To appear at POPL 2017

Contextual Isomorphisms

Paul Blain Levy
University of Birmingham, UK
P.B.Levy@cs.bham.ac.uk

Abstract

What is the right notion of "isomorphism" between types, in a simple type theory? The traditional answer is: a pair of terms that are inverse, up to a specified congruence. We firstly argue that, in the presence of effects, this answer is too liberal and needs to be restricted, using Führmann's notion of thunkability in the case of value types (as in call-by-value), or using Munch-Maccagnoni's notion of linearity in the case of computation types (as in call-by-name). Yet that leaves us with different notions of isomorphism for different kinds of type.

This situation is resolved by means of a new notion of "contextual" isomorphism (or morphism), analogous at the level of types to contextual equivalence of terms. A contextual morphism is a way of replacing one type with the other wherever it may occur in a judgement, in a way that is preserved by the action of any term with holes. For types of pure λ-calculus, we show that a contextual morphism corresponds to a traditional isomorphism. For value types, a contextual morphism corresponds to a thunkable isomorphism, and for computation types, to a linear isomorphism.

Categories and Subject Descriptors F.3.2 [*Semantics of Programming Languages*]

Keywords isomorphism, contextual equivalence, computational effects, call-by-push-value

1. Introduction

1.1 Question

Suppose we have a typed programming language, equipped with a suitable congruence $\equiv$. When can we treat two types A and B as essentially the same?

In Sections 1.2 and 1.3 we consider two answers to this question that on the face of it are entirely different. Yet as we explain in Section 1.4, they amount to the same thing.

1.2 Map-Based Isomorphisms

Let us begin with the traditional answer to this question (Di Cosmo 1995; Fiore et al. 2002; Soloviev and Di Cosmo 2005): A and B are essentially the same when there is a pair of maps $f : A \to B$ and $g : B \to A$ such that $f; g \equiv \mathsf{id}_A$ and $g; f \equiv \mathsf{id}_B$. We call this a "map-based" isomorphism.

```
x : bool ⊢
  case x of {
    true.
      read l as v.
      l := 2 ∗ v.
      sole
    false.
      read l as v.
      l := 2 ∗ v + 1.
      sole
  }          : unit
```

```
x : unit ⊢
  read l as u.
  case u of {
    2 ∗ w.
      l := w.
      true
    2 ∗ w + 1.
      l := w.
      false
  }          : bool
```

Figure 1. Isomorphism `bool` $\cong$ `unit` in a call-by-value language with mutable state

Although this answer works well for a purely functional programming language, it is too liberal for a language that includes computational effects, such as mutable state or exceptions. We look at two examples.

1. Consider a call-by-value language with a memory location `l` storing a natural number. Then the types `bool` and `unit`—ground types with two values and one value respectively—are isomorphic (Fig. 1). This may seem surprising, but can be understood using denotational semantics. Each type A denotes a set $[\![A]\!]$. A ground type with I values denotes a set with I elements. A map $A \to B$ denotes not a function $[\![A]\!] \to [\![B]\!]$, as it would if the language were pure, but rather a function $\mathbb{N} \times [\![A]\!] \to \mathbb{N} \times [\![B]\!]$, indicating the initial and final state associated with a function call. Fig. 1 denotes the bijection
$$\begin{array}{rrcl} & \mathbb{N} \times [\![\mathtt{bool}]\!] & \cong & \mathbb{N} \times [\![\mathtt{unit}]\!] \\ \text{i.e.} & \mathbb{N} \times \{\mathsf{true}, \mathsf{false}\} & \cong & \mathbb{N} \times \{\mathsf{sole}\} \\ & (k, \mathsf{true}) & \mapsto & (2k, \mathsf{sole}) \\ & (k, \mathsf{false}) & \mapsto & (2k+1, \mathsf{sole}) \end{array}$$

2. Consider a call-by-name language where an exception `e`, carrying a natural number value, can be raised and handled. Then the types `unit` and `empty`—ground types with one value and no values respectively—are isomorphic (Fig. 2). For convenience we combine case-analysis and exception handling in a single `try`–`case` construct, cf. (Benton and Kennedy 2001). Again, this isomorphism may seem surprising but can be understood using denotational semantics. Each type denotes a $\mathbb{N}$-*pointed set*, i.e. set X with a sequence of distinguished elements $(c_n)_{n \in \mathbb{N}}$ representing the exceptional behaviours. A ground type with I values denotes $(I + \mathbb{N}, (\mathsf{inr}\ n)_{n \in \mathbb{N}})$. A map $A \to B$ denotes

```
x : unit ⊢
  try–case x of {
    sole.
      raise (e, 0)
    catch (e, v).
      raise (e, v+1)
  }           : empty
```

```
x : empty ⊢
  try–case x of {
    catch (e, u).
      case u of {
        0.
          sole
        w+1.
          raise (e, w)
      }
  }           : unit
```

Figure 2. Isomorphism unit ≅ empty in a call-by-name language with exceptions

a function $U[\![A]\!] \to U[\![B]\!]$, where we write U for the carrier (underlying set) of a $\mathbb{N}$-pointed set. Fig. 2 denotes the bijection

$$\begin{array}{rrcl} & U[\![\mathtt{unit}]\!] & \cong & U[\![\mathtt{empty}]\!] \\ \text{i.e.} & \{\mathsf{sole}\} + \mathbb{N} & \cong & \emptyset + \mathbb{N} \\ & \mathsf{inl\ sole} & \mapsto & \mathsf{inr}\ 0 \\ & \mathsf{inr}\ n & \mapsto & \mathsf{inr}\ n+1 \end{array}$$

Let us consider these examples more carefully.

In the call-by-value setting (1), computations are interpreted in the Kleisli category for the state monad $\mathbb{N} \to (\mathbb{N} \times -)$ on **Set** (Moggi 1991). Our isomorphism bool ≅ unit reflects the fact that $[\![\mathtt{bool}]\!]$ and $[\![\mathtt{unit}]\!]$ are isomorphic in the Kleisli category, although they are not isomorphic sets. Intuitively, this makes it a bad isomorphism.

Principle 1. *A "good" isomorphism of call-by-value types should denote an isomorphism of sets—not merely an isomorphism in the Kleisli category.*

In the call-by-name setting (2), types denote algebras. (In this example, algebras for a signature consisting of $\mathbb{N}$ many constants.) Our isomorphism unit ≅ empty reflects the fact that $[\![\mathtt{unit}]\!]$ and $[\![\mathtt{empty}]\!]$ have isomorphic carriers, although they are not isomorphic algebras. Intuitively, this makes it a bad isomorphism.

Principle 2. *A "good" isomorphism of call-by-name types should denote an isomorphism of algebras—not merely an isomorphism of carriers.*

We can obtain a more fine-grained picture of these issues in *call-by-push-value* (Levy 2004). This is a calculus that has both *value types*, which correspond to call-by-value types and denote sets, and *computation types* (usually underlined), which correspond to call-by-name types and denote algebras. In particular the computation type FA denotes the free algebra on $[\![A]\!]$, and the value type $U\underline{B}$ denotes the carrier of the algebra $[\![\underline{B}]\!]$. Following the above principles, we distinguish between $A \cong B$ and $FA \cong FB$, and between $\underline{A} \cong \underline{B}$ and $U\underline{A} \cong U\underline{B}$.

To sum up: *the common notion of isomorphism of sets or of algebras should be carried over to types that denote these things*. However, this does not answer the question in Section 1.1. The language there was not equipped with a denotational semantics. It was merely equipped with a congruence $\equiv$. So we need to formulate "goodness" of an isomorphism in terms of $\equiv$.

- For value types (or call-by-value types), we want to say that a good isomorphism performs no effects, and for this we use Führmann's notion of *thunkability* (Führmann 1999). This can be formulated in call-by-push-value, and also in call-by-value languages provided they include nullary function types. A composite of thunkable maps is thunkable, as is the inverse of a thunkable map if it exists.
- For computation types (or call-by-name types), we want to say that a good isomorphism preserves algebra structure, and for this we use Munch-Maccagnoni's notion of *linearity* (Munch-Maccagnoni 2014). This can be formulated in call-by-push-value, and also in call-by-name languages provided they include unary sum types. The composite of linear maps is linear, as is the inverse of a linear map if it exists.

We conclude: *value types should be considered essentially the same when they are related by a thunkable isomorphism, and computation types when they are related by a linear isomorphism*. But this means that, unhappily, we have three different notions of map-based isomorphism (unrestricted, thunkable, linear) appropriate in different settings (pure language, value types, computation types). We would prefer a single notion that is appropriate in all settings.

1.3 Contextual Morphisms and Isomorphisms

Leaving the previous section aside, we shall give in this section an entirely different answer to our question, one that makes no mention of a map $A \to B$ or a map $B \to A$.

Consider the corresponding problem for terms. When should two terms (of the same type) be regarded as essentially the same? A well-known answer (Morris, Jr. 1968) is the notion of *contextual equivalence*, also called *observational equivalence*. It is based on the premise that we can recognize the "observable behaviour" of a *program*, a closed term of ground type. A pair of terms M and M' are contextually equivalent when, for any *program with a hole* $\mathcal{N}[\cdot]$, the programs $\mathcal{N}[M]$ and $\mathcal{N}[M']$ have the same observable behaviour.

We tackle our question in a similar spirit. Instead of a notion of "observable behaviour", we have a congruence $\equiv$. (This congruence might itself be contextual equivalence, but that is irrelevant.) If types A and B are "essentially the same", then surely any $\equiv$-class of terms

$$\mathtt{x} : \mathtt{bool} + (\mathtt{nat} \to A), \mathtt{y} : A \to A \vdash M : A \to \mathtt{nat}$$

should correspond to an $\equiv$-class of terms

$$\mathtt{x} : \mathtt{bool} + (\mathtt{nat} \to B), \mathtt{y} : B \to B \vdash M : B \to \mathtt{nat}$$

So, considering the *judgement with a type-hole*[1]

$$\mathtt{x} : \mathtt{bool} + (\mathtt{nat} \to [\cdot\cdot]), \mathtt{y} : [\cdot\cdot] \to [\cdot\cdot] \vdash [\cdot\cdot] \to \mathtt{nat}$$

it does not matter whether $[\cdot\cdot]$ is filled with A or B. In general, each $\equiv$-class of terms $\Gamma[A] \vdash M : C[A]$ should correspond to an $\equiv$-class of terms $\Gamma[B] \vdash \theta(M) : C[B]$.

So we want a bijection θ on $\equiv$-classes of terms, for each judgement with a type-hole. But that is not enough; we also want each term constructor of the language, such as application or λ-abstraction, to *preserve* θ. Saying that application preserves θ means that for any terms

$$\Gamma[A] \vdash M : C[A] \to D[A] \quad \Gamma[A] \vdash N : C[A]$$

we have

$$(\theta(M))(\theta(N)) \equiv \theta(MN)$$

Similarly, saying that λ-abstraction preserves θ means that for any term

$$\Gamma[A], \mathtt{x} : C[A] \vdash M : D[A]$$

we have

$$\lambda\mathtt{x}.\theta(M) \equiv \theta(\lambda\mathtt{x}.M)$$

[1] Because our type syntax does not include any binding, the distinction between type-holes and type identifiers is immaterial.

More generally, a term with holes $\mathcal{N}[\cdot_h]_{h\in H}$ preserves θ when

$$\mathcal{N}[\theta(M_h)]_{h\in H} = \theta(\mathcal{N}[M_h]_{h\in H})$$

(The precise typing assumptions will be given in Section 2.) Clearly if every term constructor preserves θ, then, by induction, so does every term with holes. This requirement can be understood as saying that θ is "invisible" to the syntax.

To summarize: a *contextual isomorphism* $A \cong B$ provides, for each judgement with a hole, a bijection on $\equiv$-classes of terms, preserved by every term constructor or, equivalently, by every term with holes. If, rather than a bijection, we have merely a *function* sending each $\equiv$-class of terms $\Gamma[A] \vdash M : C[A]$ to an $\equiv$-class of terms $\Gamma[B] \vdash \theta(M) : C[B]$, we call that a *contextual morphism*.

As we shall see in Section 8–9, there is no contextual isomorphism $\mathtt{bool} \cong \mathtt{unit}$ in call-by-value with state, nor $\mathtt{unit} \cong \mathtt{empty}$ in call-by-name with exceptions.

1.4 Map-based vs Contextual Isomorphisms

Map-based and contextual isomorphism appear to be very different notions, but it turns out that in many settings they correspond.

1. In pure λ-calculi, under reasonable assumptions on $\equiv$, contextual morphisms correspond to unrestricted map-based isomorphisms.
2. In call-by-push-value calculi, under reasonable assumptions on $\equiv$, contextual morphisms between value types correspond to thunkable isomorphisms, and those between computation types correspond to linear isomorphisms.
3. In effectful λ-calculi that include nullary function types, under assumptions on $\equiv$ that are reasonable for call-by-value semantics, contextual morphisms correspond to thunkable isomorphisms.
4. In effectful λ-calculi that include unary sum types, under assumptions on $\equiv$ that are reasonable for call-by-name semantics, contextual morphisms correspond to linear isomorphisms.

Each of these statements implies that every contextual morphism is an isomorphism, a surprising fact in itself which does not hold in all languages[2].

The importance of the above correspondences is that each notion of isomorphism—contextual and map-based—has an advantage over the other. On the one hand, contextual isomorphism provides a single, *a priori* notion of essential sameness that is appropriate for any typed calculus equipped with a congruence, unlike map-based isomorphism, which as mentioned above has a different variant (unrestricted, thunkable or linear) for different settings. On the other hand, it is too complicated to present a isomorphism $A \cong B$ in contextual form, for the same reason that it is difficult to prove a contextual equivalence, viz. the quantification over all contexts. Instead we may present the corresponding map-based isomorphism, for which we need only exhibit two terms and check that they are mutually inverse and, in the effectful setting, thunkable or linear.

1.5 Structure of Paper

Whereas the above introduction discussed map-based isomorphism before contextual morphism, the paper will treat contextual morphism first since it is the main contribution. Although this work is applicable to all kinds of effects, we highlight state and exceptions since they illustrate well the distinction between good and bad isomorphisms.

For λ-calculus, whether pure or effectful, we define

- contextual morphism in Section 2;
- map-based isomorphism in Section 3.

We then study

- pure λ-calculus (Section 4)
- call-by-push-value (Section 5–8)
- call-by-value and call-by-name (Section 9).

In each case we present the correspondence between contextual morphisms and map-based isomorphisms. We give examples and non-examples in Sections 8–9.

1.6 Related work

Type isomorphisms for general references have been studied in (Clairambault 2011), for classical logic in (Laurent 2005; Laurent et al. 2005) and for polarized systems in (Munch-Maccagnoni 2013)[Section IV.5.3]. Type isomorphisms have been studied in the setting of game semantics in (Clairambault 2011; de Lataillade 2008; Laurent 2005). Functoriality in thunkable maps has appeared in (Führmann 1999; Selinger 2001). A connection between parametricity in isomorphisms and a result that every morphism is an isomorphism is explored in (Freyd et al. 1992; Robinson 1994)

1.7 Conventions

In a category $\mathcal{C}$, an *isomorphism* is a morphism with a (necessarily unique) inverse. The isomorphisms of $\mathcal{C}$ form a groupoid, written $\mathbf{Isos}\,(\mathcal{C})$, e.g. $\mathbf{Isos}\,(\mathbf{Set})$ is the groupoid of sets and bijections.

On a typed calculus, a *congruence* is an equivalence on typed terms in context, preserved by every term constructor and by weakening, i.e. if $\Gamma \subseteq \Gamma'$ then $\Gamma \vdash M \equiv N : C$ implies $\Gamma' \vdash M \equiv N : C$. We write $\mathcal{Q}^{\equiv}(\Gamma \vdash C)$ for the set of $\equiv$-classes of terms $\Gamma \vdash M : C$. A particular $\equiv$-class is written $(\Gamma \vdash M : C)^{\equiv}$.

2. Contextual Equivalence and Morphisms

We first reprise the notion of contextual equivalence, taking care to spell out the typing assumptions involved, and then proceed by analogy to define contextual morphism and isomorphism.

These are general definitions that make sense in any typed calculus with term judgement $\Gamma \vdash M : C$.

2.1 Contextual Equivalence

Contextual equivalence involves the notion of a *term with a hole* (also known as a *context*), given by the typing judgement

$$\mathcal{N} \;:\; (\Gamma \vdash C) \longrightarrow (\Delta \vdash D)$$

which says that $\mathcal{N}$ is a term in context Δ of type D that may contain occurrences of a hole $[\cdot]$ to be replaced by a term $\Gamma \vdash M : C$. More generally, we can consider terms with several holes: the typing judgement

$$\mathcal{N} \;:\; (\Gamma_h \vdash C_h)_{h\in H} \longrightarrow (\Delta \vdash D) \qquad (1)$$

says that $\mathcal{N}$ is a term in context Δ of type D that may contain occurrences of a hole $[\cdot_h]$, to be replaced by a term $\Gamma_h \vdash M_h : C_h$, for each $h \in H$. The typing rules for terms with holes are the same as those for $\vdash$ e.g.

$$\frac{\mathcal{N} \;:\; (\Gamma_h \vdash C_h)_{h\in H} \longrightarrow (\Delta, \mathtt{x} : B \vdash D)}{\lambda\mathtt{x}.\mathcal{N} \;:\; (\Gamma_h \vdash C_h)_{h\in H} \longrightarrow (\Delta \vdash B \to D)}$$

$$\frac{}{\mathtt{x} \;:\; (\Gamma_h \vdash C_h)_{h\in H} \longrightarrow (\Delta \vdash B)}\;(\mathtt{x} : B) \in \Delta$$

with the additional axiom

$$\frac{}{[\cdot_h] \;:\; (\Gamma_h \vdash C_h)_{h\in H} \longrightarrow (\Gamma' \vdash C_h)}\;h \in H, \Gamma_h \subseteq \Gamma'$$

[2] For example, in a pure language of arithmetic with no identifiers and only ground types, there would be a unique contextual morphism $\mathtt{empty} \to \mathtt{unit}$ but none in the opposite direction.

As explained in (Pitts 1997), we do not identify terms with holes that are α-equivalent, e.g. $\lambda\mathtt{x}.[\cdot]$ and $\lambda\mathtt{y}.[\cdot]$ cannot be identified.

For a term with holes

$$\mathcal{N} \ : \ (\Gamma_h \vdash C_h)_{h \in H} \longrightarrow (\Delta \vdash D)$$

we obtain from any family of terms $\mathbf{M} = (\Gamma_h \vdash M_h : C_h)_{h \in H}$ a term $\Delta \vdash \mathcal{N}[\mathbf{M}] : D$, by induction on $\mathcal{N}$ in the evident way. It is M_h, using weakening, in the case $\mathcal{N} = [\cdot_h]$. Likewise, for any congruence $\equiv$, we obtain a function $\mathcal{N}^{\equiv}$

$$\begin{array}{rcc} \prod_{h \in H} \mathcal{Q}^{\equiv}(\Gamma_h \vdash C_h) & \to & \mathcal{Q}^{\equiv}(\Delta \vdash D) \\ (\Gamma_h \vdash M_h : C_h)^{\equiv}_{h \in H} & \mapsto & (\Delta \vdash \mathcal{N}[M_h]_{h \in H} : D)^{\equiv} \end{array}$$

We can now reprise the definition of contextual equivalence. Assume a set of *ground* types, and for each ground type D

- a set $\langle\!\langle D \rangle\!\rangle$ of *meanings*
- for each term $\vdash M : D$ (called a *program*) a meaning $\langle\!\langle M \rangle\!\rangle \in \langle\!\langle D \rangle\!\rangle$.

We call this collection of data an *observation system.*

Definition 1. *Two terms $\Gamma \vdash M, M' : C$ are* contextually equivalent*, wrt a specified observation system, when for any ground type D and any program with a hole $\mathcal{N} \ : \ (\Gamma \vdash C) \longrightarrow (\vdash D)$ we have $\langle\!\langle \mathcal{N}[M] \rangle\!\rangle = \langle\!\langle \mathcal{N}[M'] \rangle\!\rangle$.*

2.2 Contextual Morphisms

We now have to extend each part of our syntax to incorporate a type-hole. More generally, K type holes, where K is a set; we take $K = 1$ if just one type-hole is desired.

- *Types with K type-holes* are defined by the usual syntax for types, extended with type-holes

  $$C ::= \ \cdots \ | \ [\cdot\cdot_k] \ (k \in K)$$

- A *typing context with K type-holes* is a finite set of identifiers, each assigned a type with K type-holes, written $\mathtt{x} : A$.
- A *judgement with K type-holes*, written $\Gamma \vdash C$, consists of
  - a typing context Γ with K type-holes
  - and a type C with K type-holes.
- We define a typing judgement for terms with type-holes and holes

  $$\mathcal{N} \ : \ (\Gamma_h \vdash C_h)_{h \in H} \xrightarrow[K]{} (\Delta \vdash D) \qquad (2)$$

  where $(\Gamma_h \vdash C_h)_{h \in H}$ is a family of judgements with K type-holes, and $(\Delta \vdash D)$ a judgement with K type-holes. The typing rules for (2) are just the same as those given for (1) in Section 2.1.

Given an K-indexed family of types $\mathbf{A} = (A_k)_{k \in K}$, we define

- for each type C with K type-holes, a type $C[\mathbf{A}]$ by induction on C–it is A_k in the case $C = [\cdot\cdot_k]$
- for each typing context Γ with type-hole, a typing context

  $$\Gamma[\mathbf{A}] \quad \overset{\text{def}}{=} (\mathtt{x} : C[\mathbf{A}])_{(\mathtt{x}:C) \in \Gamma}$$

- for each judgement $\Gamma \vdash C$ with type-hole, a judgement

  $$(\Gamma \vdash C)[\mathbf{A}] \overset{\text{def}}{=} (\Gamma[\mathbf{A}] \vdash C[\mathbf{A}])$$

- for each term with type-hole and holes (2) a term with holes

  $$\mathcal{N}[\mathbf{A}] \ : \ ((\Gamma_h \vdash C_h)[\mathbf{A}])_{h \in H} \longrightarrow (\Delta \vdash D)[\mathbf{A}]$$

  by induction on $\mathcal{N}$.

$$\begin{tikzcd}
\prod_{h \in H} \mathcal{Q}^{\equiv}((\Gamma_h \vdash C_h)[A]) \arrow[r, "\mathcal{N}[A]^{\equiv}"] \arrow[d, "\prod_{h \in H} \theta_{\Gamma_h \vdash C_h}"'] & \mathcal{Q}^{\equiv}((\Delta \vdash D)[A]) \arrow[d, "\theta_{\Delta \vdash D}"] \\
\prod_{h \in H} \mathcal{Q}^{\equiv}((\Gamma_h \vdash C_h)[B]) \arrow[r, "\mathcal{N}[B]^{\equiv}"'] & \mathcal{Q}^{\equiv}((\Delta \vdash D)[B])
\end{tikzcd}$$

Figure 3. θ must be preserved by $\mathcal{N}$

Whereas contextual equivalence is defined wrt an observation system, contextual isomorphism (or morphism) is defined wrt a congruence.

Definition 2. *Let A and B be types. A* contextual morphism *$\theta \ : \ A \to B$, with respect to a congruence $\equiv$, consists of a function*

$$\theta_{\Gamma \vdash C} \ : \ \mathcal{Q}^{\equiv}((\Gamma \vdash C)[A]) \ \to \ \mathcal{Q}^{\equiv}((\Gamma \vdash C)[B])$$

for each judgement with a type-hole $\Gamma \vdash C$, where θ is preserved by every term with type-hole and holes

$$\mathcal{N} \ : \ (\Gamma_h \vdash C_h)_{h \in H} \xrightarrow[1]{} (\Delta \vdash D)$$

i.e. Figure 3 commutes. θ is a contextual isomorphism *when each function $\theta_{\Gamma \vdash C}$ is a bijection.*

Remark 1. *As stated in Section 1.3, it suffices for θ to be preserved by every term constructor. Figure 3 then follows by induction on $\mathcal{N}$.*

The category of types and contextual morphisms will be written $\mathbf{Con}_1$ (with the subscript explained in the next section). We take $\equiv$ to be fixed. The groupoid of types and contextual isomorphisms is precisely $\mathbf{Isos}\,(\mathbf{Con}_1)$.

2.3 Families of Terms, Families of Types

So far we have considered contextual equivalence betweeen terms and contextual morphisms between types. These notions can be generalized to *families* of terms or of types.

Let $(\Gamma_h \vdash C_h)_{h \in H}$ be a family of judgements. Then two families of terms

$$\begin{array}{rcl} \mathbf{M} & = & (\Gamma_h \vdash M_h : C_h)_{h \in H} \\ \mathbf{M}' & = & (\Gamma_h \vdash M'_h : C_h)_{h \in H} \end{array}$$

are said to be *contextually equivalent* wrt a specified observation system when for any ground type D and program with a hole $\mathcal{N} \ : \ (\Gamma_h \vdash C_h)_{h \in H} \longrightarrow (\vdash D)$ we have $\langle\!\langle \mathcal{N}[\mathbf{M}] \rangle\!\rangle = \langle\!\langle \mathcal{N}[\mathbf{M}'] \rangle\!\rangle$.

We proceed likewise for families of types. Let K be a set. We form a category $\mathbf{Con}_K$, in which an object is an K-indexed family of types $\mathbf{A} = (A_k)_{k \in K}$. A morphism $\theta \ : \ \mathbf{A} \to \mathbf{B}$ is a *contextual morphism* wrt $\equiv$, i.e. a function

$$\theta_{\Gamma \vdash C} \ : \ \mathcal{Q}^{\equiv}((\Gamma \vdash C)[\mathbf{A}]) \ \to \ \mathcal{Q}^{\equiv}((\Gamma \vdash C)[\mathbf{B}])$$

for each judgement with K type-holes $\Gamma \vdash C$, where θ is preserved by every term with type-holes and holes

$$\mathcal{N} \ : \ (\Gamma_h \vdash C_h)_{h \in H} \xrightarrow[K]{} (\Delta \vdash D)$$

i.e. Figure 3, with A and B replaced by $\mathbf{A}$ and $\mathbf{B}$, commutes.

3. Map-based Isomorphisms

We begin by recalling the traditional notion of isomorphism of types.

Consider any typed calculus that includes the following typing rules:

$$\frac{}{\Gamma \vdash \mathtt{x} : A}\,(\mathtt{x} : A) \in \Gamma \qquad \frac{\Gamma \vdash M : A \quad \Gamma, \mathtt{x} : A \vdash N : B}{\Gamma \vdash \mathtt{let}\ M\ \mathtt{be}\ \mathtt{x}.\ N : B}$$

$$\begin{aligned}
\mathtt{let}\ \mathtt{y}\ \mathtt{be}\ \mathtt{x}.\ M &\equiv M[\mathtt{y}/\mathtt{x}] \\
\mathtt{let}\ M\ \mathtt{be}\ \mathtt{x}.\ \mathtt{x} &\equiv M \\
\mathtt{let}\ M\ \mathtt{be}\ \mathtt{x}.\ \mathtt{let}\ N\ \mathtt{be}\ \mathtt{y}.\ P &\equiv \\
&\mathtt{let}\ (\mathtt{let}\ M\ \mathtt{be}\ \mathtt{x}.\ N)\ \mathtt{be}\ \mathtt{y}.\ P
\end{aligned}$$

Figure 4. Some basic laws (with typing and freshness assumptions omitted)

Here $\mathtt{let}\ M\ \mathtt{be}\ \mathtt{x}.\ N$ can be taken either as primitive or to abbreviate $(\lambda\mathtt{x}.N)M$.

Let $\equiv$ be a congruence that includes the laws of Fig. 4, and hence is preserved by renaming. These laws are reasonable regardless of whether the language is pure, call-by-value effectful or call-by-name effectful. We form the category $\mathbf{Map}$ in which

- an object is a type
- a morphism $A \longrightarrow B$ is a *map*, i.e. an $\equiv$-class of terms $\mathtt{x} : A \vdash M : B$
- the identity on A is $(\mathtt{x} : A \vdash \mathtt{x} : A)^{\equiv}$
- the composite of $(\mathtt{x} : A \vdash M : B)^{\equiv}$ and $(\mathtt{x} : B \vdash N : C)^{\equiv}$ is $(\mathtt{x} : A \vdash \mathtt{let}\ M\ \mathtt{be}\ \mathtt{x}.\ N : B)^{\equiv}$.

A *map-based isomorphism* is an isomorphism in $\mathbf{Map}$.

4. Pure λ-Calculus

We come to our first main result. For simply typed λ-calculus, with a congruence $\equiv$ that includes the β- and η-laws, the category $\mathbf{Con}_1$ is isomorphic to $\mathbf{Isos}\,(\mathbf{Map})$, This justifies *a posteriori* the traditional notion of isomorphism. More generally, we shall see that $\mathbf{Con}_K$ is isomorphic to $\mathbf{Isos}\,(\mathbf{Map})^K$.

4.1 Syntax and Laws

Suppose a set of base types is given. Then our types are inductively defined:

$$A \quad ::= \quad b\,(\text{base type}) \mid 0 \mid A + A \mid 1 \mid A \times A \mid A \to A$$

Suppose a set of typed constants[3] is also given. Then our terms are inductively defined:

$$\begin{aligned}
M \quad ::= \quad & c\,(\text{constant}) \mid \mathtt{x} \mid \mathtt{let}\ M\ \mathtt{be}\ \mathtt{x}.\ N \\
& \mid \mathtt{case}\ M\ \mathtt{of}\ \{\,\} \mid \mathtt{inl}\ M \mid \mathtt{inr}\ M \\
& \mid \mathtt{case}\ M\ \mathtt{of}\ \{\mathtt{inl}\ \mathtt{x}.\ M, \mathtt{inr}\ \mathtt{x}.\ M\} \\
& \mid \langle\,\rangle \mid \langle M, M\rangle \mid \pi\, M \mid \pi'\, M \\
& \mid \lambda\mathtt{x}\, M \mid M\, M
\end{aligned}$$

We omit the evident typing rules. Let $\equiv$ be a congruence that includes all the laws in Fig. 5, and hence is preserved by substitution. All these laws are reasonable for pure λ-calculus, though in the presence of effects they would not be.

4.2 Functoriality and Naturality in Isomorphisms

It is well known (Lambek and Scott 1986) that the category $\mathbf{Map}$ is *bicartesian closed*. For any such category $\mathcal{C}$, both $\times$ and $+$ are functors $\mathcal{C} \times \mathcal{C} \to \mathcal{C}$. By contrast, $\to$ is contravariant on the left: it forms a functor $\mathcal{C}^{\mathsf{op}} \times \mathcal{C} \to \mathcal{C}$. Figure 6 displays these functors for $\mathbf{Map}$.. The mixed variance does not trouble us, since we use only isomorphisms. Every type C with K type-holes gives a (covariant) functor $\mathbf{Isos}\,(\mathbf{Map})^K \to \mathbf{Isos}\,(\mathbf{Map})$, by induction on C. It sends an object $\mathbf{A} \in \mathbf{Map}^K$ to $C[\mathbf{A}]$, and is defined on an isomorphism $\sigma : \mathbf{A} \cong \mathbf{B}$ as follows.

$$\begin{aligned}
\mathtt{let}\ M\ \mathtt{be}\ \mathtt{x}.\ N &\equiv N[M/\mathtt{x}] \\
\mathtt{case}\ (\mathtt{inl}\ M)\ \mathtt{of}\ \{\mathtt{inl}\ \mathtt{x}.\ N, \mathtt{inr}\ \mathtt{y}.\ N'\} &\equiv N[M/\mathtt{x}] \\
\mathtt{case}\ (\mathtt{inr}\ M)\ \mathtt{of}\ \{\mathtt{inl}\ \mathtt{x}.\ N, \mathtt{inr}\ \mathtt{y}.\ N'\} &\equiv N'[M/\mathtt{y}] \\
\pi\langle M, M'\rangle &\equiv M \\
\pi'\langle M, M'\rangle &\equiv M' \\
(\lambda\mathtt{x}.\ M)N &\equiv M[N/\mathtt{x}]
\end{aligned}$$

$$\begin{aligned}
N[M/\mathtt{z}] &\equiv \mathtt{case}\ M\ \mathtt{of}\ \{\,\} \\
N[M/\mathtt{z}] &\equiv \mathtt{case}\ M\ \mathtt{of}\ \{\mathtt{inl}\ \mathtt{x}.\ N[\mathtt{inl}\ \mathtt{x}/\mathtt{z}], \\
&\qquad \mathtt{inr}\ \mathtt{x}.\ N[\mathtt{inr}\ \mathtt{x}/\mathtt{z}]\} \\
M &\equiv \langle\,\rangle \\
M &\equiv \langle \pi M, \pi' M\rangle \\
M &\equiv \lambda\mathtt{x}.(M\mathtt{x})
\end{aligned}$$

Figure 5. β- and η-laws (with typing and freshness assumptions omitted)

$$\begin{aligned}
b[\sigma] &\overset{\text{def}}{=} \mathsf{id}_b \\
[\cdot\cdot_k][\sigma] &\overset{\text{def}}{=} \sigma_k \\
0[\sigma] &\overset{\text{def}}{=} \mathsf{id}_0 \\
(C + D)[\sigma] &\overset{\text{def}}{=} C[\sigma] + D[\sigma] \\
1[\sigma] &\overset{\text{def}}{=} \mathsf{id}_1 \\
(C \times D)[\sigma] &\overset{\text{def}}{=} C[\sigma] \times D[\sigma] \\
(C \to D)[\sigma] &\overset{\text{def}}{=} C[\sigma]^{-1} \to D[\sigma]
\end{aligned}$$

Thus types are functorial in isomorphisms. We next show that *judgements* are too, but this time the functors are to $\mathbf{Set}$. Recall that any category $\mathcal{C}$ has a *hom functor* $\mathcal{C}^{\mathsf{op}} \times \mathcal{C} \to \mathbf{Set}$, sending

- an object, i.e. a pair (X, Y) of $\mathcal{C}$-objects, to $\mathcal{C}(X, Y)$, the set of morphisms $X \to Y$
- a morphism $(X, Y) \to (X', Y')$, i.e. a pair of $\mathcal{C}$-morphisms $f : X' \to X$ and $h : Y \to Y'$, to the function $\mathcal{C}(X, Y) \to \mathcal{C}(X', Y')$ sending g to the composite

$$\begin{tikzcd} X' \arrow[r, "f"] & X \arrow[r, "g"] & Y \arrow[r, "h"] & Y' \end{tikzcd}$$

The following is a variant of the hom functor for $\mathbf{Map}$.

Definition 3. *Let $\mathcal{U} = \{\mathtt{y}_0, \ldots, \mathtt{y}_{n-1}\}$ be a finite set of identifiers. We define a functor*

$$\mathcal{Q}^{\equiv}(\vdash) \ : \ (\mathbf{Map}^{\mathsf{op}})^{\mathcal{U}} \times \mathbf{Map} \to \mathbf{Set}$$

as follows.

- *An object is a judgement $\Gamma \vdash C$ where the domain of Γ is $\mathcal{U}$. It is sent to the set $(\Gamma \vdash C)^{\equiv}$.*
- *A morphism $((\mathtt{y} : A_{\mathtt{y}})_{\mathtt{y} \in \mathcal{U}} \vdash C) \to ((\mathtt{y} : B_{\mathtt{y}})_{\mathtt{y} \in \mathcal{U}} \vdash D)$, consists of an equivalence class $(\mathtt{x} : B_{\mathtt{y}} \vdash N_{\mathtt{y}} : A_{\mathtt{y}})^{\equiv}$ for each $\mathtt{y} \in \mathcal{U}$ and another equivalence class $(\mathtt{x} : C \vdash P : D)^{\equiv}$. It is sent to the function*

$$\mathcal{Q}^{\equiv}((\mathtt{y} : A_{\mathtt{y}})_{\mathtt{y} \in \mathcal{U}} \vdash C) \to \mathcal{Q}^{\equiv}((\mathtt{y} : B_{\mathtt{y}})_{\mathtt{y} \in \mathcal{U}} \vdash D)$$

sending $((\mathtt{y} : A_{\mathtt{y}})_{\mathtt{y} \in \mathcal{U}} \vdash M : C)^{\equiv}$ to

$$((\mathtt{y} : B_{\mathtt{y}})_{\mathtt{y} \in \mathcal{U}} \vdash P[M[N_{\mathtt{y}}[\mathtt{y}/\mathtt{x}]/\mathtt{y}]_{\mathtt{y} \in \mathcal{U}}/\mathtt{x}] : D)^{\equiv}$$

[3] More generally, we could consider binding operations rather than constants, but that extra generality is redundant since we have included product and function types.

$$\begin{array}{lclcl}
(\mathtt{x} : A \vdash M : B)^{\equiv} & + & (\mathtt{x} : C \vdash N : D)^{\equiv} & \overset{\text{def}}{=} & (\mathtt{x} : A + C \vdash \mathtt{case\ x\ of}\ \{\mathtt{inl\ x.\ inl}\ M, \mathtt{inr\ x.\ inr}\ N\} : B + D)^{\equiv} \\
(\mathtt{x} : A \vdash M : B)^{\equiv} & \times & (\mathtt{x} : C \vdash N : D)^{\equiv} & \overset{\text{def}}{=} & (\mathtt{x} : A \times C \vdash \langle M[\pi\mathtt{x}/\mathtt{x}], N[\pi'\mathtt{x}/\mathtt{x}]\rangle : B \times D)^{\equiv} \\
(\mathtt{x} : B \vdash M : A)^{\equiv} & \to & (\mathtt{x} : C \vdash N : D)^{\equiv} & \overset{\text{def}}{=} & (\mathtt{x} : A \to C \vdash \lambda\mathtt{y}.\ N[(\mathtt{x}(M[\mathtt{y}/\mathtt{x}]))/\mathtt{x}] : B \to D)^{\equiv}
\end{array}$$

Figure 6. The functors $+$, $\times$ and $\to$ applied to morphisms of $\mathbf{Map}$

Hence, for each judgement with K type-holes $\Gamma \vdash C$, we obtain a functor

$$\mathcal{Q}^{\equiv}((\Gamma \vdash C)[\cdot\cdot]) \ : \ \mathbf{Isos}\,(\mathbf{Map})^K \to \mathbf{Set} \qquad (3)$$

It sends an object $\mathbf{A} \in \mathbf{Map}^K$ to $\mathcal{Q}^{\equiv}((\Gamma \vdash C)[\mathbf{A}])$, and an isomorphism $\sigma \ : \ \mathbf{A} \cong \mathbf{B}$ to

$$\mathcal{Q}^{\equiv}((\mathtt{y} : D[\sigma]^{-1})_{(\mathtt{y}:D)\in\Gamma} \vdash C[\sigma])$$

Finally, every term with type-holes and holes is natural in isomorphisms.

Proposition 1. *Let $\mathcal{N} \ : \ (\Gamma_h \vdash C_h)_{h\in H} \xrightarrow[K]{} (\Delta \vdash D)$. For every isomorphism $\sigma \ : \ \mathbf{A} \cong \mathbf{B}$ in $\mathbf{Map}^K$ the following square in $\mathbf{Set}$ commutes.*

$$\begin{tikzcd}
\prod_{h\in H} \mathcal{Q}^{\equiv}((\Gamma_h \vdash C_h)[\mathbf{A}]) \arrow[r, "\mathcal{N}[\mathbf{A}]^{\equiv}"] \arrow[d, "\prod_{h\in H} \mathcal{Q}^{\equiv}((\Gamma_h \vdash C_h)[\sigma])"] & \mathcal{Q}^{\equiv}((\Delta \vdash D)[\mathbf{A}]) \arrow[d, "\mathcal{Q}^{\equiv}((\Delta \vdash D)[\sigma])"] \\
\prod_{h\in H} \mathcal{Q}^{\equiv}((\Gamma_h \vdash C_h)[\mathbf{B}]) \arrow[r, "\mathcal{N}[\mathbf{B}]^{\equiv}"'] & \mathcal{Q}^{\equiv}((\Delta \vdash D)[\mathbf{B}])
\end{tikzcd}$$

Proof. Induction on $\mathcal{N}$. □

4.3 Characterizing Contextual Morphisms

Let K be a set.

Definition 4. *Let $\sigma \ : \ \mathbf{A} \cong \mathbf{B}$ in $\mathbf{Map}^K$. We write $\mathcal{G}\sigma$ for the following contextual isomorphism $\mathbf{A} \to \mathbf{B}$. For each judgement $\Gamma \vdash C$ with K type-holes,*

$$(\mathcal{G}\sigma)_{\Gamma\vdash C} \ : \ \mathcal{Q}^{\equiv}((\Gamma \vdash C)[\mathbf{A}]) \to \mathcal{Q}^{\equiv}((\Gamma \vdash C)[\mathbf{B}])$$

is $\mathcal{Q}^{\equiv}((\Gamma \vdash C)[\sigma])$.

The required commutativity of Fig. 3 follows from Prop. 1. We therefore obtain an identity-on-objects functor $\mathcal{G} \ : \ \mathbf{Isos}\,(\mathbf{Map})^K \to \mathbf{Con}_K$. It preserves identities and composition because (3) does. Our goal is to show that it is an isomorphism of categories. The proof resembles that of the Yoneda Lemma.

First, a trivial observation.

Lemma 2. *For any type C,*

$$C[\cdot\cdot] \ : \ \mathbf{Isos}\,(\mathbf{Map})^K \to \mathbf{Isos}\,(\mathbf{Map})$$

is the constant functor to C.

Proof. Induction on C. □

Next we describe how to recover σ and σ^{-1} from $\mathcal{G}\sigma$.

Lemma 3. *Let $\sigma \ : \ \mathbf{A} \cong \mathbf{B}$ in $\mathbf{Map}^K$.*

$$\begin{array}{lrcll}
\textit{For each } k \in K & \sigma_k & = & (\mathcal{G}\sigma)_{\mathtt{x}:A_k\vdash[\cdot\cdot_k]}\ \mathsf{id}_{\mathbf{A}} & (4) \\
 & \sigma_k^{-1} & = & (\mathcal{G}\sigma)_{\mathtt{x}:[\cdot\cdot_k]\vdash A_k}\ \mathsf{id}_{\mathbf{A}} & (5)
\end{array}$$

Proof. σ_k must be of the form $(\mathtt{x} : A_k \vdash M : B_k)^{\equiv}$. For (4),

$$\begin{array}{rcl}
\text{RHS} & = & \mathcal{Q}^{\equiv}((\mathtt{x} : A_k \vdash [\cdot\cdot_k])[\sigma])\ \mathsf{id}_{\mathbf{A}} \\
 & = & \mathcal{Q}^{\equiv}(\mathtt{x} : A_k[\sigma]^{-1} \vdash [\cdot\cdot_k][\sigma])\ \mathsf{id}_{\mathbf{A}} \\
 & = & \mathcal{Q}^{\equiv}(\mathtt{x} : \mathsf{id}_{A_k} \vdash \sigma_k)\ \mathsf{id}_{\mathbf{A}} \\
 & & \quad \text{by Lemma 2} \\
 & = & (\mathtt{x} : A_k \vdash M[\mathtt{x}[\mathtt{x}/\mathtt{x}]/\mathtt{x}] : B_k)^{\equiv} \\
 & = & \sigma_k
\end{array}$$

(5) is proved similarly. □

Theorem 4. *$\mathcal{G}$ is an isomorphism of categories $\mathbf{Isos}\,(\mathbf{Map})^K \cong \mathbf{Con}_K$.*

Proof. Given a contextual morphism $\theta \ : \ \mathbf{A} \to \mathbf{B}$, we must show $\theta = \mathcal{G}\sigma$ for a unique $\sigma \ : \ \mathbf{A} \cong \mathbf{B}$ in $\mathbf{Map}$.

$$\begin{array}{lrcl}
\text{For each } k \in K, \text{ set} & \sigma_k & \overset{\text{def}}{=} & \theta_{\mathtt{x}:A_k\vdash[\cdot\cdot_k]}\ \mathsf{id}_{\mathbf{A}} \\
 & \sigma_k^{-1} & \overset{\text{def}}{=} & \theta_{\mathtt{x}:[\cdot\cdot_k]\vdash A_k}\ \mathsf{id}_{\mathbf{A}}
\end{array}$$

By Lemma 3 this is the only possibility. Preservation of θ by

$$\begin{array}{l}
\mathtt{let}\ [\cdot_0]\ \mathtt{be\ x.}\ [\cdot_1] \\
\ :\ (\mathtt{x} : A_k \vdash [\cdot\cdot_k]), ([\cdot\cdot_k] \vdash A_k) \xrightarrow[K]{} (\mathtt{x} : A_k \vdash A_k)
\end{array}$$

applied to $\mathsf{id}_{A_k}, \mathsf{id}_{A_k}$ gives

$$\begin{array}{rcl}
\sigma;\sigma^{-1} & = & \theta_{\mathtt{x}:A_k\vdash A_k}(\mathtt{x} : A \vdash \mathtt{let\ x\ be\ x.\ x} : A)^{\equiv} \\
 & = & \theta_{\mathtt{x}:A_k\vdash A_k}\mathsf{id}_A
\end{array}$$

Preservation of θ by

$$\mathtt{x} \ : \ \xrightarrow[K]{} (\mathtt{x} : A_k \vdash A_k)$$

applied to nothing gives

$$\mathsf{id}_A \quad = \quad \theta_{\mathtt{x}:A_k\vdash A_k}\ \mathsf{id}_A$$

so $\sigma_k;\sigma_k^{-1} = \mathsf{id}_{A_k}$. By a similar argument $\sigma_k^{-1};\sigma_k = \mathsf{id}_{B_k}$. So σ and σ^{-1} are inverse.

Next we show that for every type C with K type-holes,

$$\begin{array}{rcl}
C[\sigma] & = & \theta_{\mathtt{x}:C[\mathbf{A}]\vdash C}\ \mathsf{id}_{C[\mathbf{A}]} \\
C[\sigma]^{-1} & = & \theta_{\mathtt{x}:C\vdash C[\mathbf{A}}\ \mathsf{id}_{C[\mathbf{A}]}
\end{array}$$

by induction on C. The case where $C = [\cdot\cdot_k]$ is trivial, and each type constructor case involves one appeal to θ-preservation.

Finally, given a judgement $\Gamma \vdash D$ with K type-holes and $\omega \in \mathcal{Q}^{\equiv}((\Gamma \vdash D)[\mathbf{A}])$, we wish to show

$$\theta_{\Gamma\vdash D}\,\omega = \mathcal{Q}^{\equiv}((\Gamma \vdash D)[\sigma])\,\omega \qquad (6)$$

Pick $M \in \omega$ and an enumeration $\Gamma = \mathtt{y}_0 : C_0, \ldots, \mathtt{y}_{n-1} : C_{n-1}$. Then preservation of θ by

$$\begin{array}{l}
\mathtt{let}\ (\mathtt{let}\ \mathtt{y}_0\ \mathtt{be\ x.}\ [\cdot_0])\ \mathtt{be}\ \mathtt{y}_0. \\
\cdots \\
\mathtt{let}\ (\mathtt{let}\ \mathtt{y}_{n-1}\ \mathtt{be\ x.}\ [\cdot_{n-1}])\ \mathtt{be}\ \mathtt{y}_{n-1}. \\
\mathtt{let}\ M\ \mathtt{be\ x.}\ [\cdot_n] \\
\ :\ (\mathtt{x} : C_i \vdash C_i[\mathbf{A}])_{i<n}, (\mathtt{x} : D[\mathbf{A}] \vdash D) \xrightarrow[K]{} (\Gamma \vdash D)
\end{array}$$

applied to $\mathsf{id}_{C_0}, \ldots, \mathsf{id}_{C_{n-1}}, \mathsf{id}_D$ gives (6). □

Corollary 5. *Every contextual morphism, i.e. every morphism in* $\mathbf{Con}_K$*, is an isomorphism.*

5. Call-by-Push-Value

5.1 The Calculus

We now turn from pure to effectful λ-calculus, specifically call-by-push-value. In this section, we give the definition of contextual morphism, but referring only to computations, not values.

We briefly recall call-by-push-value; for details, see (Levy 2004). Suppose a set of value base types (b) and a set of computation base types ($\underline{b}$) are given. Then our types are inductively defined:

$$\begin{array}{rl} \text{Value types } A ::= & b \mid U\underline{B} \mid 0 \mid A + A \mid 1 \mid A \times A \mid \mathtt{nat} \\ \text{Computation types } \underline{B} ::= & \underline{b} \mid FA \mid 1_\Pi \mid \underline{B} \,\Pi\, \underline{B} \mid A \to \underline{B} \end{array}$$

Computation types are often underlined. $U\underline{B}$ is the type of thunks of computations in $\underline{B}$, whilst FA is the type of computations whose goal is to return a value of type A. Functions are computations (unlike in call-by-value) so function types are computation types. A *typing context* Γ is a finite set of identifiers, each assigned a value type, written $\mathtt{x} : A$. There are two typing judgements: $\Gamma \vdash^{\mathsf{v}} V : A$ for values and $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ for computations.

Suppose a set of typed value constants and a set of typed computation constants are given. The typing rules are shown in Fig. 7. Sequencing is represented as $M \,\mathtt{to}\, \mathtt{x}.\, N$. For typing contexts Γ and Δ, a *substitution* $k : \Gamma \to \Delta$ associates to each $(\mathtt{x} : A) \in \Gamma$ a value $\Delta \vdash^{\mathsf{v}} k(\mathtt{x}) : A$. This induces a substitution function k^* from terms in context Γ to terms in context Δ. A *congruence* is an equivalence relation on computations in context $\Gamma \vdash^{\mathsf{c}} M \equiv N : \underline{B}$ and on values in context $\Gamma \vdash^{\mathsf{v}} V \equiv W : A$ that is preserved by each term constructor and by weakening.

Note that we disallow pattern-matching into values. This is to avoid the operational complication of values needing to be evaluated, so-called *complex values*. However, somewhat inconsistently and purely to keep the examples easy, we have included some complex values: $V + 1$, $2 * V$ and $V = W$.

A strong monad T on a suitable category $\mathcal{V}$ gives a denotational semantics of call-by-push-value, where a value type denotes a $\mathcal{V}$-object and a computation type denotes an (Eilenberg-Moore) T-algebra. In particular $[\![U\underline{B}]\!]$ is the carrier of $[\![\underline{B}]\!]$, and $[\![FA]\!]$ the free T-algebra on $[\![A]\!]$. A typing context Γ denotes $\prod_{(\mathtt{x}:A)\in\Gamma}[\![A]\!]$. A value $\Gamma \vdash^{\mathsf{v}} V : B$ denotes a $\mathcal{V}$-morphism $[\![\Gamma]\!] \to [\![B]\!]$, and a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes a $\mathcal{V}$-morphism $[\![\Gamma]\!] \to$ the carrier of $[\![\underline{B}]\!]$.

To translate to call-by-push-value, a call-by-value type A is mapped to a value type A^{v}, and a call-by-value term $\Gamma \vdash M : B$ to a computation $(\mathtt{x} : A^{\mathsf{v}})_{(\mathtt{x}:A)\in\Gamma} \vdash^{\mathsf{c}} M^{\mathsf{v}} : FB^{\mathsf{v}}$. In particular:

$$\begin{array}{rcl} \mathtt{x}^{\mathsf{v}} & = & \mathtt{return}\,\mathtt{x} \\ (\mathtt{let}\, M\, \mathtt{be}\, \mathtt{x}.\, N)^{\mathsf{v}} & = & M^{\mathsf{v}}\, \mathtt{to}\, \mathtt{x}.\, N^{\mathsf{v}} \end{array}$$

A call-by-name type A is mapped to a computation type A^{n}, and a call-by-name term $\Gamma \vdash M : B$ to a computation $(\mathtt{x} : UA^{\mathsf{n}})_{(\mathtt{x}:A)\in\Gamma} \vdash^{\mathsf{c}} M^{\mathsf{n}} : B^{\mathsf{n}}$. In particular:

$$\begin{array}{rcl} \mathtt{x}^{\mathsf{n}} & = & \mathtt{force}\,\mathtt{x} \\ (\mathtt{let}\, M\, \mathtt{be}\, \mathtt{x}.\, N)^{\mathsf{n}} & = & \mathtt{let}\,\mathtt{thunk}\, M^{\mathsf{n}}\, \mathtt{be}\, \mathtt{x}.\, N^{\mathsf{n}} \end{array}$$

5.2 Contextual Morphisms

In call-by-push-value, when we define contextual morphism $A \to B$, we do not want A to be replaceable by B in *all* judgements, only in computation judgements. For example, we want an isomorphism $0 \times 0 \cong 0$, yet there is no value $\mathtt{x} : 0 \times 0 \vdash^{\mathsf{v}} V : 0$ because of our disallowance of pattern-matching into values.

Remark 2. *Readers who prefer to include pattern-matching into values would have a value* $\mathtt{x} : 0 \times 0 \vdash^{\mathsf{v}} \mathtt{case}\ \mathtt{x}\ \mathtt{of}\ \{\} : 0$*, so they should instead consider the following example. In call-by-push-value with type recursion, we want an isomorphism* $\mathtt{x} : \mathtt{rec}\, X.\, 1 \times X \cong 0$*, yet there is no value* $\mathtt{x} : \mathtt{rec}\, X.\, 1 \times X \vdash^{\mathsf{v}} V : 0$*.*

We write $\mathcal{Q}^{\equiv}(\Gamma \vdash^{\mathsf{c}} \underline{C})$ for the set of $\equiv$-classes of computations $\Gamma \vdash^{\mathsf{c}} M : \underline{C}$. A particular $\equiv$-class is written $(\Gamma \vdash^{\mathsf{c}} M : \underline{C})^{\equiv}$. Any computation with computation holes

$$\mathcal{N} : (\Gamma_h \vdash^{\mathsf{c}} \underline{C}_h)_{h\in H} \to (\Delta \vdash^{\mathsf{c}} \underline{D})$$

gives rise to a function $\mathcal{N}^{\equiv}$

$$\begin{array}{rcl} \prod_{h\in H} \mathcal{Q}^{\equiv}(\Gamma_h \vdash^{\mathsf{c}} \underline{C}_h) & \to & \mathcal{Q}^{\equiv}(\Delta \vdash^{\mathsf{c}} \underline{D}) \\ (\Gamma_h \vdash^{\mathsf{c}} M_h : \underline{C}_h)^{\equiv}_{h\in H} & \mapsto & (\Delta \vdash^{\mathsf{c}} \mathcal{N}[M_h]_{h\in H} : \underline{D})^{\equiv} \end{array}$$

Definition 5. *Let K and L be sets. Types with K value and L computation type-holes are defined by the syntax*

$$\begin{array}{rl} A ::= & b \mid U\underline{B} \mid 0 \mid A + A \\ & \mid 1 \mid A \times A \mid \mathtt{nat} \mid [\cdot_k]\ (k \in K) \\ \underline{B} ::= & \underline{b} \mid FA \mid 1_\Pi \mid \underline{B} \,\Pi\, \underline{B} \mid A \to \underline{B} \mid [\underline{\cdot}_l]\ (l \in L) \end{array}$$

We then define in the evident way:

- *typing context with K value and L computation type-holes*
- *value judgement (or computation judgement) with K value and L computation type-holes*
- *value with type-holes and holes, writen*
$$\mathcal{V} : (\Gamma_h \vdash^{\mathsf{c}} \underline{C}_h)_{h\in H} \xrightarrow[K,L]{} (\Delta \vdash^{\mathsf{v}} D)$$
- *computation with type-holes and holes, written*
$$\mathcal{N} : (\Gamma_h \vdash^{\mathsf{c}} \underline{C}_h)_{h\in H} \xrightarrow[K,L]{} (\Delta \vdash^{\mathsf{c}} \underline{D})$$

Let A and B be value types. A contextual morphism $\theta : A \to B$*, with respect to a congruence $\equiv$, consists of a function*

$$\theta_{\Gamma\vdash^{\mathsf{c}}\underline{C}} : \mathcal{Q}^{\equiv}((\Gamma \vdash^{\mathsf{c}} \underline{C})[A]) \to \mathcal{Q}^{\equiv}((\Gamma \vdash^{\mathsf{c}} \underline{C})[B])$$

for each computation judgement with a value type-hole $\Gamma \vdash^{\mathsf{c}} \underline{C}$, such that θ is preserved by every computation with a value type-hole and computation holes

$$\mathcal{N} : (\Gamma_h \vdash^{\mathsf{c}} \underline{C}_h)_{h\in H} \xrightarrow[1,0]{} (\Delta \vdash^{\mathsf{c}} \underline{D})$$

i.e. Fig. 8 commutes. θ is a contextual isomorphism *when every function $\theta_{\Gamma\vdash^{\mathsf{c}}\underline{C}}$ is a bijection.*

Remark 3. *By contrast with Remark 1, we cannot meaningfully say that every θ is preserved by every term constructor, because it is given only on computation judgements.*

The category of value types and contextual morphisms is written $\mathbf{Con}_{1,0}$. We likewise define the category $\mathbf{Con}_{0,1}$ of computation types and contextual morphisms. We can generalize both these categories by working with families of types, as in Section 2.3. Let K and L be sets; then we define a category $\mathbf{Con}_{K,L}$, in which an object

$$\mathbf{A} = ((A_k)_{k\in K}, (\underline{A}_l)_{l\in L}))$$

consists of an K-indexed family of value types and a L-indexed family of computation types. A morphism $\theta : \mathbf{A} \to \mathbf{B}$ is a contextual morphism, which consists of a function

$$\theta_{\Gamma\vdash^{\mathsf{c}}\underline{C}} : \mathcal{Q}^{\equiv}((\Gamma \vdash^{\mathsf{c}} \underline{C})[\mathbf{A}]) \to \mathcal{Q}^{\equiv}((\Gamma \vdash^{\mathsf{c}} \underline{C})[\mathbf{B}])$$

$$\frac{}{\Gamma \vdash^{\mathsf{v}} c : A}\ \text{(c:A value constant)} \qquad \frac{}{\Gamma \vdash^{\mathsf{c}} c : \underline{B}}\ \text{(c:}\underline{\text{B}}\text{ computation constant)} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A}{\Gamma \vdash^{\mathsf{c}} \mathtt{return}\ V : FA} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : FA \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} N : \underline{B}}{\Gamma \vdash^{\mathsf{c}} M\ \mathtt{to}\ \mathtt{x}.\ N : \underline{B}} \qquad \frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \lambda \mathtt{x}.M : A \to \underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{c}} M : A \to \underline{B}}{\Gamma \vdash^{\mathsf{c}} M\, V : \underline{B}}$$

$$\frac{}{\Gamma \vdash^{\mathsf{v}} \mathtt{x} : A}\ (\mathtt{x} : A) \in \Gamma \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{v}} \mathtt{thunk}\ M : U\underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : U\underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{force}\ V : \underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A}{\Gamma \vdash^{\mathsf{v}} \mathtt{inl}\ V : A + A'} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A'}{\Gamma \vdash^{\mathsf{v}} \mathtt{inr}\ V : A + A'}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A + A' \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B} \qquad \Gamma, \mathtt{y} : A' \vdash^{\mathsf{c}} M' : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{case}\ V\ \mathtt{of}\ \{\mathtt{inl}\ \mathtt{x}.\ M,\ \mathtt{inr}\ \mathtt{y}.\ M'\} : \underline{B}} \qquad \frac{}{\Gamma \vdash^{\mathsf{v}} \langle\,\rangle : 1} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : 1 \quad \Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{split}\ V\ \mathtt{as}\ \langle\,\rangle.\ M : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{v}} V' : A'}{\Gamma \vdash^{\mathsf{v}} \langle V, V' \rangle : A \times A'} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{split}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y} \rangle.\ M : \underline{B}} \qquad \frac{}{\Gamma \vdash^{\mathsf{c}} \prec \succ : 1_{\Pi}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : 0}{\Gamma \vdash^{\mathsf{c}} \mathtt{case}\ V\ \mathtt{of}\ \{\,\} : \underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B} \quad \Gamma \vdash^{\mathsf{c}} M' : \underline{B}'}{\Gamma \vdash^{\mathsf{c}} \prec M, M' \succ : \underline{B}\ \Pi\ \underline{B}'} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}\ \Pi\ \underline{B}'}{\Gamma \vdash^{\mathsf{c}} \pi\, M : \underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}\ \Pi\ \underline{B}'}{\Gamma \vdash^{\mathsf{c}} \pi'\, M : \underline{B}'}$$

Natural numbers

$$\frac{}{\Gamma \vdash^{\mathsf{v}} \mathtt{const}_n : \mathtt{nat}}\ n \in \mathbb{N} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \mathtt{nat}}{\Gamma \vdash^{\mathsf{v}} V\mathtt{+1} : \mathtt{nat}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \mathtt{nat} \quad \Gamma \vdash^{\mathsf{c}} M : \underline{B} \quad \Gamma, \mathtt{x} : \mathtt{nat} \vdash^{\mathsf{c}} M' : \underline{B}}{\Gamma \vdash^{\mathsf{v}} \mathtt{case}\ V\ \mathtt{as}\ \{\mathtt{0}.\ M,\ \mathtt{x+1}.\ M'\} : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : \mathtt{nat} \quad \Gamma \vdash^{\mathsf{v}} V' : \mathtt{nat}}{\Gamma \vdash^{\mathsf{v}} V = V' : 1 + 1} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \mathtt{nat}}{\Gamma \vdash^{\mathsf{v}} \mathtt{2} * \mathtt{V} : \mathtt{nat}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \mathtt{nat} \quad \Gamma, \mathtt{x} : \mathtt{nat} \vdash^{\mathsf{c}} M : \underline{B} \quad \Gamma, \mathtt{y} : \mathtt{nat} \vdash^{\mathsf{c}} M' : \underline{B}}{\Gamma \vdash^{\mathsf{v}} \mathtt{case}\ V\ \mathtt{as}\ \{\mathtt{2} * \mathtt{x}.\ M,\ \mathtt{2} * \mathtt{y} + \mathtt{1}.\ M'\} : \underline{B}}$$

State

$$\frac{\Gamma, \mathtt{x} : \mathtt{nat} \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{read}\ \mathtt{l}\ \mathtt{as}\ \mathtt{x}.\ M : \underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \mathtt{nat} \quad \Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{v}} \mathtt{l} := V.\ M : \underline{B}}$$

Exceptions

$$\frac{\Gamma \vdash^{\mathsf{v}} V : \mathtt{nat}}{\Gamma \vdash^{\mathsf{c}} \mathtt{raise}\ (\mathtt{e}, V) : \underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} M : FA \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} N : \underline{B} \quad \Gamma, \mathtt{y} : \mathtt{nat} \vdash N' : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{try}\ M\{\ \mathtt{to}\ \mathtt{x}.\ N,\ \mathtt{catch}\ (\mathtt{e}, \mathtt{y}).\ N'\} : \underline{B}}$$

Figure 7. Terms of call-by-push-value

$$\begin{tikzcd}
\prod_{h \in H} \mathcal{Q}^{\equiv}((\Gamma_h \vdash^{\mathsf{c}} \underline{C}_h)[A]) \arrow[r, "{\mathcal{N}[A]^{\equiv}}"] \arrow[d, "\prod_{h \in H} \theta_{\Gamma_h \vdash^{\mathsf{c}} \underline{C}_h}"'] & \mathcal{Q}^{\equiv}((\Delta \vdash^{\mathsf{c}} \underline{D})[A]) \arrow[d, "\theta_{\Delta \vdash^{\mathsf{c}} \underline{D}}"] \\
\prod_{h \in H} \mathcal{Q}^{\equiv}((\Gamma_h \vdash^{\mathsf{c}} \underline{C}_h)[B]) \arrow[r, "{\mathcal{N}[B]^{\equiv}}"'] & \mathcal{Q}^{\equiv}((\Delta \vdash^{\mathsf{c}} \underline{D})[B])
\end{tikzcd}$$

Figure 8. θ must be preserved by $\mathcal{N}$

for each computation judgement with K value and L computation type-holes $\Gamma \vdash^{\mathsf{c}} \underline{C}$, such that θ is preserved by every computation with type-holes and computation holes

$$\mathcal{N} \ :\ (\Gamma_h \vdash^{\mathsf{c}} \underline{C}_h)_{h \in H} \xrightarrow[K,L]{} (\Delta \vdash^{\mathsf{c}} \underline{D})$$

The isomorphisms in $\mathbf{Con}_{K,L}$ are precisely the contextual isomorphisms.

6. Thunkability and Linearity

Our aim is to characterize contextual morphisms by analogy with Section 4.3: a contextual morphism between value types corresponds to a thunkable isomorphism, and a contextual morphism between computation types to a linear isomorphism. Before doing this we need to explain thunkability and linearity and their properties. Let $\equiv$ be a congruence that includes all the laws of Fig. 9–10; hence it is preserved by substitution of values into computations.

Definition 6. *1. A computation* $\Gamma \vdash^{\mathsf{c}} M : FA$ *is* thunkable *up to* $\equiv$ *when*

$$\begin{array}{lll} \Gamma \vdash^{\mathsf{c}} \mathtt{return}\ \mathtt{thunk}\ M & \equiv & \\ \quad M\ \mathtt{to}\ \mathtt{x}.\ \mathtt{return}\ \mathtt{thunk}\ \mathtt{return}\ \mathtt{x} & & : FUFA \end{array} \tag{7}$$

2. A computation $\Gamma, \mathtt{u} : U\underline{B} \vdash^{\mathsf{c}} M : \underline{C}$ *is* linear *in* $\mathtt{u}$ *up to* $\equiv$ *when*

$$\begin{array}{lll} \Gamma, \mathtt{z} : UFU\underline{B} \vdash^{\mathsf{c}} & \mathtt{force}\ \mathtt{z}\ \mathtt{to}\ \mathtt{u}.\ M & \equiv \\ \mathtt{let}\ \mathtt{thunk}\ (\mathtt{force}\ \mathtt{z}\ \mathtt{to}\ \mathtt{w}.\ \mathtt{force}\ \mathtt{w})\ \mathtt{be}\ \mathtt{u}.\ M : & \underline{C} & \end{array} \tag{8}$$

Equation (7) is intended to express the idea that M is effect-free, and equation (8) is intended to say that M is effect-preserving. We explain these in terms of the semantics given by a strong monad.

Thunkability A monad T is *of codescent type* when the fork

$$\begin{tikzcd}
1_{\mathcal{V}} \arrow[r, "\eta"] & T \arrow[r, shift left, "T\eta"] \arrow[r, shift right, "\eta T"'] & T^2
\end{tikzcd}$$

is an equalizer (Bucalo et al. 2003). This is equivalent to η_X being a regular monomorphism for all X (Barr and Wells 1985)[Lemma

$$
\begin{array}{rcl}
\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M & \equiv & M[V/\mathtt{x}] \\
(\mathtt{return}\ V)\ \mathtt{to}\ \mathtt{x}.\ M & \equiv & M[V/\mathtt{x}] \\
\mathtt{force}\ \mathtt{thunk}\ M & \equiv & M \\
\mathtt{case}\ (\mathtt{inl}\ V)\ \mathtt{of}\ \{\mathtt{inl}\ \mathtt{x}.\ M,\ \mathtt{inr}\ \mathtt{y}.\ M'\} & \equiv & M[V/\mathtt{x}] \\
\mathtt{case}\ (\mathtt{inr}\ V)\ \mathtt{of}\ \{\mathtt{inl}\ \mathtt{x}.\ M,\ \mathtt{inr}\ \mathtt{y}.\ M'\} & \equiv & M'[V/\mathtt{y}] \\
\mathtt{split}\ \langle\,\rangle\ \mathtt{as}\ \langle\,\rangle.\ M & \equiv & M \\
\mathtt{split}\ \langle V, V'\rangle\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.\ M & \equiv & M[V/\mathtt{x}, V'/\mathtt{y}] \\
\pi \prec M, M' \succ & \equiv & M \\
\pi' \prec M, M' \succ & \equiv & M' \\
(\lambda\mathtt{x}.M)V & \equiv & M[V/\mathtt{x}] \\
M & \equiv & M\ \mathtt{to}\ \mathtt{x}.\ \mathtt{return}\ \mathtt{x} \\
V & \equiv & \mathtt{thunk}\ \mathtt{force}\ V \\
M[V/\mathtt{z}] & \equiv & \mathtt{case}\ V\ \mathtt{of}\ \{\,\} \\
M[V/\mathtt{z}] & \equiv & \mathtt{case}\ V\ \mathtt{of}\ \{\mathtt{inl}\ \mathtt{x}.\ M[\mathtt{inl}\ \mathtt{x}/\mathtt{z}],\ \mathtt{inr}\ \mathtt{y}.\ M[\mathtt{inr}\ \mathtt{y}/\mathtt{z}]\} \\
M[V/\mathtt{z}] & \equiv & \mathtt{split}\ V\ \mathtt{as}\ \langle\,\rangle.\ M[\langle\,\rangle/\mathtt{z}] \\
M[V/\mathtt{z}] & \equiv & \mathtt{split}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.\ M[\langle \mathtt{x}, \mathtt{y}\rangle/\mathtt{z}] \\
M & \equiv & \prec\ \succ \\
M & \equiv & \prec \pi M, \pi' M \succ \\
M & \equiv & \lambda\mathtt{x}.(M\,\mathtt{x}) \\
(P\ \mathtt{to}\ \mathtt{x}.\ M)\ \mathtt{to}\ \mathtt{y}.\ N & \equiv & P\ \mathtt{to}\ \mathtt{x}.\ (M\ \mathtt{to}\ \mathtt{y}.\ N) \\
\prec\ \succ & \equiv & P\ \mathtt{to}\ \mathtt{x}.\ \prec\ \succ \\
\prec P\ \mathtt{to}\ \mathtt{x}.\ M,\ P\ \mathtt{to}\ \mathtt{x}.\ M' \succ & \equiv & P\ \mathtt{to}\ \mathtt{x}.\ \prec M, M' \succ \\
\lambda\mathtt{y}.(P\ \mathtt{to}\ \mathtt{x}.\ M) & \equiv & P\ \mathtt{to}\ \mathtt{x}.\ \lambda\mathtt{y}.M
\end{array}
$$

Figure 9. Laws of call-by-push-value (typing and freshness assumptions omitted)

6, p. 110]. Every monad on **Set** satisfies it, except for those isomorphic to the monad $X \mapsto 1$ or to the monad $X \mapsto \begin{cases} \emptyset & (X = \emptyset) \\ 1 & \text{otherwise} \end{cases}$. Thunkability of $\Gamma \vdash^{\mathsf{c}} M : FA$ says that the diagram

$$\begin{tikzcd}
\llbracket\Gamma\rrbracket \arrow[r, "\llbracket M\rrbracket"] \arrow[d, "\llbracket M\rrbracket"'] & T\llbracket A\rrbracket \arrow[d, "T\eta_{\llbracket A\rrbracket}"] \\
T\llbracket A\rrbracket \arrow[r, "\eta_{T\llbracket A\rrbracket}"'] & T^2\llbracket A\rrbracket
\end{tikzcd}$$

commutes. So if T is of codescent type, there is a unique $\mathcal{V}$-morphism $f : \llbracket\Gamma\rrbracket \to \llbracket A\rrbracket$ such that

$$\begin{tikzcd}
\llbracket\Gamma\rrbracket \arrow[d, "f"'] \arrow[dr, "\llbracket M\rrbracket"] & \\
\llbracket A\rrbracket \arrow[r, "\eta_{\llbracket A\rrbracket}"'] & T\llbracket A\rrbracket
\end{tikzcd}$$

This precisely says that M returns a value without performing effects.

Linearity Let T be a monad with strength t. For a $\mathcal{V}$-object X and T-algebras (Y, θ) and (Z, ϕ), a *T-algebra homomorphism* $(Y, \theta) \xrightarrow[X]{} (Z, \phi)$ is a $\mathcal{V}$-morphism $f : X \times Y \to Z$ satisfying

$$\begin{tikzcd}
X \times TY \arrow[r, "t_{X,Y}"] \arrow[d, "X\times\theta"'] & T(X \times Y) \arrow[r, "Tf"] & TZ \arrow[d, "\phi"] \\
X \times Y \arrow[rr, "f"'] & & Z
\end{tikzcd}$$

Linearity of $\Gamma, \mathtt{u} : U\underline{B} \vdash^{\mathsf{c}} M : \underline{C}$ in $\mathtt{u}$ says that $\llbracket M\rrbracket$ is a T-algebra homomorphism $\llbracket\underline{B}\rrbracket \xrightarrow[\llbracket\Gamma\rrbracket]{} \llbracket\underline{C}\rrbracket$.

Remark The above account is intended to give a basic grasp of thunkability and linearity, but there are more general models of call-by-push-value given not by a monad but by an *adjunction* $F \dashv U : \mathcal{D} \to \mathcal{C}$. In such models, thunkable maps correspond to $\mathcal{C}$-morphisms only if the adjunction is of *codescent type*; linear maps correspond to $\mathcal{D}$-morphisms only if it is of *descent type*. Many models of interest enjoy these properties, and any model can be completed to one that enjoys them (Munch-Maccagnoni 2014).

An elegant and useful characterization of thunkability and linearity in terms of associativity between call-by-value composition (`to`) and call-by-name composition (`let thunk`) was presented in (Munch-Maccagnoni 2014).

Proposition 6. *1. A computation $\Gamma \vdash^{\mathsf{c}} M : FA$ is thunkable iff for every substitution $k : \Gamma \to \Delta$ and two computations $\Delta, \mathtt{x} : A \vdash^{\mathsf{c}} N : \underline{B}$ and $\Delta, \mathtt{u} : U\underline{B} \vdash^{\mathsf{c}} P : \underline{C}$ we have*

$$\begin{array}{ll} \Delta \vdash^{\mathsf{c}} & k^*M\ \mathtt{to}\ \mathtt{x}.\ (\mathtt{let}\ \mathtt{thunk}\ N\ \mathtt{be}\ \mathtt{u}.\ P) \quad \equiv \\ & \mathtt{let}\ \mathtt{thunk}\ (k^*M\ \mathtt{to}\ \mathtt{x}.\ N)\ \mathtt{be}\ \mathtt{u}.\ P \quad : \underline{C} \end{array} \tag{9}$$

2. A computation $\Gamma, \mathtt{u} : U\underline{B} \vdash^{\mathsf{c}} P : \underline{C}$ is linear in $\mathtt{y}$ iff for every substitution $k : \Gamma \to \Delta$ and two computations $\Delta \vdash^{\mathsf{c}} M : FA$ and $\Delta, \mathtt{x} : A \vdash^{\mathsf{c}} N : \underline{B}$ we have

$$\begin{array}{ll} \Delta \vdash^{\mathsf{c}} & M\ \mathtt{to}\ \mathtt{x}.\ (\mathtt{let}\ \mathtt{thunk}\ N\ \mathtt{be}\ \mathtt{u}.\ k^*P) \quad \equiv \\ & \mathtt{let}\ \mathtt{thunk}\ (M\ \mathtt{to}\ \mathtt{x}.\ N)\ \mathtt{be}\ \mathtt{u}.\ k^*P \quad : \underline{C} \end{array} \tag{10}$$

Natural numbers

$$\begin{array}{rcl}
\mathtt{const}_n + 1 & \equiv & \mathtt{const}_{n+1} \\
2 * \mathtt{const_n} & \equiv & \mathtt{const}_{2n} \\
(\mathtt{const}_n = \mathtt{const}_n) & \equiv & \mathtt{inl}\,\langle\,\rangle \\
(\mathtt{const}_m = \mathtt{const}_n) & \equiv & \mathtt{inr}\,\langle\,\rangle \qquad\qquad (m \neq n) \\
\mathtt{case}\ 0\ \mathtt{as}\ \{0.\,M,\ \mathtt{y}{+}1.\,M'\} & \equiv & M \\
\mathtt{case}\ V{+}1\ \mathtt{as}\ \{0.\,M,\ \mathtt{y}{+}1.\,M'\} & \equiv & M'[V/\mathtt{y}] \\
M[V/\mathtt{z}] & \equiv & \mathtt{case}\ V\ \mathtt{as}\ \{0.\,M[0/\mathtt{z}],\ \mathtt{y}{+}1.\,M[\mathtt{y}{+}1/\mathtt{z}]\} \\
\mathtt{case}\ 2 * \mathtt{V}\ \mathtt{as}\ \{2 * \mathtt{x}.\,M,\ 2 * \mathtt{y} + 1.\,M'\} & \equiv & M[V/\mathtt{x}] \\
\mathtt{case}\ 2 * \mathtt{V} + 1\ \mathtt{as}\ \{2 * \mathtt{x}.\,M,\ 2 * \mathtt{y} + 1.\,M'\} & \equiv & M'[V/\mathtt{y}] \\
M[V/\mathtt{z}] & \equiv & \mathtt{case}\ V\ \mathtt{as}\ \{2 * \mathtt{x}.\,M[2 * \mathtt{x}/\mathtt{z}],\ 2 * \mathtt{y} + 1.\,M[2 * \mathtt{y} + 1/\mathtt{z}]\}
\end{array}$$

State (Plotkin and Power 2002)

$$\begin{array}{rcl}
(\mathtt{read}\ \mathtt{l}\ \mathtt{as}\ \mathtt{x}.\,M)\ \mathtt{to}\ \mathtt{y}.\,N & \equiv & \mathtt{read}\ \mathtt{l}\ \mathtt{as}\ \mathtt{x}.\,(M\ \mathtt{to}\ \mathtt{y}.\,N) \\
(\mathtt{l} := V.\,M)\ \mathtt{to}\ \mathtt{y}.\,N & \equiv & \mathtt{l} := V.\,(M\ \mathtt{to}\ \mathtt{y}.\,N) \\
\mathtt{read}\ \mathtt{l}\ \mathtt{as}\ \mathtt{x}.\,M & \equiv & \mathtt{read}\ \mathtt{l}\ \mathtt{as}\ \mathtt{x}.\,\mathtt{l} := \mathtt{x},\ M \\
\mathtt{l} := V.\,\mathtt{read}\ \mathtt{l}\ \mathtt{as}\ \mathtt{x}.\,M & \equiv & \mathtt{l} := V.\,M[V/\mathtt{x}] \\
\mathtt{l} := V.\,\mathtt{l} := W.\,M & \equiv & \mathtt{l} := W.\,M
\end{array}$$

Exceptions (Levy 2006a)

$$\begin{array}{rcl}
\mathtt{try}\ (\mathtt{return}\ V)\{\ \mathtt{to}\ \mathtt{x}.\,M,\ \mathtt{catch}\ (\mathtt{e},\mathtt{y}).\,N\} & \equiv & M[V/\mathtt{x}] \\
\mathtt{try}\ (\mathtt{raise}\ (\mathtt{e},W))\{\ \mathtt{to}\ \mathtt{x}.\,M,\ \mathtt{catch}\ (\mathtt{e},\mathtt{y}).\,N\} & \equiv & N[W/\mathtt{y}] \\
M\ \mathtt{to}\ \mathtt{x}.\,N & \equiv & \mathtt{try}\ M\{\ \mathtt{to}\ \mathtt{x}.\,N,\ \mathtt{catch}\ (\mathtt{e},\mathtt{y}).\,\mathtt{raise}\ (\mathtt{e},\mathtt{y})\} \\
\left(M \left\{ \begin{array}{l} \mathtt{to}\ \mathtt{x}.\,N \\ \mathtt{catch}\ \mathtt{y}.\,N' \end{array}\right.\right) \left\{ \begin{array}{l} \mathtt{to}\ \mathtt{z}.\,P \\ \mathtt{catch}\ \mathtt{w}.\,P' \end{array}\right. & \equiv & M \left\{ \begin{array}{l} \mathtt{to}\ \mathtt{x}.\,N \left\{ \begin{array}{l} \mathtt{to}\ \mathtt{z}.\,P \\ \mathtt{catch}\ \mathtt{w}.\,P' \end{array}\right. \\ \mathtt{catch}\ \mathtt{y}.\,N' \left\{ \begin{array}{l} \mathtt{to}\ \mathtt{z}.\,P \\ \mathtt{catch}\ \mathtt{w}.\,P' \end{array}\right. \end{array}\right. \\
\prec\,\succ & \equiv & P\{\ \mathtt{to}\ \mathtt{x}.\ \prec\,\succ,\ \mathtt{catch}\ \mathtt{y}.\ \prec\,\succ\} \\
\prec P \left\{ \begin{array}{l} \mathtt{to}\ \mathtt{x}.\,M \\ \mathtt{catch}\ \mathtt{y} \in N \end{array}\right. ,\ P \left\{ \begin{array}{l} \mathtt{to}\ \mathtt{x}.\,M' \\ \mathtt{catch}\ \mathtt{y} \in N' \end{array}\right. \succ & \equiv & P \left\{ \begin{array}{l} \mathtt{to}\ \mathtt{x}.\ \prec M, M' \succ \\ \mathtt{catch}\ \mathtt{y}.\ \prec N, N' \succ \end{array}\right. \\
\lambda\mathtt{z}.\left(P \left\{ \begin{array}{l} \mathtt{to}\ \mathtt{x}.\,M \\ \mathtt{catch}\ \mathtt{y}.\,N \end{array}\right.\right) & \equiv & P \left\{ \begin{array}{l} \mathtt{to}\ \mathtt{x}.\,\lambda\mathtt{z}.\,M \\ \mathtt{catch}\ \mathtt{y}.\,\lambda\mathtt{z}.\,N \end{array}\right.
\end{array}$$

Figure 10. Laws of natural numbers, state and exceptions (typing and freshness assumptions omitted)

The following property of thunkable computations is called "centrality" in (Power and Robinson 1997).

Proposition 7. *If $\Gamma \vdash^{\mathsf{c}} M : FA$ is thunkable then for all $\Gamma \vdash^{\mathsf{c}} N : FB$ and $\Gamma, \mathtt{x} : A, \mathtt{y} : B \vdash^{\mathsf{c}} P : \underline{C}$ we have*

$$M\ \mathtt{to}\ \mathtt{x}.\,N\ \mathtt{to}\ \mathtt{y}.\,P \equiv N\ \mathtt{to}\ \mathtt{y}.\,M\ \mathtt{to}\ \mathtt{x}.\,P$$

Proposition 8. *1. The following computations are thunkable. Typing assumptions are omitted.*

$$\begin{array}{rl}
\mathtt{return}\ V & \\
k^* M & \textit{if } M \textit{ is thunkable} \\
M\ \mathtt{to}\ \mathtt{x}.\,N & \textit{if } M \textit{ and } N \textit{ are thunkable} \\
\mathtt{case}\ V\ \mathtt{of}\ \{\,\} & \\
\mathtt{case}\ V\ \mathtt{of}\ \{\mathtt{inl}\ \mathtt{x}.\,M, \mathtt{inr}\ \mathtt{y}.\,M'\} & \\
 & \textit{if } M \textit{ and } M' \textit{ are thunkable} \\
\mathtt{split}\ V\ \mathtt{as}\ \langle\,\rangle.\,M & \textit{if } M \textit{ is thunkable} \\
\mathtt{split}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y} \rangle.\,M & \textit{if } M \textit{ is thunkable} \\
\mathtt{case}\ V\ \mathtt{of}\ \{0.\,M, \mathtt{y}{+}1.\,M'\} & \\
 & \textit{if } M \textit{ and } M' \textit{ are thunkable} \\
\mathtt{case}\ V\ \mathtt{of}\ \{2 * \mathtt{x}.\,M, 2 * \mathtt{y} + 1.\,M'\} & \\
 & \textit{if } M \textit{ and } M' \textit{ are thunkable}
\end{array}$$

2. The following computations are linear in u*. Typing assumptions are omitted, but* u *must be fresh for V, N and k.*

$$\begin{array}{rl}
\mathtt{force}\ \mathtt{u} & \\
k^* M & \textit{if } M \textit{ is linear in } \mathtt{u} \\
\mathtt{let}\ \mathtt{thunk}\ M\ \mathtt{be}\ \mathtt{v}.\,N & \textit{if } M \textit{ is linear in } \mathtt{u} \textit{ and } N \textit{ in } \mathtt{v} \\
\mathtt{case}\ V\ \mathtt{of}\ \{\,\} & \\
\mathtt{case}\ V\ \mathtt{of}\ \{\mathtt{inl}\ \mathtt{x}'\ M, \mathtt{inr}\ \mathtt{y}.\,M'\} & \\
 & \textit{if } M \textit{ and } M' \textit{ are linear in } \mathtt{u} \\
\mathtt{split}\ V\ \mathtt{as}\ \langle\,\rangle.\,M & \textit{if } M \textit{ is linear in } \mathtt{u} \\
\mathtt{split}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y} \rangle.\,M & \textit{if } M \textit{ is linear in } \mathtt{u} \\
\mathtt{case}\ V\ \mathtt{of}\ \{0.\,M, \mathtt{y}{+}1.\,M'\} & \\
 & \textit{if } M \textit{ and } M' \textit{ are linear in } \mathtt{u} \\
\mathtt{case}\ V\ \mathtt{of}\ \{2 * \mathtt{x}.\,M, 2 * \mathtt{y} + 1.\,M'\} & \\
 & \textit{if } M \textit{ and } M' \textit{ are linear in } \mathtt{u} \\
\prec\,\succ & \\
\prec M, M' \succ & \textit{if } M \textit{ and } M' \textit{ are linear in } \mathtt{u}
\end{array}$$

$$\begin{array}{rl} \pi\, M & \textit{if } M \textit{ is linear in } \mathtt{u} \\ \pi'\, M & \textit{if } M \textit{ is linear in } \mathtt{u} \\ \lambda\mathtt{x}.M & \textit{if } M \textit{ is linear in } \mathtt{u} \\ M\, V & \textit{if } M \textit{ is linear in } \mathtt{u} \\ N\ \mathtt{to}\ \mathtt{x}.\ M & \textit{if } N \textit{ is thunkable and } M \textit{ linear in } \mathtt{u} \\ M\ \mathtt{to}\ \mathtt{x}.\ N & \textit{if } M \textit{ is linear in } \mathtt{u} \end{array}$$

Proposition 9. *1. Let $\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : FB$ be thunkable. Let $\Gamma, \mathtt{y} : B \vdash^{\mathsf{c}} N : FA$ be an* inverse *of M, i.e.*

$$\begin{array}{lcccl} \Gamma, \mathtt{x} : A & \vdash^{\mathsf{c}} & M\ \mathtt{to}\ \mathtt{y}.\ N & \equiv & \mathtt{return}\ \mathtt{x} & : FA \\ \Gamma, \mathtt{y} : B & \vdash^{\mathsf{c}} & N\ \mathtt{to}\ \mathtt{x}.\ M & \equiv & \mathtt{return}\ \mathtt{y} & : FB \end{array}$$

Then N is thunkable.

2. Let $\Gamma, \mathtt{u} : U\underline{B} \vdash^{\mathsf{c}} M : \underline{C}$ be linear in $\mathtt{u}$. Let $\Gamma, \mathtt{v} : U\underline{C} \vdash^{\mathsf{c}} N : \underline{B}$ be an inverse *of M i.e.*

$$\begin{array}{lcccl} \Gamma, \mathtt{u} : U\underline{B} & \vdash^{\mathsf{c}} & \mathtt{let}\ \mathtt{thunk}\ M\ \mathtt{be}\ \mathtt{y}.\ N & \equiv & \mathtt{force}\ \mathtt{u} & : \underline{B} \\ \Gamma, \mathtt{v} : U\underline{C} & \vdash^{\mathsf{c}} & \mathtt{let}\ \mathtt{thunk}\ N\ \mathtt{be}\ \mathtt{x}.\ M & \equiv & \mathtt{force}\ \mathtt{v} & : \underline{C} \end{array}$$

Then N is linear in $\mathtt{v}$.

Let us write $\mathcal{Q}^{\equiv}(\Gamma \vdash^{\mathsf{v}} C)$ for the set of all $\equiv$-classes of thunkable $\Gamma \vdash^{\mathsf{c}} M : FC$. Then any value with computation holes

$$\mathcal{V}\ :\ (\Gamma_h \vdash^{\mathsf{c}} \underline{C}_h)_{h \in H} \to (\Delta \vdash^{\mathsf{v}} D)$$

gives rise to a function $(\mathcal{V})^{\equiv}$

$$\begin{array}{lcl} \prod_{h \in H} \mathcal{Q}^{\equiv}(\Gamma_h \vdash^{\mathsf{c}} \underline{C}_h) & \to & \mathcal{Q}^{\equiv}(\Delta \vdash^{\mathsf{v}} D) \\ (\Gamma_h \vdash^{\mathsf{c}} M_h : \underline{C}_h)^{\equiv}_{h \in H} & \mapsto & \\ \qquad (\Delta \vdash^{\mathsf{c}} \mathtt{return}\ \mathcal{V}[M_h]_{h \in H} : FD)^{\equiv} & & \end{array}$$

7. Characterizing Contextual Morphisms

We now adapt our account of contextual morphisms in λ-calculus to call-by-push-value, using thunkable and linear maps.

7.1 Functoriality and Naturality in Thunkable and Linear Isomorphisms

Definition 7. *1. A* thunkable map *$A \longrightarrow B$ is an $\equiv$-class of thunkable computations $\mathtt{x} : A \vdash^{\mathsf{c}} M : FB$. These form a category $\mathbf{Thk}$. The identity on A is $(\mathtt{x} : A \vdash^{\mathsf{c}} \mathtt{return}\ \mathtt{x} : FA)^{\equiv}$. The composite of $(\mathtt{x} : A \vdash^{\mathsf{c}} M : FB)^{\equiv}$ and $(\mathtt{x} : B \vdash^{\mathsf{c}} N : FC)^{\equiv}$ is $(\mathtt{x} : A \vdash^{\mathsf{c}} M\ \mathtt{to}\ \mathtt{x}.\ N : FC)^{\equiv}$.*

2. A linear map *$\underline{A} \longrightarrow \underline{B}$ is an $\equiv$-class of computations $\mathtt{x} : U\underline{A} \vdash^{\mathsf{c}} N : \underline{B}$ linear in $\mathtt{x}$. These form a category $\mathbf{Lin}$. The identity on $\underline{A}$ is $(\mathtt{x} : U\underline{A} \vdash^{\mathsf{c}} \mathtt{force}\ \mathtt{x} : \underline{A})^{\equiv}$. The composite of $(\mathtt{x} : U\underline{A} \vdash^{\mathsf{c}} M : \underline{B})^{\equiv}$ $(\mathtt{x} : U\underline{B} \vdash^{\mathsf{c}} N : \underline{C})^{\equiv}$ is $(\mathtt{x} : U\underline{A} \vdash^{\mathsf{c}} \mathtt{let}\ \mathtt{thunk}\ M\ \mathtt{be}\ \mathtt{x}.\ N : \underline{C})^{\equiv}$.*

As in Section 4.2, our type constructors are mixed variance functors.

$$\begin{array}{rccccc} U & : & \mathbf{Lin} & \to & \mathbf{Thk} \\ + & : & \mathbf{Thk} \times \mathbf{Thk} & \to & \mathbf{Thk} \\ \times & : & \mathbf{Thk} \times \mathbf{Thk} & \to & \mathbf{Thk} \\ F & : & \mathbf{Thk} & \to & \mathbf{Lin} \\ \Pi & : & \mathbf{Lin} \times \mathbf{Lin} & \to & \mathbf{Lin} \\ \to & : & \mathbf{Thk}^{\mathsf{op}} \times \mathbf{Lin} & \to & \mathbf{Lin} \end{array}$$

They are defined in Figure 11.

Using these, every type is functorial in its type-holes. Let K and L be sets, and we write $\mathbf{A}$, $\mathbf{B}$ etc. for a pair of an K-indexed family of value types and an L-indexed family of computation types. We define functors as follows.

- Every value type C with K value and L computation type-holes gives a functor

$$C[\cdot\cdot]\ :\ \mathbf{Isos}\,(\mathbf{Thk})^K \times \mathbf{Isos}\,(\mathbf{Lin})^L \to \mathbf{Isos}\,(\mathbf{Thk})$$

- Every computation type $\underline{C}$ with K value and L computation type-holes gives a functor

$$\underline{C}[\cdot\cdot]\ :\ \mathbf{Isos}\,(\mathbf{Thk})^K \times \mathbf{Isos}\,(\mathbf{Lin})^L \to \mathbf{Isos}\,(\mathbf{Lin})$$

Next we need to make judgements into functors.

Definition 8. *Let $\mathcal{U} = \{\mathtt{y}_0, \ldots, \mathtt{y}_{n-1}\}$ be a finite set of identifiers. We define a functor*

$$\mathcal{Q}^{\equiv}(\,\vdash^{\mathsf{c}}\,)\ :\ (\mathbf{Thk}^{\mathsf{op}})^{\mathcal{U}} \times \mathbf{Lin} \to \mathbf{Set}$$

as follows.

- *An object is a judgement $\Gamma \vdash^{\mathsf{c}} \underline{C}$ where the domain of Γ is $\mathcal{U}$. It is sent to the set $(\Gamma \vdash^{\mathsf{c}} \underline{C})^{\equiv}$.*
- *A morphism $((\mathtt{y} : A_{\mathtt{y}})_{\mathtt{y} \in \mathcal{U}} \vdash^{\mathsf{c}} \underline{C}) \to ((\mathtt{y} : B_{\mathtt{y}})_{\mathtt{y} \in \mathcal{U}} \vdash^{\mathsf{c}} \underline{D})$, consists of an equivalence class $(\mathtt{x} : B_{\mathtt{y}} \vdash^{\mathsf{c}} N_{\mathtt{y}} : FA_{\mathtt{y}})^{\equiv}$, with $N_{\mathtt{y}}$ thunkable, for each $\mathtt{y} \in \mathcal{U}$, and an equivalence class $(\mathtt{x} : U\underline{C} \vdash^{\mathsf{c}} P : \underline{D})^{\equiv}$, with P linear in $\mathtt{x}$. It is sent to the function*

$$\mathcal{Q}^{\equiv}((\mathtt{y} : A_{\mathtt{y}})_{\mathtt{y} \in \mathcal{U}} \vdash^{\mathsf{c}} \underline{C}) \to \mathcal{Q}^{\equiv}((\mathtt{y} : B_{\mathtt{y}})_{\mathtt{y} \in \mathcal{U}} \vdash^{\mathsf{c}} \underline{D})$$

sending $((\mathtt{y} : A_{\mathtt{y}})_{\mathtt{y} \in \mathcal{U}} \vdash^{\mathsf{c}} M : \underline{C})^{\equiv}$ to

$$\begin{array}{lcl} ((\mathtt{y} : B_{\mathtt{y}})_{\mathtt{y} \in \mathcal{U}} & \vdash^{\mathsf{c}} & N_{\mathtt{y}_0}\ \mathtt{to}\ \mathtt{y}_0. \\ & & \ldots \\ & & N_{\mathtt{y}_{n-1}}\ \mathtt{to}\ \mathtt{y}_{n-1}. \\ & & P[\mathtt{thunk}\ M/\mathtt{x}]\ :\ \underline{D})^{\equiv} \end{array}$$

We also define a functor

$$\mathcal{Q}^{\equiv}(\,\vdash^{\mathsf{v}}\,)\ :\ (\mathbf{Thk}^{\mathsf{op}})^{\mathcal{U}} \times \mathbf{Thk} \to \mathbf{Set}$$

as follows.

- *An object is a judgement $\Gamma \vdash^{\mathsf{v}} C$ where the domain of Γ is $\mathtt{u}$. It is sent to $(\Gamma \vdash^{\mathsf{v}} C)^{\equiv}$.*
- *A morphism $((\mathtt{y} : A_{\mathtt{y}})_{\mathtt{y} \in \mathcal{U}} \vdash^{\mathsf{v}} C) \to ((\mathtt{y} : B_{\mathtt{y}})_{\mathtt{y} \in \mathcal{U}} \vdash^{\mathsf{v}} D)$ consists of an equivalence class $(\mathtt{x} : B_{\mathtt{y}} \vdash^{\mathsf{c}} N_{\mathtt{y}} : FA_{\mathtt{y}})^{\equiv}$, with $N_{\mathtt{y}}$ thunkable, for each $\mathtt{y} \in \mathcal{U}$, and an equivalence class $(\mathtt{x} : C \vdash^{\mathsf{c}} P : FD)^{\equiv}$, with P thunkable. It is sent to the function*

$$\mathcal{Q}^{\equiv}((\mathtt{y} : A_{\mathtt{y}})_{\mathtt{y} \in \mathcal{U}} \vdash^{\mathsf{v}} C) \to \mathcal{Q}^{\equiv}((\mathtt{y} : B_{\mathtt{y}})_{\mathtt{y} \in \mathcal{U}} \vdash^{\mathsf{v}} D)$$

sending $((\mathtt{y} : A_{\mathtt{y}})_{\mathtt{y} \in \mathcal{U}} \vdash^{\mathsf{c}} M : FC)^{\equiv}$, with M thunkable, to

$$\begin{array}{lcl} ((\mathtt{y} : B_{\mathtt{y}})_{\mathtt{y} \in \mathcal{U}} & \vdash^{\mathsf{c}} & N_{\mathtt{y}_0}\ \mathtt{to}\ \mathtt{y}_0. \\ & & \ldots \\ & & N_{\mathtt{y}_{n-1}}\ \mathtt{to}\ \mathtt{y}_{n-1}. \\ & & M\ \mathtt{to}\ \mathtt{x}.\ P\ \ :\ FD)^{\equiv} \end{array}$$

By Proposition 7, these definitions are independent of the particular ordering of $\mathcal{U}$.

Again letting K and L be sets, we define functors as follows.

- Every value judgement $\Gamma \vdash^{\mathsf{v}} C$ with K value and L computation type-holes gives a functor

$$\mathcal{Q}^{\equiv}((\Gamma \vdash^{\mathsf{v}} C)[\cdot\cdot])\ :\ \mathbf{Isos}\,(\mathbf{Thk})^K \times \mathbf{Isos}\,(\mathbf{Lin})^L \to \mathbf{Set}$$

- Every computation judgement $\Gamma \vdash^{\mathsf{c}} \underline{C}$ with K value and L computation type-holes gives a functor

$$\mathcal{Q}^{\equiv}((\Gamma \vdash^{\mathsf{c}} \underline{C})[\cdot\cdot])\ :\ \mathbf{Isos}\,(\mathbf{Thk})^K \times \mathbf{Isos}\,(\mathbf{Lin})^L \to \mathbf{Set}$$

$$
\begin{array}{rcl}
U\,(\mathtt{x} : U\underline{A} \vdash^{\mathsf{c}} M : \underline{B})^{\equiv} & \overset{\text{def}}{=} & (\mathtt{x} : U\underline{A} \vdash^{\mathsf{c}} \mathtt{return}\ \mathtt{thunk}\ M : FU\underline{B})^{\equiv} \\
(\mathtt{x} : A \vdash M : FB)^{\equiv} \quad + \quad (\mathtt{x} : C \vdash N : FD)^{\equiv} & \overset{\text{def}}{=} & \begin{array}{lll} (\mathtt{x} : A + C & \vdash^{\mathsf{c}} & \mathtt{case}\ \mathtt{x}\ \mathtt{of}\ \{ \\ & & \quad \mathtt{inl}\ \mathtt{x}.\ M\ \mathtt{to}\ \mathtt{y}.\ \mathtt{return}\ \mathtt{inl}\ \mathtt{y} \\ & & \quad \mathtt{inr}\ \mathtt{x}.\ N\ \mathtt{to}\ \mathtt{y}.\ \mathtt{return}\ \mathtt{inr}\ \mathtt{y} \\ & & \}\quad : F(B + D))^{\equiv} \end{array} \\
(\mathtt{x} : A \vdash M : FB)^{\equiv} \quad \times \quad (\mathtt{x} : C \vdash N : FD)^{\equiv} & \overset{\text{def}}{=} & \begin{array}{lll} (\mathtt{x} : A \times C & \vdash^{\mathsf{c}} & \mathtt{split}\ \mathtt{x}\ \mathtt{as}\ \langle \mathtt{y}, \mathtt{z} \rangle. \\ & & M[\mathtt{y}/\mathtt{x}]\ \mathtt{to}\ \mathtt{u}. \\ & & N[\mathtt{z}/\mathtt{x}]\ \mathtt{to}\ \mathtt{v}. \\ & & \mathtt{return}\ \langle \mathtt{u}, \mathtt{v} \rangle : F(B \times C))^{\equiv} \end{array} \\
F\,(\mathtt{x} : A \vdash^{\mathsf{c}} M : FB)^{\equiv} & \overset{\text{def}}{=} & (\mathtt{x} : UFA \vdash^{\mathsf{c}} \mathtt{force}\ \mathtt{x}\ \mathtt{to}\ \mathtt{x}.\ M : FB)^{\equiv} \\
(\mathtt{x} : U\underline{A} \vdash^{\mathsf{c}} M : \underline{B})^{\equiv} \quad \Pi \quad (\mathtt{x} : U\underline{C} \vdash^{\mathsf{c}} N : \underline{D})^{\equiv} & \overset{\text{def}}{=} & \begin{array}{lll} (\mathtt{x} : U(\underline{A}\,\Pi\,\underline{C}) & \vdash^{\mathsf{c}} & \prec M[\mathtt{thunk}\ \pi\ \mathtt{force}\ \mathtt{x}/\mathtt{x}], \\ & & \quad N[\mathtt{thunk}\ \pi'\ \mathtt{force}\ \mathtt{x}/\mathtt{x}] \succ \\ & & \quad\ :\ \underline{B}\,\Pi\,\underline{D})^{\equiv} \end{array} \\
(\mathtt{x} : B \vdash^{\mathsf{c}} M : FA)^{\equiv} \quad \to \quad (\mathtt{x} : U\underline{C} \vdash^{\mathsf{c}} N : \underline{D})^{\equiv} & \overset{\text{def}}{=} & \begin{array}{lll} (\mathtt{x} : U(A \to \underline{C}) & \vdash^{\mathsf{c}} & \lambda\mathtt{y}. \\ & & M[\mathtt{y}/\mathtt{x}]\ \mathtt{to}\ \mathtt{z}. \\ & & N[\mathtt{thunk}\ ((\mathtt{force}\ \mathtt{x})\ \mathtt{z})/\mathtt{x}] \\ & & \quad :\ B \to \underline{D})^{\equiv} \end{array}
\end{array}
$$

Figure 11. The functors U, $+$, $\times$, F, Π and $\to$ applied to morphisms of $\mathbf{Thk}$ and $\mathbf{Lin}$

We then show that every value with type-holes and holes

$$\mathcal{V}\ :\ (\Gamma_h \vdash^{\mathsf{c}} \underline{C}_h)_{h \in H} \xrightarrow[K,L]{} (\Delta \vdash^{\mathsf{v}} D)$$

and every computation with type-holes and holes

$$\mathcal{N}\ :\ (\Gamma_h \vdash^{\mathsf{c}} \underline{C}_h)_{h \in H} \xrightarrow[K,L]{} (\Delta \vdash^{\mathsf{c}} \underline{D})$$

is natural as the type-hole argument ranges over $\mathbf{Isos}\,(\mathbf{Thk})^K \times \mathbf{Isos}\,(\mathbf{Lin})^L$.

7.2 Characterizing Contextual Morphisms in Call-by-push-value

We now want to show that

$$\begin{array}{rcl} \mathbf{Con}_{1,0} & \cong & \mathbf{Isos}\,(\mathbf{Thk}) \\ \mathbf{Con}_{0,1} & \cong & \mathbf{Isos}\,(\mathbf{Lin}) \end{array}$$

and more generally

$$\mathbf{Con}_{K,L} \quad\cong\quad \mathbf{Isos}\,(\mathbf{Thk})^K \times \mathbf{Isos}\,(\mathbf{Lin})^L$$

We proceed as in Section 4.3.

Definition 9. *Let $\sigma\ :\ \mathbf{A} \cong \mathbf{B}$ in $\mathbf{Isos}\,(\mathbf{Thk})^K \times \mathbf{Isos}\,(\mathbf{Lin})^L$. We write $\mathcal{G}\sigma$ for the following contextual isomorphism $\mathbf{A} \cong \mathbf{B}$. For each computation judgement $\Gamma \vdash^{\mathsf{c}} \underline{C}$ with K value and L computation type-holes,*

$$(\mathcal{G}\sigma)_{\Gamma \vdash^{\mathsf{c}} \underline{C}}\ :\ \mathcal{Q}^{\equiv}((\Gamma \vdash^{\mathsf{c}} \underline{C})[\mathbf{A}]) \to \mathcal{Q}^{\equiv}((\Gamma \vdash^{\mathsf{c}} \underline{C})[\mathbf{B}])$$

is $\mathcal{Q}^{\equiv}((\Gamma \vdash^{\mathsf{c}} \underline{C})[\sigma])$.

Commutativity of Fig. 8 holds by naturality, and we obtain an identity-on-objects functor

$$\mathcal{G}\ :\ \mathbf{Isos}\,(\mathbf{Thk})^K \times \mathbf{Isos}\,(\mathbf{Lin})^L \to \mathbf{Con}_{K,L}$$

Our goal is to show that it is an isomorphism of categories.

First we describe how to recover σ and σ^{-1} from $\mathcal{G}\sigma$.

Lemma 10. *Let $\sigma\ :\ \mathbf{A} \cong \mathbf{B}$ in $\mathbf{Isos}\,(\mathbf{Thk})^K \times \mathbf{Isos}\,(\mathbf{Lin})^L$.*

$$\begin{array}{lrcl} \textit{For any } k \in K \textit{ we have} & \sigma_k & = & (\mathcal{G}\sigma)_{\mathtt{x}:A_k \vdash^{\mathsf{c}} F[\cdot\cdot_k]}\ \mathsf{id}_{A_k} \\ & \sigma_k^{-1} & = & (\mathcal{G}\sigma)_{\mathtt{x}:[\cdot\cdot_k] \vdash^{\mathsf{c}} FA_k}\ \mathsf{id}_{A_k} \\ \textit{For any } l \in L \textit{ we have} & \underline{\sigma}_l & = & (\mathcal{G}\sigma)_{\mathtt{x}:U\underline{A}_l \vdash^{\mathsf{c}} [\underline{\cdot\cdot}_l]}\ \mathsf{id}_{\underline{A}_l} \\ & \underline{\sigma}_l^{-1} & = & (\mathcal{G}\sigma)_{\mathtt{x}:U[\underline{\cdot\cdot}_l] \vdash^{\mathsf{c}} \underline{A}_l}\ \mathsf{id}_{\underline{A}_l} \end{array}$$

Theorem 11. *$\mathcal{G}$ is an isomorphism of categories.*

Proof. For a contextual morphism $\theta\ :\ \mathbf{A} \to \mathbf{B}$, we must show $\theta = \mathcal{G}\,\sigma$ for unique $\sigma\ :\ \mathbf{A} \cong \mathbf{B}$ in $\mathbf{Isos}\,(\mathbf{Thk})^K \times \mathbf{Isos}\,(\mathbf{Lin})^L$.

$$\begin{array}{lrcl} \text{For each } k \in K \text{ set} & \sigma_k & \overset{\text{def}}{=} & \theta_{\mathtt{x}:A_k \vdash^{\mathsf{c}} F[\cdot\cdot_k]}\ \mathsf{id}_{A_k} \\ & \sigma_k^{-1} & \overset{\text{def}}{=} & \theta_{\mathtt{x}:[\cdot\cdot_k] \vdash^{\mathsf{c}} FA_k}\ \mathsf{id}_{A_k} \\ \text{and for each } l \in L \text{ set} & \underline{\sigma}_l & \overset{\text{def}}{=} & \theta_{\mathtt{x}:U\underline{A}_l \vdash^{\mathsf{c}} [\underline{\cdot\cdot}_l]}\ \mathsf{id}_{\underline{A}_l} \\ & \underline{\sigma}_l^{-1} & \overset{\text{def}}{=} & \theta_{\mathtt{x}:U[\underline{\cdot\cdot}_l] \vdash^{\mathsf{c}} \underline{A}_l}\ \mathsf{id}_{\underline{A}_l} \end{array}$$

By Lemma 10 this is the only possibility. For $k \in K$, we show σ_k to be thunkable as follows. Preservation of θ by

$$\begin{array}{l} \mathtt{return}\ \mathtt{thunk}\ [\cdot] \\ \ :\ (\mathtt{x} : A_k \vdash^{\mathsf{c}} F[\cdot\cdot_l]) \xrightarrow[K,L]{} (\mathtt{x} : A_k \vdash^{\mathsf{c}} FUF[\cdot\cdot_k]) \end{array}$$

applied to `return x` gives

$$\mathtt{return}\ \mathtt{thunk}\ \sigma_k = \theta_{\mathtt{x}:A \vdash^{\mathsf{c}} FUF[\cdot\cdot_k]}(\mathtt{return}\ \mathtt{thunk}\ \mathtt{return}\ \mathtt{x}) \tag{11}$$

Preservation of θ by

$$\begin{array}{l} [\cdot\cdot_k]\ \mathtt{to}\ \mathtt{y}.\ \mathtt{return}\ \mathtt{thunk}\ \mathtt{return}\ \mathtt{y} \\ \ :\ (\mathtt{x} : A_k \vdash^{\mathsf{c}} F[\cdot\cdot_k]) \xrightarrow[K,L]{} (\mathtt{x} : A_k \vdash^{\mathsf{c}} FUF[\cdot\cdot_k]) \end{array}$$

applied to `return x` gives

$$\begin{array}{c} \sigma_k\ \mathtt{to}\ \mathtt{y}.\ \mathtt{return}\ \mathtt{thunk}\ \mathtt{return}\ \mathtt{y} = \\ \theta_{\mathtt{x}:A_k \vdash^{\mathsf{c}}_{\cdot\cdot} FUF[\cdot\cdot_k]}((\mathtt{return}\ \mathtt{x})\ \mathtt{to}\ \mathtt{y}.\ \mathtt{return}\ \mathtt{thunk}\ \mathtt{return}\ \mathtt{y}) \end{array} \tag{12}$$

```
$x : UF(1+1) \vdash^{c}$
  force x to y.
  case y of {
    inl z.
      read l as v.
      l := 2 * v.
      return ⟨⟩
    inr z.
      read l as v.
      l := 2 * v + 1.
      return ⟨⟩
  }          : $F1$
```

```
$x : UF1 \vdash^{c}$
  force x to z.
  read l as u.
  case u of {
    2 * w.
      l := w.
      return inl ⟨⟩
    2 * w + 1.
      l := w.
      return inr ⟨⟩
  }          : $F(1+1)$
```

Figure 12. Linear isomorphism $F(1+1) \cong F1$ in call-by-push-value with mutable state

The right-hand sides of (11) and (12) are equal, so the left-hand sides are equal, i.e. σ_k is thunkable. By a similar argument, $\underline{\sigma}_l$ is linear for all $l \in L$. The rest of the proof follows that of Theorem 4. □

Corollary 12. *Every contextual morphism is an isomorphism.*

8. Examples and Non-examples

To directly give examples and non-examples of contextual isomorphisms would be complicated. Happily we do not need to: Theorem 11 lets us instead work with thunkable or linear isomorphisms. For a ground type A, let $\mathsf{Vals}(A)$ be the set of values $\vdash^{\mathsf{v}} V : A$.

In call-by-push-value with mutable state, Figure 12 displays a linear isomorphism $F(1+1) \cong F1$. Both computations are linear in $\mathtt{x}$ because of Proposition 8(2) (first and last items). We rule out a thunkable iso $1+1 \cong 1$, provided $\equiv$ has the following property.

Definition 10. *Let $\equiv$ be a congruence on call-by-push-value with state, containing Figures 9–10 (minus the exceptions part). It is* precise for state *when, for every ground type A,*

1. *for every $m \in \mathbb{N}$ and $\vdash^{\mathsf{c}} M : FA$ there is unique $n \in \mathbb{N}$ and $W \in \mathsf{Vals}(A)$ such that*

$$\mathtt{l} := \mathtt{const}_m.\, M \quad \equiv \quad \mathtt{l} := \mathtt{const}_n.\, \mathtt{return}\, W$$

2. *for every $\vdash^{\mathsf{c}} M, M' : FA$, if*

$$\mathtt{l} := \mathtt{const}_m.\, M \quad \equiv \quad \mathtt{l} := \mathtt{const}_m.\, M'$$

for all $m \in \mathbb{N}$ then $M \equiv M'$.

For every ground type A, we deduce[4] the following.

- For $W, W' \in \mathsf{Vals}(A)$, if $\mathtt{return}\ W \equiv \mathtt{return}\ W'$ then $W = W'$, by applying (1) to $m = 0$ and $M = W$.
- For any thunkable $\vdash^{\mathsf{c}} M : FA$, there is (unique) $W \in \mathsf{Vals}(A)$ s.t. $M \equiv \mathtt{return}\ W$. Proof: for every $m \in \mathbb{N}$ obtain n_m and W_m such that

$$\mathtt{l} := \mathtt{const}_m.\, M \quad \equiv \quad \mathtt{l} := \mathtt{const}_{n_m}.\, W_m$$

Then $n_m = m$ and $W_m = W_0$ because otherwise

$$\begin{aligned}
\mathtt{return\ false} \quad &\equiv \quad \begin{array}{l} (\mathtt{l} := 0.\ M)\ \mathtt{to\ x}. \\ (\mathtt{l} := \mathtt{const}_m.\ M)\ \mathtt{to\ y}. \\ \mathtt{read\ l\ as\ u}. \\ \mathtt{return}\ (\mathtt{x} = \mathtt{y})\ \mathtt{and}\ (\mathtt{u} = \mathtt{const}_m) \end{array} \\
&\equiv \quad \begin{array}{l} M\ \mathtt{to\ z}. \\ (\mathtt{l} := 0.\ \mathtt{return\ z})\ \mathtt{to\ x}. \\ (\mathtt{l} := \mathtt{const}_m.\ \mathtt{return\ z})\ \mathtt{to\ y}. \\ \mathtt{read\ l\ as\ u}. \\ \mathtt{return}\ (\mathtt{x} = \mathtt{y}\ \mathtt{and}\ (\mathtt{u} = \mathtt{const}_m) \end{array} \\
&\equiv \quad \mathtt{return\ true}
\end{aligned}$$

Therefore $M \equiv \mathtt{return}\ W_0$ by (2).

Let $\mathsf{VGround}$ be the category of ground types and thunkable maps. Let $\mathcal{H} : \mathsf{VGround} \to \mathbf{Set}$ be the following functor.

- A ground type A is sent to $\mathsf{Vals}(A)$.
- A thunkable map $(\mathtt{x} : A \vdash^{\mathsf{c}} M : FB)^{\equiv}$ is sent to the function $\mathsf{Vals}(A) \to \mathsf{Vals}(B)$ sending V to the unique W such that $M[V/\mathtt{x}] \equiv \mathtt{return}\ W$.

Since $\mathcal{H}(1+1) \ncong \mathcal{H}\,1$ in $\mathbf{Set}$, we have $1+1 \ncong 1$ in $\mathsf{VGround}$.

In call-by-push-value with exceptions, Figure 13 displays a thunkable isomorphism $UF1 \cong UF0$. Both computations are thunkable because of Proposition 8(1) (first item). We rule out a linear iso $F1 \cong F0$, provided $\equiv$ has the following property.

Definition 11. 1. *For any ground type A and $p \in \mathsf{Vals}(A) + \mathbb{N}$, let $\vdash^{\mathsf{c}} \tilde{p} : FA$ be $\mathtt{return}\ V$ if $p = \mathsf{inl}\ V$, and $\mathtt{raise}\ (\mathtt{e}, \mathtt{const}_n)$ if $p = \mathsf{inr}\ n$.*

2. *Let $\equiv$ be a congruence on call-by-push-value with exceptions, containing Figures 9–10 (minus the state part). It is* precise for exceptions *when, for any ground type A and any computation $\vdash^{\mathsf{c}} M : FA$, there is unique $p \in \mathsf{Vals}(A) + \mathbb{N}$ such that $M \equiv \tilde{p}$.*

Let $\mathsf{CGround}$ be the category in which an object is a ground type and a morphism $A \to B$ is a linear map $FA \to FB$. Let $\mathbb{N}/\mathbf{Set}$ be the category of $\mathbb{N}$-pointed sets and homomorphisms. Let $\mathcal{K} : \mathsf{CGround} \to \mathbb{N}/\mathbf{Set}$ be the following functor.

- A ground type A is sent to $(\mathsf{Vals}(A) + \mathbb{N}, (\mathsf{inr}\ n)_{n \in \mathbb{N}})$.
- A linear map $(\mathtt{x} : UFA \vdash^{\mathsf{c}} M : FB)^{\equiv}$ is sent to the homomorphism $(\mathsf{Vals}(A)+\mathbb{N}, (\mathsf{inr}\ n)_{n \in \mathbb{N}}) \to (\mathsf{Vals}(B)+\mathbb{N}, (\mathsf{inr}\ n)_{n \in \mathbb{N}})$ sending p to the unique q such that $M[\mathtt{thunk}\ \tilde{p}/\mathtt{x}] \equiv \tilde{q}$.

Since $\mathcal{K}\,1 \ncong \mathcal{K}\,0$ in $\mathbb{N}/\mathbf{Set}$, we have $1 \ncong 0$ in $\mathsf{CGround}$, i.e. there is no linear isomorphism $F1 \cong F0$.

9. Call-by-Value and Call-by-Name

We sketch how to adapt our results to the call-by-value and call-by-name settings.

To define thunkability for a call-by-value language, suppose that $(\,) \to A$ is a nullary function type, i.e. the type of functions that take no arguments and have return type A. Let $\equiv$ be a congruence that includes the standard call-by-value laws (Moggi 1988). A term $\Gamma \vdash M : A$ is *thunkable* up to $\equiv$ when

$$\Gamma \vdash^{\mathsf{c}} \lambda(\,).\, M \equiv \mathtt{let}\ M\ \mathtt{be\ x}.\ \lambda(\,).\, \mathtt{x} \quad : (\,) \to A$$

To define linearity for a call-by-name language, suppose that $+(B)$ is a unary sum type[5], i.e. a datatype with a single constructor $\mathtt{just}$ taking one argument of type B. Let $\equiv$ be a congruence that includes

[4] Essentially this is a proof that the state monad $\mathbb{N} \to (\mathbb{N} \times -)$ on $\mathbf{Set}$ is of codescent type.

[5] Call-by-name sum types do not satisfy the η-law, so they are not categorical coproducts.

```
x : UF1 ⊢^c return thunk (
  try (force x){
    to y.
      raise (e, 0)
    catch (e, v).
      raise (e, v+1)
  }
)                : FUF0
```

```
x : UF0 ⊢^c return thunk (
  try (force x){
    to z.
      case z as { }
    catch (e, u).
      case u of {
        0.
          return ⟨ ⟩
        w+1.
          raise (e, w)
      }
  }
)                : FUF1
```

Figure 13. Thunkable isomorphism $UF\mathtt{unit} \cong UF\mathtt{empty}$ in call-by-push-value with exceptions

```
x : A → bool ⊢^c λp.
  case (x p) of {
    true.
      read l as v.
      l := 2 ∗ v.
      sole
    false.
      read l as v.
      l := 2 ∗ v + 1.
      sole
  }        : A → unit
```

```
x : A → unit ⊢^c λp.
  case (x p) of {.
    sole.
      read l as u.
      case u of {
        2 ∗ w.
          l := w.
          true
        2 ∗ w + 1.
          l := w.
          false
      }
  }             : A → bool
```

Figure 14. Thunkable isomorphism $A \to \mathtt{bool} \cong A \to \mathtt{unit}$ in call-by-value with mutable state

the standard call-by-name laws (Levy 2004)[Figure A.8]. A term $\Gamma, \mathtt{u} : B \vdash M : C$ is *linear* in $\mathtt{u}$ up to $\equiv$ when

$$\begin{array}{l}\Gamma, \mathtt{z} : +(B) \vdash \\ \quad \mathtt{let}\ (\mathtt{case\ z\ of}\ \{\mathtt{just\ w.\ w}\})\ \mathtt{be\ u.}\ M \\ \quad \equiv \quad \mathtt{case\ z\ of}\ \{\mathtt{just\ u.}\ M\} \quad : C\end{array}$$

Both these notions, when translated into call-by-push-value, coincide with Definition 6. The requirements for nullary function types or unary sum types in the language are reasonable, as both are included in the “jumbo” version of λ-calculus (Levy 2006b).

The category of types and contextual morphisms, for a congruence $\equiv$, is isomorphic

- to the groupoid of types and thunkable isomorphisms, if nullary function types are present and $\equiv$ includes the call-by-value laws
- to the groupoid of types and linear isomorphisms, if unary sum types are present and $\equiv$ includes the call-by-name laws.

This allows us, as in Section 8, to work with thunkable and linear isomorphisms instead of contextual ones. For example:

- In call-by-value with state, Figure 14 displays a thunkable isomorphism $A \to \mathtt{bool} \cong A \to \mathtt{unit}$, for any type A. This translates as $U(A \to F(1+1)) \cong U(A \to F1)$. But there is no thunkable isomorphism $\mathtt{bool} \cong \mathtt{unit}$, as in Section 8.
- In call-by-name with exceptions, Figure 15 displays a linear isomorphism $A + \mathtt{unit} \cong A + \mathtt{empty}$, for any type A. This translates as $F(U\underline{A} + UF1) \cong F(U\underline{A} + UF0)$. But there is no linear isomorphism $\mathtt{unit} \cong \mathtt{empty}$, as in Section 8.

10. Conclusions

Contextual isomorphisms provides a unified minimal standard for regarding two types as essentially the same. Map-based isomorphisms, provided they are thunkable or linear, give a sound, complete and convenient technique for generating contextual isomorphisms.

References

M. Barr and C. Wells. *Toposes, Triples and Theories*. Springer-Verlag, Berlin, 1985.

N. Benton and A. Kennedy. Exceptional syntax. *Journal of Functional Programming*, 11(4):395–410, 2001.

```
x : A+unit ⊢^c case x of {
  inl p.
    inl p
  inr q.
    inr (
      try—case q of {
        sole.
          raise (e, 0)
        catch (e, v).
          raise (e, v+1)
      }
    )
}              : A + empty
```

```
x : A+empty ⊢^c case x of {
  inl p.
    inl p
  inr q.
    inr (
      try—case q of {
        catch (e, u).
          case u of {
            0.
              sole
            w+1.
              raise(e, w)
          }
      }
    )
}              : A + unit
```

Figure 15. Linear isomorphism $A + \mathtt{unit} \cong A + \mathtt{empty}$ in call-by-name with exceptions

A. Bucalo, C. Führmann, and A. Simpson. An equational notion of lifting monad. *Theoretical Computer Science*, 294(1-2):31–60, 2003.

P. Clairambault. Isomorphisms of types in the presence of higher-order references. In *Twenty-Sixth Annual IEEE Symposium on Logic In Computer Science*, 2011.

J. de Lataillade. Second-order type isomorphisms through game semantics. *Annals of Pure and Applied Logic*, 151(2-3):115–150, 2008.

R. Di Cosmo. *Isomorphisms of types: from lambda-calculus to information retrieval and language design*. Progress in Theoretical Computer Science. Birkhäuser, 1995.

M. Fiore, R. Di Cosmo, and V. Balat. Remarks on isomorphisms in typed lambda calculi with empty and sum types. In *Proceedings of the 17th IEEE Symposium on Logic in Computer Science (LICS-02))*, pages 147–158, Los Alamitos, July 22–25 2002. IEEE Computer Society.

P. J. Freyd, E. P. Robinson, and G. Rosolini. Functorial parametricity. In A. Scedrov, editor, *Proceedings of the 7th Annual IEEE Symposium on*

Logic in Computer Science, pages 444–452, Santa Cruz, CA, June 1992. IEEE Computer Society Press. ISBN 0-8186-2734-2.

C. Führmann. Direct models for the computational λ-calculus. In S. Brookes, A. Jung, M. Mislove, and A. Scedrov, editors, *Proceedings of the 15th Conference in Mathematical Foundations of Programming Semantics, New Orleans*, volume 20 of *ENTCS*, pages 147–172, 1999.

J. Lambek and P. Scott. *Introduction to Higher Order Categorical Logic*. Cambridge University Press, Cambridge, 1986.

O. Laurent. Classical isomorphisms of types. *Mathematical Structures in Computer Science*, 15(5):969–1004, 2005.

O. Laurent, M. Quatrini, and L. Tortora de Falco. Polarized and focalized linear and classical proofs. *Annals of Pure and Applied Logic*, 134(2–3): 217–264, July 2005.

P. B. Levy. Monads and adjunctions for global exceptions. In *Proceedings, 22nd Conference on the Mathematical Foundations of Programming Semantics*, volume 158 of *ENTCS*, 2006a.

P. B. Levy. Jumbo λ-calculus. In *Proc. , 33rd International Colloquium on Automata, Languages and Programming*, volume 4052 of *LNCS*. Springer, 2006b.

P. B. Levy. *Call-By-Push-Value. A Functional/Imperative Synthesis*. Semantic Structures in Computation. Springer, 2004.

E. Moggi. Computational lambda-calculus and monads. LFCS Report ECS-LFCS-88-66, University of Edinburgh, Oct. 1988.

E. Moggi. Notions of computation and monads. *Information and Computation*, 93, 1991.

J. H. Morris, Jr. *Lambda-Calculus Models of Programming Languages*. Ph. D. dissertation, MIT, Dec. 1968. Report No. MAC–TR–57.

G. Munch-Maccagnoni. *Syntaxe et modèles d'une composition non-associative des programmes et des preuves*. PhD thesis, Université Paris-Diderot - Paris VII, Dec. 10 2013.

G. Munch-Maccagnoni. Models of a non-associative composition. In A. Muscholl, editor, *Foundations of Software Science and Computation Structures - 17th International Conference, Proceedings*, volume 8412 of *Lecture Notes in Computer Science*, pages 396–410. Springer, 2014.

A. M. Pitts. Operationally-based theories of program equivalence. In P. Dybjer and A. M. Pitts, editors, *Semantics and Logics of Computation*, Publications of the Newton Institute, pages 241–298. Cambridge University Press, 1997.

G. Plotkin and J. Power. Notions of computation determine monads. In *Proceedings, Foundations of Software Science and Computation Structures, 2002*, volume 2303 of *LNCS*, pages 342–356. Springer, 2002.

A. J. Power and E. P. Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5):453–468, Oct. 1997.

E. P. Robinson. Parametricity as isomorphism. *Theoretical Computer Science*, 136(1):163–181, 1994.

P. . Selinger. Control categories and duality: On the categorical semantics of the $\lambda\mu$-calculus. *Mathematical Structures in Computer Science*, 11(2): 207–260, 2001.

S. Soloviev and R. Di Cosmo, editors. *Isomorphism of Types*, volume 15(5) of *Mathematical Structures in Computer Science*. Cambridge University Press, 2005.