq \\ \quad (g : \forall\ n \to \|\ A\ \|^1\text{-}in\ (1+n) \to B)\ \times \\ \quad (\forall\ x \to g\ 0\ |\ x\ | \equiv f\ x)\ \times \\ \quad (\forall\ n\ x \to g\ (1+n)\ |\ n\ ,\ x\ |\text{-}in \equiv g\ n\ x) \end{array} \tag{85}$$

The following is a consequence of Lemma 82:

$$CC\ f \simeq CC^{\mathrm{C}}\ f \tag{86}$$

We could use CC^{C} to define a coinductive notion of lens, but with a small change we end up with a definition that is small.

In the presence of univalence one can express weak constancy of *type-valued* functions in the following way:

$$\begin{array}{l} Constant^{\mathrm{S}} : \{A : Type\ a\} \to \\ \qquad\qquad (A \to Type\ p) \to Type\ (a \sqcup p) \\ Constant^{\mathrm{S}}\ P = \forall\ x\ y \to P\ x \simeq P\ y \end{array} \tag{87}$$

(“S” stands for “small”.) We use $Constant^{\mathrm{S}}$ to define a notion of coherent constancy for type-valued functions:

$$\begin{array}{l} CC^{\mathrm{S}} : \{A : \mathit{Type}\ a\} \to \\ \qquad (A \to \mathit{Type}\ p) \to \mathit{Type}\ (a \sqcup p) \\ CC^{\mathrm{S}} = \mathit{Coherently}\ \mathit{Constant}^{\mathrm{S}} \\ \qquad\qquad (\lambda\ f\ c \to \mathit{rec}\ f\ (\lambda\ x\ y \to \simeq\!\to\!\equiv\ (c\ x\ y))) \end{array} \tag{88}$$

Here $\simeq\!\to\!\equiv$, which is a consequence of univalence, has type $B \simeq C \to B \equiv C$. We get the following equivalence:

$$CC^{\mathrm{C}}\ f \simeq CC^{\mathrm{S}}\ f \tag{89}$$

We can thus define a *small* coinductive notion of higher lens:

$$\begin{array}{l} \mathit{Lens}^{\mathrm{C}} : \mathit{Type}\ a \to \mathit{Type}\ b \to \mathit{Type}\ (a \sqcup b) \\ \mathit{Lens}^{\mathrm{C}}\ A\ B = (\mathit{get} : A \to B) \times CC^{\mathrm{S}}\ (\mathit{get}\ ^{-1}\_) \end{array} \tag{90}$$

This variant is equivalent to the other ones:

$$\mathit{Lens}^{\mathrm{E}}\ A\ B \simeq \mathit{Lens}^{\mathrm{C}}\ A\ B \tag{91}$$

Because the family of fibres of the getter of this kind of lens is coherently constant we also get that it is weakly constant:

$$(b_1\ b_2 : B) \to \mathit{get}\ ^{-1}\ b_1 \simeq \mathit{get}\ ^{-1}\ b_2 \tag{92}$$

We can then define the setter in the same way as for $\mathit{Lens}^{\mathrm{F}}$ (66). We have proved that in our implementation the equivalence above (91) preserves getters and setters.

XIV. Unrestricted Composition

The coinductive formulation of lenses allows us to give an unrestricted implementation of composition, which works for types with arbitrary universe levels. The following function is a key building block:

$$\begin{array}{l} \{P : A \to \mathit{Type}\ p\}\ \{Q : B \to \mathit{Type}\ q\} \\ (f : B \to A) \to (\forall\ x \to P\ (f\ x) \simeq Q\ x) \to \\ CC^{\mathrm{S}}\ P \to CC^{\mathrm{S}}\ Q \end{array} \tag{93}$$

Note that P and Q are allowed to target different universes. Implementing a corresponding function directly for CC (46) seems to be hard, but we can do it for CC^{S}. The function is implemented using guarded corecursion and copatterns [23], and for the property field we can show that Q is weakly constant in the following way, given that P is:

$$Q\ x \quad \simeq \quad P\ (f\ x) \quad \simeq \quad P\ (f\ y) \quad \simeq \quad Q\ y$$

Another key building block is the following lemma:

$$\begin{array}{l} \{P : A \to \mathit{Type}\ p\} \\ \{Q : (x : A) \times P\ x \to \mathit{Type}\ q\} \to \\ CC^{\mathrm{S}}\ P \to CC^{\mathrm{S}}\ Q \to \\ CC^{\mathrm{S}}\ (\lambda\ x \to (y : P\ x) \times Q\ (x\ ,\ y)) \end{array} \tag{94}$$

This lemma is easy to prove for CC. However, we have proved it directly for CC^{S}, in order to get it to compute in a certain way.

Let us now implement composition:

$$\_\circ\_ : \mathit{Lens}^{\mathrm{C}}\ B\ C \to \mathit{Lens}^{\mathrm{C}}\ A\ B \to \mathit{Lens}^{\mathrm{C}}\ A\ C \tag{95}$$

Given $(\mathit{get}_1\ ,\ c_1) : \mathit{Lens}^{\mathrm{C}}\ B\ C$ and $(\mathit{get}_2\ ,\ c_2) : \mathit{Lens}^{\mathrm{C}}\ A\ B$ we define the resulting lens in the following way: The getter is the composition of the two getters. We use Lemma 93 and c_2 to construct a value of the following type:

$$CC^{\mathrm{S}}\ (\lambda\ ((\_\ ,\ b\ ,\ \_) : (c : C) \times \mathit{get}_1\ ^{-1}\ c) \to \mathit{get}_2\ ^{-1}\ b)$$

We then combine this value with c_2 using Lemma 94, obtaining a value of the following type:

$$CC^{\mathrm{S}}\ (\lambda\ c \to ((b\ ,\ \_) : \mathit{get}_1\ ^{-1}\ c) \times \mathit{get}_2\ ^{-1}\ b)$$

Finally we show that the resulting getter is coherently constant, $CC^{\mathrm{S}}\ ((\lambda\ a \to \mathit{get}_1\ (\mathit{get}_2\ a))\ ^{-1}\_)$, by using Lemma 93 and the fact that there is an equivalence between the types $(\lambda\ x \to f\ (g\ x))\ ^{-1}\ z$ and $((y\ ,\ \_) : f\ ^{-1}\ z) \times g\ ^{-1}\ y$. It is here that we make use of the fact that Lemma 93 holds for predicates that target possibly different universes.

With our implementation of the composition operator we get that the setter is implemented in the canonical way:

$$\mathit{set}\ (l_1 \circ l_2)\ a\ c \equiv \mathit{set}\ l_2\ a\ (\mathit{set}\ l_1\ (\mathit{get}\ l_2\ a)\ c) \tag{96}$$

(This equality holds by definition.) Using Lemma 41 it is then easy to prove that, if the view type of the resulting lens is stable, then the composition operator is associative, and it has a left and right unit.

We can also use this composition operator to implement an unrestricted composition operator for $\mathit{Lens}^{\mathrm{E}}$, and due to Lemmas 96 and 60 we get that it matches the other implementation of composition (59) when all types have the same universe level and the view type of the resulting lens is stable.

XV. Homotopy Levels, Continued

Let us now investigate h-levels of higher lens types. The following variant of *Coherently* will be used as a proof device:

$$\begin{array}{l} \textbf{record}\ \mathit{Coherently}'\ \{A : \mathit{Type}\ a\}\ \{B : \mathit{Type}\ b\} \\ \quad (P : \{A : \mathit{Type}\ a\} \to (A \to B) \to \mathit{Type}\ p) \\ \quad (\mathit{step} : \{A : \mathit{Type}\ a\}\ (f : A \to B) \to P\ f \to \\ \qquad\qquad \|\ A\ \|^1 \to B) \\ \quad (f : A \to B) \\ \quad (Q : \{A : \mathit{Type}\ a\} \to (A \to B) \to \mathit{Type}\ q) \\ \quad (\mathit{pres} : \{A : \mathit{Type}\ a\}\ \{f : A \to B\}\ \{p : P\ f\} \to \\ \qquad\qquad Q\ f \to Q\ (\mathit{step}\ f\ p)) \\ \quad (q : Q\ f) : \mathit{Type}\ p\ \textbf{where} \\ \quad \textbf{coinductive} \\ \quad \textbf{field}\ \mathsf{property}\ : P\ f \\ \qquad\quad \mathsf{coherent} : \mathit{Coherently}'\ P\ \mathit{step}\ (\mathit{step}\ f\ \mathsf{property}) \\ \qquad\qquad\qquad\qquad Q\ \mathit{pres}\ (\mathit{pres}\ q) \end{array} \tag{97}$$

Note that the property field is unchanged, but that we ensure that the predicate Q holds for all the functions f. The following equivalence is easy to prove:

$$\begin{array}{l} \mathit{Coherently}\ P\ \mathit{step}\ f \simeq \\ \mathit{Coherently}'\ P\ \mathit{step}\ f\ Q\ \mathit{pres}\ q \end{array} \tag{98}$$

One can express the right-hand side of this equivalence as an indexed M-type for the indexed container (of the kind described by Ahrens et al. [22]) where the index type is $(A : \mathit{Type}\ a) \times (f : A \to B) \times Q\ f$, the shape for the index $(\_\ ,\ f\ ,\ \_)$ is $P\ f$, the positions are trivial, and the "next" index for the index $(A\ ,\ f\ ,\ q)$ and the shape p is

$(\| A \|^1 , \textit{step } f\ p , \textit{pres } q)$. The h-level of such an M-type is n if all the shapes have h-level n [22], so we get the following lemma:

$$\begin{array}{l}(\{A : \textit{Type } a\}\ \{f : A \to B\} \to \\ \quad Q\ f \to \textit{H-level } n\ (P\ f)) \to \\ \textit{H-level } n\ (\textit{Coherently}'\ P\ \textit{step } f\ Q\ \textit{pres } q)\end{array} \tag{99}$$

Using this lemma we can prove the following result:

$$\begin{array}{l}\{P : \{A : \textit{Type } a\} \to (A \to \textit{Type } f) \to \textit{Type } p\} \\ \{\textit{step} : \{A : \textit{Type } a\}\ (F : A \to \textit{Type } f) \to P\ F \to \\ \qquad \| A \|^1 \to \textit{Type } f\} \\ \{F : A \to \textit{Type } f\} \\ (h_1 : (a : A) \to \textit{H-level } n\ (F\ a)) \\ (h_2 : \{A : \textit{Type } a\}\ \{F : A \to \textit{Type } f\} \to \\ \qquad ((a : A) \to \textit{H-level } n\ (F\ a)) \to \\ \qquad \textit{H-level } n\ (P\ F)) \\ (h_3 : \{A : \textit{Type } a\}\ \{F : A \to \textit{Type } f\}\ \{p : P\ F\} \to \\ \qquad ((a : A) \to \textit{H-level } n\ (F\ a)) \to \\ \qquad (a : A) \to \textit{H-level } n\ (\textit{step } F\ p \mid a \mid)) \to \\ \textit{H-level } n\ (\textit{Coherently } P\ \textit{step } F)\end{array} \tag{100}$$

One can use the eliminator for the one-step truncation to construct a variant of h_3:

$$\begin{array}{l}h_3' : \{A : \textit{Type } a\}\ \{F : A \to \textit{Type } f\}\ \{p : P\ F\} \to \\ \qquad ((a : A) \to \textit{H-level } n\ (F\ a)) \to \\ \qquad (a : \| A \|^1) \to \textit{H-level } n\ (\textit{step } F\ p\ a)\end{array}$$

Due to Lemma 98 it then suffices to prove the result for the following type, which is easy using Lemma 99 and h_2:

$$\begin{array}{l}\textit{Coherently}'\ P\ \textit{step } F \\ \quad (\lambda\ F \to \forall\ a \to \textit{H-level } n\ (F\ a))\ h_3'\ h_1\end{array}$$

Using Lemma 100 and standard properties about h-levels [4] we get that $CC^{\mathrm{S}}\ P$ has h-level n if P has this h-level (pointwise):

$$(\forall\ a \to \textit{H-level } n\ (P\ a)) \to \textit{H-level } n\ (CC^{\mathrm{S}}\ P) \tag{101}$$

We can now prove that $\textit{Lens}^{\mathrm{C}}\ A\ B$ has h-level n if A and B have that h-level given that the other type is inhabited:

$$\begin{array}{l}(B \to \textit{H-level } n\ A) \to (A \to \textit{H-level } n\ B) \to \\ \textit{H-level } n\ (\textit{Lens}^{\mathrm{C}}\ A\ B)\end{array} \tag{102}$$

This follows from Lemma 101. As a consequence we get that if A has positive h-level n, then the type of higher lenses from A to B has h-level n:

$$\begin{array}{l}\textit{H-level } (1 + n)\ A \to \\ \textit{H-level } (1 + n)\ (\textit{Lens}^{\mathrm{C}}\ A\ B)\end{array} \tag{103}$$

When proving that a type has a positive h-level one can assume that the type is inhabited [4], so we can assume that we have a value of type $\textit{Lens}^{\mathrm{C}}\ A\ B$. Now Lemma 48 tells us that B has h-level $1 + n$ if A is inhabited, so we can conclude by using Lemma 102.

As an aside we can mention that Lemma 102 (with the first "B" replaced by "A") and Lemma 103 hold also for traditional lenses. The proofs are straightforward. See the accompanying code for details.

XVI. Related Work

After Foster et al. [1] introduced lenses they became popular in the functional programming community, partly because they provide a composable approach to working with deeply nested immutable structures, as discussed in the introduction.

In the functional programming community, and especially the Haskell one with its lens library [24], lenses usually occur as special cases of a more general abstraction called an *optic*, including for example *traversals* [25] and *prisms* [3], and often expressed via *impredicative* formulations, like the Van Laarhoven representation [7] mentioned earlier.

Lenses have been characterised as coalgebras [26], [27], algebras [14], and internal functors and cofunctors [28], and they have been used in linguistics [29] and quantum computing [30]. The theory of lenses also goes back to before their inception: Foster et al. [1] state that "our set of very well behaved lenses is isomorphic to the set of *translators under constant complement*", where the latter notion is due to Bancilhon and Spyratos [2].

Despite the great interest that lenses have generated we are not aware of any other work on framing the concept of a lens within the context of homotopy type theory, with one exception: the work of Grenrus [31]. Grenrus discusses both lenses and prisms, but the study of lenses seems to be preliminary, there are few results. Grenrus criticises our higher lenses: he presents two lenses that have equal setters, and states that they are "observably different"; he seems to suggest that the two lenses are not equal. The view type of the lenses is the type of booleans, and because the type of booleans is inhabited we know that these lenses are equal (40). However, Grenrus' statement is not wrong. How can this be?

Let us take a look at an example which is closely based on that presented by Grenrus. We define a function that constructs lenses, given a family of equivalences:

$$\begin{array}{l}(\textit{Bool} \to \textit{Bool} \simeq \textit{Bool}) \to \\ \textit{Lens}^{\mathrm{E}}\ (\textit{Bool} \times \textit{Bool})\ \textit{Bool}\end{array} \tag{104}$$

The remainder type of the constructed lens is $\textit{Bool}$. The equivalence between $\textit{Bool} \times \textit{Bool}$ and $\textit{Bool} \times \textit{Bool}$ is constructed by letting the second component be unchanged, and using the value of the second component to decide which of the two equivalences should be used for the first component. We get the two lenses of the example by instantiating the family in two different ways: in one case we use the identity equivalence for true and the *not* function for false, and in the other case we do it the other way around. It is easy to prove that the lenses that we obtain in this way, ex_1 and ex_2, have equal setters, and thus we get that they are equal. However, the lenses *are* in a sense observably different; the following equalities hold by definition:

$$\textit{remainder}\ ex_1\ (\mathsf{true}\ ,\ \mathsf{true}) \equiv \mathsf{true} \tag{105}$$
$$\textit{remainder}\ ex_2\ (\mathsf{true}\ ,\ \mathsf{true}) \equiv \mathsf{false} \tag{106}$$

We do not find this any more surprising than the fact [32] that one can have two values of the contractible type $(A : \textit{Type}) \times (A \simeq \textit{Bool})$ that are equal (assuming univalence all values of this type are equal), but for which the second projections are distinct (the identity equivalence and the not function). Note that one cannot use "$cong$ proj_2" to conclude that the identity equivalence and the not function are equal, because the second projection proj_2 is in this case not a non-dependent function. The $remainder$ function is also not non-dependent, it has type $(l : Lens^{\mathsf{E}}\ A\ B) \to A \to \mathsf{R}\ l$.

XVII. Conclusion

We have presented several equivalent formulations of higher lenses, explored the properties of the definitions, and tried to show that higher lenses are better behaved than traditional ones in the setting of homotopy type theory.

We have focused on very well-behaved lenses, ignoring all the other optics, and also ignoring alternative representations like the one due to Van Laarhoven [7]. An obvious avenue for further work would be to try to generalise the results to, say, traversals.

Acknowledgements

We would like to thank the anonymous reviewers, as well as Christian Sattler, who proved Lemma 47.

Danielsson has been supported by a grant from the Swedish Research Council (621-2013-4879). Vezzosi was supported by a research grant (13156) from VILLUM FONDEN.

References

[1] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt, "Combinators for bi-directional tree transformations: A linguistic approach to the view update problem," in *POPL® 2005: The 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages®*, 2005, pp. 233–246. DOI:10.1145/1040305.1040325

[2] F. Bancilhon and N. Spyratos, "Update semantics of relational views," *ACM Transactions on Database Systems*, vol. 6, no. 4, pp. 557–575, 1981. DOI:10.1145/319628.319634

[3] M. Pickering, J. Gibbons, and N. Wu, "Profunctor optics: Modular data accessors," *The Art, Science, and Engineering of Programming*, vol. 1, no. 2, 2017. DOI:10.22152/programming-journal.org/2017/1/7

[4] The Univalent Foundations Program, *Homotopy Type Theory: Univalent Foundations of Mathematics*, 1st ed., 2013. [Online]. Available: https://homotopytypetheory.org/book/

[5] A. Vezzosi, A. Mörtberg, and A. Abel, "Cubical Agda: A dependently typed programming language with univalence and higher inductive types," *Proceedings of the ACM on Programming Languages*, vol. 3, no. ICFP, pp. 87:1–87:29, 2019. DOI:10.1145/3341691

[6] N. A. Danielsson, "Code related to the paper "Higher Lenses"," 2021. DOI:10.5281/zenodo.4727911

[7] T. van Laarhoven, "CPS based functional references," blog post, 2009. [Online]. Available: https://www.twanvl.nl/blog/haskell/cps-functional-references

[8] ——, "Isomorphism lenses," blog post, 2011. [Online]. Available: https://www.twanvl.nl/blog/haskell/isomorphism-lenses

[9] C. McBride, "I got plenty o' nuttin'," in *A List of Successes That Can Change the World: Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*, 2016, pp. 207–233. DOI:10.1007/978-3-319-30936-1_12

[10] R. Atkey, "Syntax and semantics of quantitative type theory," in *LICS '18 Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, 2018, pp. 56–65. DOI:10.1145/3209108.3209189

[11] N. A. Danielsson, "Dependent lenses," unpublished, 2016. [Online]. Available: http://www.cse.chalmers.se/~nad/publications/danielsson-dependent-lenses.html

[12] D. R. Licata and G. Brunerie, "$\pi_n(S^n)$ in homotopy type theory," in *Certified Programs and Proofs, Third International Conference, CPP 2013*, 2013, pp. 1–16. DOI:10.1007/978-3-319-03545-1_1

[13] F. van Doorn, "Constructing the propositional truncation using non-recursive HITs," in *CPP'16, Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, 2016, pp. 122–129. DOI:10.1145/2854065.2854076

[14] M. Johnson, R. Rosebrugh, and R. Wood, "Algebras and update strategies," *Journal of Universal Computer Science*, vol. 16, no. 5, pp. 729–748, 2010. DOI:10.3217/jucs-016-05-0729

[15] B. C. Pierce and A. Schmitt, "Lenses and view update translation," unpublished, 2003. [Online]. Available: http://www.cis.upenn.edu/~bcpierce/papers/dblenses.pdf

[16] N. Kraus, M. H. Escardó, T. Coquand, and T. Altenkirch, "Notions of anonymous existence in Martin-Löf type theory," *Logical Methods in Computer Science*, vol. 13, no. 1, pp. 1–36, 2017. DOI:10.23638/LMCS-13(1:15)2017

[17] M. Shulman, "Not every weakly constant function is conditionally constant," blog post, 2015. [Online]. Available: https://homotopytypetheory.org/2015/06/11/not-every-weakly-constant-function-is-conditionally-constant/

[18] D. R. Licata and E. Finster, "Eilenberg-MacLane spaces in homotopy type theory," in *CSL-LICS '14: Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2014. DOI:10.1145/2603088.2603153

[19] N. Kraus, "The general universal property of the propositional truncation," in *20th International Conference on Types for Proofs and Programs, TYPES'14*, 2015, pp. 111–145. DOI:10.4230/LIPIcs.TYPES.2014.111

[20] ——, "Constructions with non-recursive higher inductive types," in *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2016)*, 2016, pp. 595–604. DOI:10.1145/2933575.2933586

[21] E. Rijke, "The join construction," 2017, arXiv: 1701.07538v1 [math.CT].

[22] B. Ahrens, P. Capriotti, and R. Spadotti, "Non-wellfounded trees in homotopy type theory," in *13th International Conference on Typed Lambda Calculi and Applications, TLCA'15*, 2015, pp. 17–30. DOI:10.4230/LIPIcs.TLCA.2015.17

[23] A. Abel, B. Pientka, D. Thibodeau, and A. Setzer, "Copatterns: Programming infinite structures by observations," in *POPL '13, Proceedings of 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2013, pp. 27–38. DOI:10.1145/2429069.2429075

[24] E. Kmett *et al.*, "Lens: Lenses, folds, and traversals," 2020. [Online]. Available: https://github.com/ekmett/lens/

[25] J. Gibbons and B. C. d. S. Oliveira, "The essence of the iterator pattern," *Journal of Functional Programming*, vol. 19, no. 3 & 4, pp. 377–402, 2009. DOI:10.1017/S0956796809007291

[26] R. O'Connor, "Functor is to lens as applicative is to biplate: Introducing multiplate," 2011, arXiv: 1103.2841v2 [cs.PL].

[27] D. Ahman and T. Uustalu, "Coalgebraic update lenses," in *Proceedings of the 30th Conference on the Mathematical Foundations of Programming Semantics (MFPS XXX)*, 2014, pp. 25–48. DOI:10.1016/j.entcs.2014.10.003

[28] B. Clarke, "Internal lenses as functors and cofunctors," in *Proceedings Applied Category Theory 2019*, 2020, pp. 183–195. DOI:10.4204/EPTCS.323.13

[29] J. Hefford, V. Wang, and M. Wilson, "Categories of semantic concepts," in *Semantic Spaces at the Intersection of NLP, Physics, and Cognitive Science*, 2020. [Online]. Available: https://sites.google.com/view/semspace2020/programme

[30] M. Wilson, J. Hefford, G. Boisseau, and V. Wang, "The safari of update structures: Visiting the lens and quantum enclosures," in *Proceedings of the 3rd Annual International Applied Category Theory Conference 2020*, 2021, pp. 1–18. DOI:10.4204/EPTCS.333.1

[31] O. Grenrus, "Shattered lens," extended abstract for the Eighth Workshop on Mathematically Structured Functional Programming (MSFP 2020), 2020. [Online]. Available: https://msfp-workshop.github.io/msfp2020/grenrus.pdf

[32] J. Gross, "Composition is not what you think it is! Why "nearly invertible" isn't." Blog post, 2014. [Online]. Available: https://homotopytypetheory.org/2014/02/24/composition-is-not-what-you-think-it-is-why-nearly-invertible-isnt/