A cartesian bicategory of polynomial functors in homotopy type theory

Eric Finster, Samuel Mimram, Maxime Lucas, Thomas Seiller

A cartesian bicategory of polynomial functors in homotopy type theory

Eric Finster[1] Samuel Mimram[2,5] Maxime Lucas[3,5] Thomas Seiller[4,5]

Abstract

Polynomial functors are a categorical generalization of the usual notion of polynomial, which has found many applications in higher categories and type theory: those are generated by polynomials consisting a set of monomials built from sets of variables. They can be organized into a cartesian bicategory, which unfortunately fails to be closed for essentially two reasons, which we address here by suitably modifying the model. Firstly, a naive closure is too large to be well-defined, which can be overcome by restricting to polynomials which are finitary. Secondly, the resulting putative closure fails to properly take the 2-categorical structure in account. We advocate here that this can be addressed by considering polynomials in groupoids, instead of sets. For those, the constructions involved into composition have to be performed up to homotopy, which is conveniently handled in the setting of homotopy type theory: we use it here to formally perform the constructions required to build our cartesian bicategory, in Agda. Notably, this requires us introducing an axiomatization in a small universe of the type of finite types, as an appropriate higher inductive type of natural numbers and bijections.

Keywords: polynomial functor, groupoid, cartesian closed bicategory, homotopy type theory

1 Introduction

Polynomial functors have been introduced as a categorical generalization of traditional polynomials and have been intensively studied by Kock and collaborators [9,16,10]. They have become an important categorical tool, allowing the definition and manipulation of various structures such as opetopes [17] or type theories [3]. This motivates the study of the categorical structures they bear in order to facilitate constructions on those, and the situation turns out to be quite subtle. For instance, one would expect that they should be cartesian closed (after all, most usual categories are), but it is not the case: the cartesian product functor does not have a satisfactory right adjoint. There are essentially two reasons for that.

The first one is that there are size issues: the naive definition of the exponential appears to be too large to be a proper object in the category of polynomials. This is easily overcome by restricting to polynomials which are finitary, i.e., only involve monomials consisting of products of variables which are finite. The resulting category is isomorphic to the category of Girard's normal functors [11], which is a model of simply typed λ-calculus, and linear logic [12], which can be thought of as a quantitative variant of the relational model (historically, this model is in fact one of the starting points motivating the introduction of linear logic).

The second one is more problematic: polynomial functors carry an intuitive 2-categorical structure, but it was observed early on that the closure mentioned above fails to extend to a 2-categorical one [11,20,12]. Here, we advocate that a satisfactory answer to this problem is provided by switching from traditional polynomial functors to ones over groupoids, as first considered by Kock [16], see also [23,22]. We show that the resulting bicategory is cartesian closed; it is more generally a model of intuitionistic linear logic, which we expect to

[1] University of Cambridge
[2] École polytechnique
[3] Université Sorbonne Paris Nord
[4] CNRS
[5] Supported by the Île-de-France region through the DIM RFSI project CoHOp.

This paper will be published
in the proceedings of MFPS XXXVII
URL: `https://www.coalg.org/calco-mfps2021/mfps/`

extend as a model of differential linear logic. The resulting category is close to the "equivariant variant" of polynomials, provided by generalized species or analytic functors [5,8,7,6].

In order to assist us in our proofs and help us gain confidence in those, we have formalized most of them, from the beginning, in Agda, in the setting of univalent type theory [21] (with a custom implementation of homotopy type theory). Since this reflects the way we worked, allows easily manipulating objects such as groupoids as 1-truncated types, and ensures for free the functoriality and homotopy invariance of all constructions, we decided to present our results directly in the type-theoretic formalism, and a "translation" into the traditional set-theoretic setting for polynomial functors is planned for future works. Moreover, the present work required us to develop specific type-theoretic constructions, which we think could find application outside the scope of this work. A salient contribution is the construction, as a higher inductive type, of the type of finite sets and bijections in a small universe. The Agda code is publicly available [4].

We recall the traditional definition of polynomials, polynomial functors and associated constructions in section 2, we then present a formalization of the bicategory of polynomial functors in groupoids in section 3 and discuss the cartesian structure and the failure of being properly closed in section 4. We then introduce and study the notion of finite type in section 5 and finally properly construct a cartesian closed bicategory in section 6.

2 Polynomial functors

We begin by recalling the traditional definition of polynomial functors, as well as related constructions. All the material in this section is already known, but required in the following.

2.1 The category of polynomial functors

We briefly recall here the categorical generalization of the notion of polynomial provided by polynomial functors, and refer the reader to [9,15] for a detailed presentation. Polynomials as traditionally defined as finite sums of the form $P(X) = \sum_{0 \leq i < k} X^{n_i}$. This notion can be "categorified" by taking a set B of monomials (instead of specifying their number k) and having a set E_b of instances of X in each monomial b (instead of specifying their number n_i). This data can thus collected as a function $P : E \to B$ representing the *polynomial*, where B is the set of monomials and for each monomial b, $E_b = P^{-1}(b)$ is the set of instances of X involved in the monomial. Such a function P, induces a functor $[\![P]\!] : \mathbf{Set} \to \mathbf{Set}$ defined, by mimicking the usual definition of polynomials, as

$$[\![P]\!](X) = \sum_{b \in B} X^{E_b}$$

and we call *polynomial functor* such a functor. An interesting point of view on the above data consists in considering the elements of B as abstract *operations*, whose *parameters* are the elements of E_b, so that $[\![P]\!](X)$ corresponds to the set obtained by formally applying the operations in B to the required number of elements of the set X.

We will more generally consider the "typed" or "colored" variant of polynomials and polynomial functors, where the parameters of an operation are decorated by a "color" in a set I, as well as their output in a set J. This data can be encoded by a diagram P in $\mathbf{Set}$ of the form

$$I \xleftarrow{s} E \xrightarrow{p} B \xrightarrow{t} J \tag{1}$$

which we call a *polynomial* and consists of an uncolored polynomial p together with functions s and t respectively indicating the colors of the parameters and outputs of operations. Note that the previous uncolored setting is recovered when I and J are both the terminal set. Writing $\mathbf{Set}/I$ for the slice category of sets over a set I, such data again induces a functor $[\![P]\!] : \mathbf{Set}/I \to \mathbf{Set}/J$, called a *polynomial functor*, obtained as the composite $[\![P]\!] = \Sigma_t \circ \Pi_p \circ \Delta_s$ where Δ_s is the pullback map along s, and Σ_t (resp. Π_p) is the left (resp. right) adjoint to Δ_t (resp. Δ_p) given by post-composition by t (resp. local cartesian closure).

Alternatively, polynomial functors can be considered as acting on families instead of slice categories. Given a set I, it is well known that the slice category $\mathbf{Set}/I$ of sets over I is equivalent to the category of families indexed by I,

$$\mathbf{Set}/I \quad \simeq \quad \mathbf{Set}^I \tag{2}$$

and, through this equivalence, the associated polynomial functor is $[\![P]\!] : \mathbf{Set}^I \to \mathbf{Set}^J$ such that

$$[\![P]\!](X_i \mid i \in I) = (\sum_{b \in t^{-1}(j)} \prod_{e \in p^{-1}(b)} X_{s(e)} \mid j \in J)$$

It can be shown (and this is non-trivial) that the composite of two polynomial functors is again polynomial: this means that given two polynomial functors generated by two diagrams of the form (1), one can find a third diagram of the form (1) which is a presentation for the composite of the functors. We can thus build a category **PolyFun** where an object is a set and a morphism $I \to J$ is a polynomial functor $\mathbf{Set}/I \to \mathbf{Set}/J$ (or $\mathbf{Set}^I \to \mathbf{Set}^J$). Note that even though an operation of composition is defined on polynomials (1), we cannot build a category of those: their composition being defined by using universal constructions, it will not be strictly associative and the best we can hope for is a structure of a bicategory. This motivates investigating the 2-categorical structure of polynomials.

2.2 2-categorical structure

Given two polynomials P and P' of the form (1), both from I to J, a morphism between them consists of two functions $\beta : B \to B'$ and $\varepsilon : E \to E'$ between the operations (resp. parameters) of P and those of P' making the diagram

$$\begin{array}{ccccccc} I & \xleftarrow{s} & E & \xrightarrow{p} & B & \xrightarrow{t} & J \\ \| & & \downarrow{\scriptstyle \varepsilon} & \lrcorner & \downarrow{\scriptstyle \beta} & & \| \\ I & \xleftarrow[s']{} & E' & \xrightarrow[p']{} & B' & \xrightarrow[t']{} & J \end{array} \tag{3}$$

commute and such that the middle square is a pullback (such a morphism is sometimes said to be *cartesian* to insist on this requirement). This last condition can be understood as requiring that β preserves the arity of operations (it is also technically important because there is no sensible way of defining horizontal composition of morphisms without this condition). The composition of polynomials defined above turns out to be associative up to isomorphism, so that one can define a bicategory **Poly** whose 0-cells are sets, 1-cells are polynomials and 2-cells are morphisms of polynomials.

The resulting bicategory can be shown to be biequivalent to the 2-category **PolyFun**, obtained by adding suitable natural transformations as 2-cells to the above category [9, Theorem 2.17]. There is a subtlety concerning the 2-cells: the "natural" notion of morphism between polynomial functors, strong natural transformations, is more liberal than the notion of morphism defined above on corresponding polynomials, and one can either restrict those transformations (to cartesian natural transformations) or generalize the notion of morphism between polynomials.

2.3 Cartesian closed structure

The 1-category **PolyFun** of polynomial functors is cartesian. Considering polynomial functors as operating on families through (2), as explained above, the paring of two polynomial functors $P : I \to J$ and $Q : I \to K$ is induced by the one in **Cat**:

$$\langle P, Q \rangle : \mathbf{Set}^I \to \mathbf{Set}^J \times \mathbf{Set}^K \cong \mathbf{Set}^{J \sqcup K}$$

which motivates defining the product on objects of **PolyFun** as the coproduct of sets. We can hope that the category also has a closure for products induced by the one in **Cat**. The sequence of bijections of hom-sets

$$\mathbf{Set}^I \times \mathbf{Set}^J \to \mathbf{Set}^K \quad \cong \quad \mathbf{Set}^I \to (\mathbf{Set}^K)^{\mathbf{Set}^J} \quad \cong \quad \mathbf{Set}^I \to \mathbf{Set}^{\mathbf{Set}^J \times K}$$

suggests that we define the closure as

$$[J, K] = \mathbf{Set}^J \times K \simeq \mathbf{Set}/J \times K \tag{4}$$

However, this does not make sense because the objects of **PolyFun** are sets and this lives in a larger universe. This motivates considering polynomials which are "reasonably small". We say that a polynomial P of the form (1) is *finitary* when every operation has a finite set of parameters, i.e., the set $E_b = p^{-1}(b)$ is finite for every operation b. Polynomial functors corresponding to finitary functors can be shown to be those preserving filtered colimits and are sometimes called *normal functors* [11,12]. From now on, we restrict our category to such polynomials, and consider them up to isomorphism. This change allows us to replace $\mathbf{Set}/J$ by $\mathbf{Set}_{\mathrm{fin}}/J$ in the above definition of the exponential, where $\mathbf{Set}_{\mathrm{fin}}$ is the class of finite sets (i.e., we consider finite families of set). While this is certainly "smaller", this is still not a proper set; however, it is now equivalent to a proper set, namely the set $\mathbb{N}/J$ of functions $[n] \to J$ for some $n \in \mathbb{N}$, where $[n] = \{0, \dots, n-1\}$ is a canonical choice of a finite set with n elements. To sum up, it can be shown that the resulting category is still cartesian and we can define the exponential as $[J, K] = \mathbb{N}/J \times K$.

We have seen that the category of polynomial functors really is a 2-category (or a bicategory if we consider polynomials instead) and it is natural to expect that the cartesian closure would extend to the 2-categorical setting, by which we mean that the isomorphism

$$\mathbf{PolyFun}(I \sqcup J, K) \cong \mathbf{PolyFun}(I, \mathbb{N}/J \times K)$$

between sets of isomorphism classes of polynomial functors should extend to an equivalence of categories. It has however been observed that it is not the case, see [11, Remark 2.19], [20, Example 1.4.2] and [12, Theorem 1.24]. As an illustration, consider the polynomial functor $[\![P]\!](X) = X^2$, which is induced by the polynomial P given by the diagram $1 \leftarrow 2 \rightarrow 1 \rightarrow 1$ (we write n for a set with n elements). It can be remarked that there are two automorphisms on the polynomial P: the identity and the morphism of the form (3) where $\varepsilon : 2 \to 2$ is the transposition. The exponential transpose of $P : 0 \sqcup 1 \to 1$ is the polynomial $P^\sharp : 0 \to \mathbb{N}/1 \times 1$ (whose target is isomorphic to $\mathbb{N}$) induced by the diagram $0 \leftarrow 0 \rightarrow 1 \rightarrow \mathbb{N}$, where the morphism $1 \to \mathbb{N}$ sends the element of 1 to $2 \in \mathbb{N}$ (which is the arity of the unique operation of P). Because 0 and 1 are respectively initial and terminal in sets, the identity is the only automorphism of $P^\sharp$, whereas the two automorphisms of P should induce two automorphisms on its exponential transpose.

2.4 Toward polynomial functors in groupoids

This tension is solved in [12] by quotienting the 2-cells under an ad-hoc equivalence relation. In this paper, we advocate that a more satisfactory approach consists in switching from polynomial functors in sets to polynomial functors in groupoids: intuitively, the problem comes from the fact that, $\mathbb{N}$ being a set, there is no non-trivial endomorphism on a natural number, which we should have if we were to have a closure for the cartesian product. By switching to groupoids, we will be able to replace the set $\mathbb{N}$ in the above construction by the groupoid $\mathbf{B}$, which also has the natural numbers as objects, whose morphisms are all automorphisms, such that the group of automorphism on an object n is the n-th symmetric group.

A proper definition of polynomials in groupoids requires more than simply considering diagrams of the form (1) in the category of groupoids. For instance, the traditional definition of composition does not immediately extend to those because the category of groupoids is not locally cartesian closed, and thus the right adjoint to the change of base functor Δ_f is not defined for every morphism f, which prevents us from making an immediate generalization of the definition of polynomial functors (a way to address this consists in restricting to polynomials (1) where p and t are fibrations [22]). Following [16], this can be explained as the fact that switching from sets to groupoids can be thought of as switching from 0-truncated spaces to 1-truncated spaces, where the strict universal limits and colimits involved in the constructions on polynomials are not the right ones: we need to take limits and colimits *up to homotopy*. An important byproduct of working in homotopy type theory as we do is that all internal constructions are invariant up to homotopy, which avoids us explicitly dealing with those issues.

3 The bicategory of polynomial functors

We now present our formalization in homotopy type theory of the main steps for constructing the bicategory of polynomial functors. To be precise, we formalize here the sub-bicategory of the usual one, where we only keep invertible 2-cells (for which the problem for defining the closure is still non-trivial). The developments have been performed in Agda, and are available in the repository [4] based on our own formalization of homotopy type theory (or HoTT), following the reference book [21], to which we suppose the reader already acquainted.

We unfortunately do not have enough space here to expose in details the basic definitions of Agda and homotopy type theory, and only recall some notations. We write `Type` for the universe of small types and $\texttt{Type}_1$ for the universe of large types (in particular `Type` is an element of $\texttt{Type}_1$). We write $x \equiv y$ for the type of *identities* (or *equalities* or *paths*) between two terms x and y of the same type. We write $A \simeq B$ for the type of *equivalences* between two types A and B (possibly in different universes): it consists of functions $f : A \to B$ admitting an inverse up to homotopy, in a suitably coherent sense. We recall that a *proposition* is a type in any two elements are equal, and a *set* (resp. a *groupoid*) is a type in which the type $x \equiv y$ of identities between two elements x and y is a proposition (resp. a set). We postulate here the *univalence* axiom which states that the canonical map from identities $x \equiv y$ to equivalences $x \simeq y$ is itself an equivalence.

3.1 Formalizing polynomials

Our formalization of polynomials can be found in [4, `Polynomial.agda`]. The direct translation of the definition (1) of polynomials, as consisting of two types E and B and three functions t, p and s turns out to be quite cumbersome, because it involves quite a lot of manipulations of identities, even for the basic constructions of the category of polynomials. In the light of the equivalence (2), it is much more convenient to take the

family point of view instead of the slice one, and use the following definition, also known in the literature as an *indexed container* [2]. Namely, in a polynomial (1), the function $t : B \to J$ associates to each operation of the polynomial in B a color in J: this data can equivalently be encoded as a family of types `Op : J → Type` which to every element `j` of `J` associates the type `Op j` of operations colored by `j` of the polynomial. By performing similar transformations on the rest of the data, we reach the following definition, which is easier to work with because it uses more heavily dependent types: the explicit computations we had to perform with equality above are now implicitly handled by the dependent pattern matching of Agda.

Definition 3.1 The type of *polynomials* between two types `I` and `J` is

```
record Poly (I J : Type) : Type₁ where
  field
    Op : J → Type
    Pm : (i : I) → {j : J} → Op j → Type
```

Given types `I` and `J`, a polynomial from `I` to `J` consists of:

- a family of types `Op j` indexed by the elements `j` of `J`: the *operations* of the polynomial;
- a family of type `Pm i b` indexed by the element `i` of `I` and the operations `b` in `Pm j` for some `j` of `J` (curly brackets indicate that this argument is usually left implicit): the *parameters* of the polynomial.

The careful reader will note that the above actually defines "polynomials in types", where there is a type (as opposed to a set or a groupoid) of operations and parameters. The notion of *polynomial in groupoids* can be obtained by further restricting to the case where all the involved types (`I`, `J`, `Op j` and `Pm i b`) are groupoids, in the sense of HoTT recalled above.

3.2 First constructions on polynomials

The identity polynomial on a type `I` is defined as

```
Id : Poly I I
Op Id i = ⊤
Pm Id i {j = j} tt = i ≡ j
```

and has, for each element `i` of `I`, exactly one operation of type `i`, whose only parameter is also of type `i`. The polynomial functor induced by a polynomial `P` from `I` to `J` is

```
⟦_⟧ : Poly I J → (I → Type) → (J → Type)
⟦_⟧ P X j = Σ (Op P j) (λ c → (i : I) → (p : Pm P i c) → (X i))
```

Finally, the composite `P · Q` of two polynomials `P` from `I` to `J`, and `Q` from `J` to `K` is

```
_·_ : Poly I J → Poly J K → Poly I K
Op (P · Q) = ⟦ Q ⟧ (Op P)
Pm (P · Q) i (c , a) = Σ J (λ j → Σ (Pm Q j c) (λ p → Pm P i (a j p)))
```

Morphisms between polynomials can be encoded as follows:

Definition 3.2 The type of *morphisms* between two polynomials `P` and `Q` is

```
record Poly→ (P Q : Poly I J) : Type where
  field
    Op→ : {j : J} → Op P j → Op Q j
    Pm≃ : {i : I} {j : J} {c : Op P j} → Pm P i c ≃ Pm Q i (Op→ c)
```

A morphism thus consists of:

- a morphism between the operations of `P` and those of `Q`, which respects the typing;
- a morphism between the parameters of `P` and those of `Q`, which respects typing and operations, and is moreover an equivalence: this last requirement corresponds precisely to imposing that the square in the middle of (3) is a pullback.

We write `I ⇝ J` (resp. `P ⇝₂ Q`) for the type of polynomials from `I` to `J` (resp. morphisms of polynomials from `P` to `Q`).

Definition 3.3 A morphism `φ` as above is an *equivalence of polynomials* when the morphism on operations is an equivalence at each element of `J`:

```
Poly-equiv : {P Q : I ⇝ J} (φ : P ⇝₂ Q) → Type
```

```
Poly-equiv φ = {j : J} → is-equiv (Op→ φ {j = j})
```

We write `P ≃₂ Q` for the type of equivalences of polynomials between `P` and `Q`.

3.3 A bicategory of polynomials

Starting from there we can build all the structure one expects to find in a bicategory of polynomials:

- we can define the identity polynomial and the composition of polynomials (see above);
- we can show that composition of polynomials is associative and unital up to an equivalence of polynomials;
- we can define the horizontal and vertical composition of morphism of polynomials;
- we can show that those compositions are associative and unital up to a suitable notion of equivalence of morphisms of polynomials.

Moreover, by using univalence, one can show the following.

Proposition 3.4 *The type of equivalences between two polynomials* `P` *and* `Q` *is equivalent to the type* `P ≡ Q` *of equalities between the two polynomials:* `(P ≡ Q) ≃ (P ≃₂ Q)`.

We can therefore build a bicategory (it might be more accurate to call it a $(2,1)$-*category* since 2-cells are equivalences) of groupoids, polynomials in groupoids and equivalences, in the following sense.

Definition 3.5 A *prebicategory* consists of:

- a type of `ob` *objects*;
- for each objects `I` and `J`, we have a groupoid `hom I J` of *morphisms*;
- for each object `I` there is a distinguished morphism `id` in `hom I I` called *identity*;
- there is a composition operation

  ```
  _⊗_ : hom I J → hom J K → hom I K
  ```

  for every objects `I`, `J` and `K`;
- composition is associative and unital:

  ```
  assoc  : (P : hom I J) (Q : hom J K) (R : hom K L) → (P ⊗ Q) ⊗ R ≡ P ⊗ (Q ⊗ R)
  unit-l : (P : hom I J) → id ⊗ P ≡ P
  unit-r : (P : hom I J) → P ⊗ id ≡ P
  ```
- the traditional pentagon law

  ```
  ap (λ P → P ⊗ S) (assoc P Q R) · assoc P (Q ⊗ R) S · ap (λ Q → P ⊗ Q) (assoc Q R S) ≡
  assoc (P ⊗ Q) R S · assoc P Q (R ⊗ S)
  ```

  and triangle law

  ```
  assoc P id Q · ap (λ Q → P ⊗ Q) (unit-l Q) ≡ ap (λ P → P ⊗ Q) (unit-r P)
  ```

  of bicategories are satisfied for composable morphisms `P`, `Q`, `R` and `S`.

Above, "$\cdot$" denotes the concatenation of paths (or transitivity of equality) and `ap` is a proof that every function is a congruence for equality.

Theorem 3.6 *There is a prebicategory whose objects are groupoids, 1-cells are polynomials in groupoids and 2-cells are equivalences of polynomials.*

The notion of prebicategory generalizes the notion of precategory in HoTT [21, Section 9.1]. As the name suggests, it lacks a property in order to bear the name of a bicategory, similarly to the situation with categories. A morphism `P` in `hom I J` in a prebicategory is an *internal equivalence* when there exists morphisms `Q` and `Q'` both in `hom J I` such that `P ⊗ Q ≡ id` and `Q' ⊗ P ≡ id`, and we write `I ≃' J` for the type of internal equivalences from `I` to `J`. There is a canonical map associating to any equality of type `I ≡ J` between two objects `I` and `J` an internal equivalence from `I` to `J`. A *bicategory* is a prebicategory in which this canonical map is an equivalence, i.e., we have `(I ≡ J) ≃ (I ≃' J)` for every objects `I` and `J`.

Theorem 3.7 *The prebicategory of theorem 3.6 is a bicategory.*

Note that proposition 3.4, in addition to proving the above theorem, allows us to use equalities as 2-cells instead of morphisms in the sense of definition 3.3. Because of this, the usual structure of bicategory which is "missing" from definition 3.5 (e.g. horizontal and vertical composition of 2-cells, the exchange law, etc.) is

automatically present thanks to the general properties of equality. If this was not the case (for instance, if we wanted to consider morphisms of polynomials instead of equivalences as 2-cells), we would have had to use a much more involved notion of bicategory [1].

4 Naive cartesian closed structure

4.1 Cartesian structure

The notion of being cartesian for such a bicategory can be formalized in the expected way, by requiring the existence of a terminal object `T`, a binary product operation $\oplus$ on objects and projection operations `projl : hom (I ⊕ J) I` and `projr : hom (I ⊕ J) J` for every objects `I` and `J`, such that `hom I T` $\equiv \top$ and the canonical function `hom I (J ⊕ K) → hom I J × hom I K` (obtained by post-composition with the projection operations) is an equivalence (and thus an equality by univalence). One can show:

Theorem 4.1 *The bicategory of theorem 3.7 is cartesian, with the product being defined by coproduct $\sqcup$ on objects (the groupoids), first projection polynomial and pairing operation on polynomials being*

```
projl : (I ⊔ J) ⇝ I
Op projl i = ⊤
Pm projl (inl i) {i'} tt = i ≡ i'
Pm projl (inr j) {i'} tt = ⊥
```

```
pair : (I ⇝ J) → (I ⇝ K) → I ⇝ (J ⊔ K)
Op (pair P Q) (inl j) = Op P j
Op (pair P Q) (inr k) = Op Q k
Pm (pair P Q) i {inl j} c = Pm P i c
Pm (pair P Q) i {inr k} c = Pm Q i c
```

(and second projection is similar to first projection).

4.2 Naive closed structure

A first step toward constructing a right adjoint to the product is the formalization of the naive closure described in section 2.3 given in [4, `LargePolynomial.agda`]. Namely, the formula (4) indicates that the hom space from `I` to `J` should be `(I → Type) × J`. Of course, we encounter the same size issues as mentioned in the introduction, and we need to suppose that `Type` is the same as $\texttt{Type}_1$ (i.e., we disable the checking of universe levels), which makes the logic inconsistent, for this proof to go through. The formalization is still useful because it is a simple version of the actual one for the closure, which is more involved but does not require the extra assumption.

We define the *exponential* of a type as

```
Exp : Type → Type₁
Exp I = I → Type
```

so that the internal hom between two types `I` and `J` should be `Exp I × J`. We can indeed define a currying map

```
curry : (I ⊔ J) ⇝ K → I ⇝ (Exp J × K)
Op (curry P) (jj , k) = Σ (Op P k) (λ c → ((λ j → Pm P (inr j) c) ≡ jj))
Pm (curry P) i c = Pm P (inl i) (fst c)
```

which formally transforms an operation with a given set of inputs in `J` into an operation with corresponding family of elements in `J` as output (the inputs in `I` are preserved and the output in `K` is preserved as the second component of the output). This could be graphically pictured as

Similarly, one can also define an uncurrying map

```
uncurry : I ⇝ (Exp J × K) → (I ⊔ J) ⇝ K
Op (uncurry P) k = Σ (Exp J) (λ jj → Op P (jj , k))
Pm (uncurry P) (inl i) (jj , c) = Pm P i c
Pm (uncurry P) (inr j) (jj , c) = jj j
```

Theorem 4.2 *The above maps induce an adjunction: we have* `((I ⊔ J) ⇝ K) ≡ (I ⇝ (Exp J × K))`.

Proof We can show that the two above maps are mutually inverse in the sense that we have

`(uncurry (curry P)` $\simeq_2$ `P)`

and

`(curry (uncurry P)` $\simeq_2$ `P)`

By proposition 3.4, the equivalences of polynomials $\simeq_2$ can be turned into equalities: `curry` and `uncurry` thus form an equivalence, and we deduce the required equality by univalence. □

An alternative definition for the "large exponential" can be given as follows. The equivalence (2) between slices and families generalizes in type theory. Given a type `I`, we have an equivalence (and thus an equality by univalence) between types over `I` and families of types indexed by `I`, see [4, `Fam.agda`]:

Theorem 4.3 *Given a type* `I`*, we have the equivalence* `(I → Type) ≃ (Σ Type (λ A → A → I))`.

This indicates that we could equivalently have taken the right member of the above equivalence as a definition of `Exp I` above.

5 Finite types

Following the plan of section 2.3 for the proof, we will restrict to finitary polynomials in order to have a smaller exponential in the closure, for which we can handle the size issues. In this section, we first formalize the notion of finiteness for a type, which will then be used to define finitary functors.

5.1 Definition and properties

The proofs associated to this section can be found in [4, `FinType.agda`]. As customary, we write `Fin n` for the canonical type with n elements, its constructors being the natural numbers 0 up to $n-1$. Given a type `A`, we write `‖ A ‖` for its *propositional truncation*: an element of this type can be thought of as a witness that there exists a proof of `A`, without explicitly providing such a proof [21, Section 3.7].

Definition 5.1 A type is *finite* when it is merely equivalent to the type with n elements for some natural number n:

```
is-finite : Type → Type
is-finite A = Σ ℕ (λ n → ‖ A ≃ Fin n ‖)
```

Remark 5.2 In case one wonders why we chose to use a truncation in the above definition, let us mention that using the definition `is-finite' A = Σ ℕ (λ n → A ≃ Fin n)` would be bad because `is-finite' A` is not a proposition, `is-finite' A` is a large type, and the collection of all finite types is actually the natural numbers in the sense that one can construct an equivalence `Σ Type is-finite'` $\simeq \mathbb{N}$.

The following lemma 5.4 shows that, for a finite type `A`, there is a well-defined notion of *cardinality* which is the natural number `n` such that `‖ A ≃ Fin n ‖` holds. In order to show it, we first need an auxiliary result.

Lemma 5.3 *The type constructor* `Fin` *is injective:* `Fin m ≡ Fin n` *implies* `m ≡ n`.

Proof Given types `A` and `B`, a function `f : ⊤ ⊔ A → ⊤ ⊔ B` induces a function `f' : A → B` such that `f' a` is `f a` if it belongs to `B`, or `f tt` otherwise. Using this construction, one can show that `⊤ ⊔ A ≃ ⊤ ⊔ B` implies `A ≃ B`, and thus that `⊤ ⊔ A ≡ ⊤ ⊔ B` implies `A ≡ B` by univalence. We then conclude by induction using the fact that `Fin (suc n) ≡ ⊤ ⊔ Fin n`. It is also possible to prove this fact without resorting to univalence [14]. □

Lemma 5.4 *Given a type* `A`*, if* `‖ A ≃ Fin m ‖` *and* `‖ A ≃ Fin n ‖` *then* `m ≡ n`.

Proof The equality between natural numbers is a proposition ($\mathbb{N}$ is a set) and we can thus forget about the proposition truncations (by using the corresponding elimination rule). By transitivity, we should have `Fin m ≃ Fin n`, thus `Fin m ≡ Fin n` by univalence and thus `m ≡ n` by injectivity of the type constructor `Fin` by lemma 5.3. □

Using this, one can deduce that being finite is a proper predicate:

Proposition 5.5 *Being finite for a type is a proposition.*

Proof Suppose given two proofs of `is-finite A` for some type `A`. Those have the same first component by lemma 5.4 and the same second component by definition of the propositional truncation. They are thus equal. □

Remark 5.6 As a corollary of previous lemma, we get the fact that `is-finite A` is equivalent to `‖ Σ ℕ (λ n → A ≃ Fin n) ‖`, which could thus have served as an alternative definition.

Finite types satisfy the expected basic properties:

Proposition 5.7 *The types ⊥, ⊤ and* `Fin n` *are finite. Finite types are closed under (dependent) sums and products.*

Proposition 5.8 *Given equivalent types* `A` *and* `B`*,* `A` *is finite if and only if* `B` *is.*

Proposition 5.9 *Given types* `A` *and* `B`*, we have that* `A ⊔ B` *is finite if and only if both* `A` *and* `B` *are finite.*

Proof Suppose that `A ⊔ B` is finite. It can be shown that a decidable subtype of `Fin n` is of the form `Fin k`. More precisely: given a family `P : Fin n → Type` which is a predicate (i.e. `P i` is a proposition for every `i` in `Fin n`) and decidable (i.e. we have `(P i) ⊔ ¬ (P i)`), then there exists a natural number `k` such that `Σ (Fin n) P ≃ Fin k`. By proposition 5.8, we can deduce that a decidable subtype of a finite type is finite. Writing `split : A ⊔ B → Bool` for the function sending the elements of `A` to `true` and the elements of `B` to `false`, we have that `A` is equivalent to the subtype of `A ⊔ B` determined by the predicate `P : A ⊔ B → Type` defined by `P x = split x ≡ true`, which is decidable (because `Bool` has a decidable equality). We conclude that `A` is finite, as a decidable subtype of `A ⊔ B`, which is supposed to be finite (and `B` is also finite for similar reasons).

Conversely, if `A` and `B` are finite, writing `m` and `n` for their cardinal, we have that `A ≃ Fin m` and `B ≃ Fin n`, and thus `A ⊔ B ≃ Fin m ⊔ Fin n ≃ Fin (m + n)` (we can ignore the propositional truncation because we are eliminating into the proposition of being finite). □

Proposition 5.10 *Finite types are sets.*

Proof Suppose given a finite type `(A , (n , F))`, where `A` is a type, `n` is a natural number and `F` a proof of `‖ A ≃ Fin m ‖`. Being a set is a proposition [21, Lemma 3.3.5], by elimination of propositional truncation we can thus extract from `F` and equivalence `A ≃ Fin n`, and we conclude by transporting the fact that `Fin n` is a set (it is easily shown to be decidable and thus a set by Hedberg's theorem [21, Theorem 7.2.5]). □

We write `FinType` for the type of all finite types:

```
FinType : Type₁
FinType = Σ Type is-finite
```

Note that this type is a large one, it lives in $\mathtt{Type}_1$. By proposition 5.5, two elements of this type are equal if and only if their first components (i.e. underlying types) are equal:

Lemma 5.11 *Given elements* `A` *and* `B` *of* `FinType`*, we have* `(A ≡ B) ≃ (fst A ≡ fst B)`.

From this, we can deduce:

Proposition 5.12 `FinType` *is a groupoid.*

Proof Given type finite types `A` and `B`, the type `A ≡ B` is equivalent to the type of equalities between the underlying types of `A` and `B`, which is a set by proposition 5.10 as the types of equalities between two sets. □

5.2 A small axiomatization of the type finite types

In this section, we show an important property: the type `FinType` of finite types is equivalent to a small type [4, `Bij.agda`]. This is a generalization of the following simple observation: every finite set is isomorphic to a set of the form $[n] = \{0, \dots, n-1\}$ for some natural number n, so that the class of finite sets is equivalent to the set of natural numbers. However, `FinType` is a groupoid and not a set, in the sense of HoTT: there are non-trivial equalities between finite types which, by univalence, correspond to isomorphisms of finite types. For this reason, we do not expect that the type `FinType` is equivalent to the traditional type $\mathbb{N}$ of natural numbers, which has no non-trivial path between its element (it has decidable equality and thus is a set by Hedberg's theorem), but rather to a type that we call here $\mathbb{B}$, which has natural numbers as objects, but is moreover such that the group of path endomorphisms on an object `n` is the symmetric group on `n` elements. A more accurate picture of the situation than the equivalence between finite sets and natural numbers is thus the equivalence between the category **Bij** of finite sets an bijections and its skeleton, which has natural numbers as objects. Again, it is important to remark that **Bij** is not small (its collection of objects does not form a set) whereas its skeleton is.

The type $\mathbb{B}$ being constructed from constructors corresponding to the natural numbers, but also paths, it is natural to describe it as a higher inductive type [21, Section 6]. Those are not readily available in current plain version of Agda, but they can be simulated by working axiomatically with them, as done usually. The

definition we give here is close to the one performed in [18] in order to define Eilenberg-MacLane spaces in homotopy type theory.

Definition 5.13 We axiomatize $\mathbb{B}$ as the small type such that

- it has natural numbers as objects:
  `obj : ℕ → 𝔹`
- for every equivalence between finite sets there is a path in $\mathbb{B}$ between the corresponding natural numbers:
  `hom : {m n : ℕ} (α : Fin m ≃ Fin n) → obj m ≡ obj n`
- the path associated to identity is the identity:
  `id-coh : (n : ℕ) → hom {n = n} ≃-refl ≡ refl`
- paths are compatible with composition:
  ```
  comp-coh : {m n o : ℕ} (α : Fin m ≃ Fin n) (β : Fin n ≃ Fin o) →
             hom (≃-trans α β) ≡ hom α · hom β
  ```
- there are no higher paths, i.e., $\mathbb{B}$ is a groupoid:
  `𝔹-is-groupoid : is-groupoid 𝔹`

We also need to postulate an appropriate elimination principle, which can be found in the formalization: it roughly states that, in order to define function of type `f : 𝔹 → A`, for some groupoid `A`, it is enough to define:

- an element `f (obj n)` of `A` for every natural number `n`,
- a path `apd f (hom e) : f (obj m) ≡ f (obj n)` for every equivalence `e : Fin m ≃ Fin n`,

in suitably coherent way. The slightly more readable non-dependent version of this elimination principle is

```
𝔹-rec : (A : Type) → is-groupoid A →
        (obj* : ℕ → A) →
        (hom* : {m n : ℕ} → Fin m ≃ Fin n → obj* m ≡ obj* n) →
        (id-coh* : (n : ℕ) → hom* {n = n} ≃-refl ≡ refl) →
        (comp-coh* : {m n o : ℕ} (α : Fin m ≃ Fin n) (β : Fin n ≃ Fin o) →
                     hom* (≃-trans α β) ≡ hom* α · hom* β) →
        𝔹 → A
```

Finally, we also need to postulate two computation rules (implemented as rewriting rules). For simplicity we indicate the non-dependent versions here. Given a function `f : 𝔹 → A` obtained by applying `𝔹-rec` to some arguments with the above notations, we have

- `f (obj n) = obj* n` for any natural number `n`,
- `ap f (hom α) = hom* α` for any equivalence `α : Fin m ≃ Fin n`.

Remark 5.14 In the definition of `hom`, the reader might be surprised that we allow two different cardinalities of finite sets: given an equivalence `Fin m ≃ Fin n`, we necessarily have `m ≡ n`. But this "more general" definition simplifies the proofs in practice.

There is a canonical function `𝔹-to-Fin : 𝔹 → Type` which realizes the elements of $\mathbb{B}$ as finite types. More precisely, we define the function `𝔹-to-FinType : 𝔹 → FinType` by using the elimination principle `𝔹-rec` with the following arguments:

- `A` is the type `FinType` of finite types, which is a groupoid by proposition 5.12,
- `obj*` states that `𝔹-to-FinType (obj n) = Fin n` (through the first computation rule),
- `hom*` states that, for an equivalence `α : Fin m ≃ Fin n`, we have `ap 𝔹-to-FinType (hom α)` is the equality `Fin m ≡ Fin n` obtained from `α` by univalence (through the second computation rule).

The function `𝔹-to-Fin` can then be obtained by post-composing `𝔹-to-FinType` with the first projection, i.e. we forget about the proofs of finiteness.

One of the main contributions of this paper is the following theorem:

Theorem 5.15 *The large type of finite types and the above type are equivalent:* `FinType ≃ 𝔹`.

Proof The proof uses the "encode-decode method" introduced to compute the fundamental group of the circle [21, Section 8.1]. Given a natural number `n` and an element `b` of $\mathbb{B}$, we can encode the type of paths of

type `obj n ≡ b` in $\mathbb{B}$ as the type `Code n b` defined by induction on `b`. More precisely, we define the function `Code n : 𝔹 → Set` using `𝔹-rec` with the following arguments:

- `A` is the type `Set` of all sets, which is a groupoid,
- `obj*` states that `Code n (obj m)` should be the type `Fin n ≃ Fin m` of equivalences between `Fin m` and `Fin n`,
- `hom* α` states, given an equivalence `α : Fin m ≃ Fin m'`, that `ap (Code n) α : (Fin n ≃ Fin m) ≡ (Fin n ≃ Fin m')` should be the equality between equivalences induced by post-composition with `α`.

Then, we can define an "encoding" function `e : obj n ≡ b → Code n b` such that `e refl = ≃-refl`, and a "decoding" function `d : Code n b → obj n ≡ b` by a suitable induction on `b`. It can be shown that `d ∘ e` is the identity and that, for `α : Fin m ≃ Fin n`, we have `e (d (hom α)) ≡ α`. This can be used to deduce that the function `ap 𝔹-to-FinType : obj m ≡ obj n → Fin m ≡ Fin n` is an equivalence, a thus that the function `𝔹-to-FinType` is an embedding (by induction). The function `𝔹-to-FinType` is easily shown to be surjective: any finite type `A` is equal to `𝔹-to-FinType (obj n)` where `n` is the cardinality of `A`. We finally deduce that the function is an equivalence since it is both an embedding and surjective, see [21, Theorem 4.6.3]. □

6 A cartesian closed bicategory of finitary polynomials

6.1 Finitary polynomials

Thanks to the notion of finiteness for types, we define in [4, `FinPolynomial.agda`] finitary polynomials following the explanations of section 2.3.

Definition 6.1 A polynomial is *finitary* when, for each operation `c`, the total space of its parameters is finite:

```
is-finitary : (P : I ⇝ J) → Type
is-finitary P = {j : J} (c : Op P j) → is-finite (Σ I (λ i → Pm P i c))
```

Remark 6.2 Another definition of being finitary could be to require that each space of parameters is finite:

```
is-finitary' P = (i : I) {j : J} (c : Op P j) → is-finite (Pm P i c)
```

but one quickly convinces oneself that this notion is not suitable: identity polynomials are not generally finitary in this sense, and being finitary is not stable under composition.

Proposition 6.3 *The cartesian bicategory of theorem 3.7 restricts to a bicategory whose morphisms are finitary polynomials.*

Proof Finitary polynomials can be shown to be stable under the required operations (composition, product) using the closure of finite type under the operations of proposition 5.7. □

By using similar arguments as in section 6.2, the proof of section 4.2 can be refined to show that, ignoring size issues, the cartesian bicategory of finitary polynomials in groupoids admits `Exp I × J` as internal hom, where `Exp` is now defined as

```
Exp : Type → Type₁
Exp I = Σ (I → Type) (λ F → is-finite (Σ I F))
```

i.e., we restrict to families whose total space is finite. Moreover, through the equivalence between families and slices (see theorem 4.3), it can be shown that the above type is equivalent to the one of finite sets over `I`:

Lemma 6.4 *The above definition of the exponential is equivalent to the following one:*

```
Exp : Type → Type₁
Exp I = Σ FinType (λ N → fst N → I)
```

In turn, by theorem 5.15, we have that this type is equivalent to the one of elements of $\mathbb{B}$ over `I`:

Lemma 6.5 *The above definition of the exponential is equivalent to the following one:*

```
Exp : Type → Type
Exp I = Σ 𝔹 (λ b → 𝔹-to-Fin b → I)
```

This is now small type, and thus a reasonable candidate for the right definition of the exponential.

6.2 A cartesian closed bicategory

We can finally show that the cartesian bicategory of finitary polynomials (proposition 6.3) is closed. Before doing so, we first prove a useful lemma:

Lemma 6.6 *Given types* `I` *and* `J` *and a family* `A : (I ⊔ J) → Type`*, we have that the total type* `Σ (I ⊔ J) A` *is finite if and only if both types* `Σ I (A ∘ inl)` *and* `Σ J (A ∘ inr)` *are finite.*

Proof We have that `Σ (I ⊔ J) A` is equivalent to `(Σ I (A ∘ inl)) ⊔ (Σ J (A ∘ inr))` and we conclude by proposition 5.8 and proposition 5.9. □

As suggested in previous section, we define the exponential as

```
Exp : Type → Type
Exp I = Σ 𝔹 (λ b → 𝔹-to-Fin b → I)
```

and finally show our main theorem, by adapting the naive constructions of section 4.2 [4, `FinPolynomial.agda`, `CurryUncurry.agda`, `UncurryCurry.agda`]:

Theorem 6.7 *The cartesian bicategory of finitary polynomials is closed, with* `Exp I × J` *as internal hom from* `I` *to* `J`*.*

Proof We can define a function

```
curry : ((I ⊔ J) ⇝ K) → (I ⇝ (Exp J × K))
```

essentially as in section 4.2 except that we have to turn the output `jj : J → Type` of the currying of an operation `c` into an element of `Exp J`. By lemma 6.6, `jj` is a finite type, and thus induces an element of $\mathbb{B}$ by applying the canonical function `card : FinType → 𝔹` given theorem 5.15 which can serve as first component in `Exp J`, and the second component can be deduced from the proof that `card` is an inverse to `𝔹-to-FinType`. The fact that the resulting polynomial is finite follows from lemma 6.6. Similarly, we can define a function

```
uncurry : (I ⇝ (Exp J × K)) → ((I ⊔ J) ⇝ K)
```

which is again given as in section 4.2, using theorem 5.15 to use elements of the exponential and lemma 6.6 to prove the required finiteness of types. The two functions can be shown to be mutually inverse. Finally, the bijection should be checked to be natural (this last part is not fully formalized yet). □

7 Future work

In this work, we have constructed a cartesian closed bicategory of finitary polynomial functors in groupoids, in the setting of type theory. A natural next step is to translate these constructions in to the traditional set-theoretic setting, in order to ease the comparison with more traditional approaches to polynomial functors. As mentioned in section 2.4, we do not expect this task to be particularly easy.

A second aspect in which we wish to push investigations on this bicategory is its relation with linear logic. Namely, the bicategory of spans in groupoids can be understood as the full subbicategory of polynomials of the form (1) where the morphism p is the identity. This subbicategory is monoidal with the tensor induced on objects by cartesian product of groupoids. The inclusion functor admits a left adjoint which is a strong lax monoidal functor. This allows to model the exponential of linear logic [19] and will be detailed elsewhere. We also expect that the constructions of differential linear logic can be interpreted in the model.

Finally, our model is close to the one of generalized species, which is also a model of (differential) linear logic [5,7] inspired by combinatorial species [13]. We would like to understand the relationship between those two models, which is hinted at in [16, Section 3.9]: one of the main difference is that whereas generalized species are be composed using traditional composition of profunctors, which involves a quotient, homotopy polynomial functors perform a homotopy quotient, and we should be able to obtain the first from the second by suitably discarding homotopical information (i.e., "taking π_0").

On the long term, we finally want to investigate the generalization to polynomials in ∞-groupoids, as first studied in [10]. This would make more transparent the comparison with the type theoretic formalization, but we expect the study of this model to be much more involved on a technical level. The definitions given here should work identically without the hypothesis that types are groupoids, and the resulting polynomials should organize as an ∞-category instead of a bicategory, but there is currently no known definition of ∞-categories inside type theory, preventing us from formally showing this result for now.

References

[1] Ahrens, B., D. Frumin, M. Maggesi and N. van der Weide, *Bicategories in Univalent Foundations*, in: *4th International Conference on Formal Structures for Computation and Deduction*, 2019, pp. 1–17.

[2] Altenkirch, T., N. Ghani, P. Hancock, C. McBride and P. Morris, *Indexed containers*, Journal of Functional Programming **25** (2015).

[3] Arkor, N. and M. Fiore, *Algebraic models of simple type theories: A polynomial approach*, in: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, 2020, pp. 88–101.

[4] Finster, E., M. Lucas, S. Mimram and T. Seiller (2021), https://github.com/smimram/fibred-polynomials.

[5] Fiore, M., *Generalised species of structures: Cartesian closed and differential structure* (2004).

[6] Fiore, M., *Analytic functors between presheaf categories over groupoids*, Theoretical Computer Science **546** (2014), pp. 120–131.

[7] Fiore, M., N. Gambino, M. Hyland and G. Winskel, *The cartesian closed bicategory of generalised species of structures*, Journal of the London Mathematical Society **77** (2008), pp. 203–220.

[8] Fiore, M. P., *Differential structure in models of multiplicative biadditive intuitionistic linear logic*, in: *International Conference on Typed Lambda Calculi and Applications*, Springer, 2007, pp. 163–177.

[9] Gambino, N. and J. Kock, *Polynomial functors and polynomial monads*, in: *Mathematical Proceedings of the Cambridge Philosophical Society*, 1, Cambridge University Press, 2013, pp. 153–192.

[10] Gepner, D., R. Haugseng and J. Kock, ∞-*Operads as Analytic Monads*, arXiv preprint arXiv:1712.06469 (2017).

[11] Girard, J.-Y., *Normal functors, power series and λ-calculus*, Annals of pure and applied logic **37** (1988), pp. 129–177.

[12] Hasegawa, R., *Two applications of analytic functors*, Theoretical Computer Science **272** (2002), pp. 113–175.

[13] Joyal, A., *Foncteurs analytiques et especes de structures*, in: *Combinatoire énumérative*, Springer, 1986 pp. 126–159.

[14] Kidney, D. O., *A small proof that fin is injective*, https://doisinkidney.com/posts/2019-11-15-small-proof-fin-inj.html (2019).

[15] Kock, J., *Notes on polynomial functors*, Manuscript, version (2009).

[16] Kock, J., *Data types with symmetries and polynomial functors over groupoids*, Electronic Notes in Theoretical Computer Science **286** (2012), pp. 351–365.

[17] Kock, J., A. Joyal, M. Batanin and J.-F. Mascari, *Polynomial functors and opetopes*, Advances in Mathematics **224** (2010), pp. 2690–2737.

[18] Licata, D. R. and E. Finster, *Eilenberg-MacLane spaces in homotopy type theory*, in: *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2014, pp. 1–9.

[19] Melliès, P.-A., *Categorical semantics of linear logic*, Panoramas et syntheses **27** (2009), pp. 15–215.

[20] Taylor, P., *Quantitative domains, groupoids and linear logic*, in: *Category Theory and Computer Science*, Springer, 1989, pp. 155–181.

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

[22] Vidmar, J., "Polynomial functors and W-types for groupoids," Ph.D. thesis, University of Leeds (2018).

[23] Weber, M., *Polynomials in categories with pullbacks*, Theory and applications of categories **30** (2015), pp. 533–598.