Slice Nondeterminism

Niels F. W. Voorneveld
Tallinn University of Technology, Estonia

Abstract

This paper studies a technique for describing and formalising nondeterministic functions, using slice categories. Results of a nondeterministic function are modelled by an object of the slice category over the codomain of the function, which is an indexed family over the codomain. Two such families denote the same set of results if slice morphisms exist between them in both directions. We formulate the category of nondeterministic functions by expressing a set of possible results as an equivalence class of objects. If we allow families to use any indexing set, this category will be equivalent to the category of relations. When we limit ourselves to a smaller universe of indexing sets, we get a subcategory which more closely resembles nondeterministic programs. We compare this category with other representations of the category of relations, and see how many properties can be carried over, such as its product, coproduct and other monoidal structures. We can describe inductive nondeterministic structures by lifting free monads from the category of sets. Moreover, due to the intensional nature of the slice representation, nondeterministic processes are easily represented, such as interleaving concurrency and labelled transition systems. This paper has been formalised in Agda.

2012 ACM Subject Classification Theory of computation → Categorical semantics; Theory of computation → Type theory

Keywords and phrases Category theory, Agda, Slice category, Nondeterministic functions, Powerset

Digital Object Identifier 10.4230/LIPIcs.ITP.2023.31

Supplementary Material *Software*: `https://github.com/Voorn/Slice-Nondeterminism` archived at `swh:1:dir:540737f2bdc27c9a2bad796cd6713d39de325e94`

Funding *Niels F. W. Voorneveld*: Supported by the ESF funded Estonian IT Academy research measure (project 2014-2020.4.05.19-0001)

1 Introduction

Nondeterminism slips into many facets of computation, be it a run-time optimiser making decisions based on unknown facts, a machine learning algorithm which outputs preferences from beyond a black box, or dependencies of a program on an ever-changing and chaotic environment. If you want to verify correctness of programs, you may not be able to precisely model every aspect, and will have to embrace the fact that to an extent, a program may behave unpredictably.

In essence, nondeterminism is the potential for a program to produce different results in different runs, even in situations when all known external conditions in both runs appear to be the exact same. These multiple possible results are often gathered in a set, a subset of all plausible results allowed by the program's type. This can be seen as an element of the powerset over plausible results.

$$\{a \in A \mid P(a)\} \qquad \text{with } P \text{ a predicate on } A \tag{1}$$

A subset of a set A is commonly described using a predicate on the set. Given a predicate P on A, we can define the subset of all elements of A satisfying P. This is a fundamental construction in set theory. However, it is not directly suited as a tool for gathering possible results of a program from a constructive perspective. Given a program it is often difficult to find a computable way to test for the fact that a certain result is produced. On the other

hand, given a predicate, it is difficult to find a program which nondeterministically produces the exact collection of results satisfying the predicate. The two concepts seem misaligned computationally.

Instead, a program tends to behave nondeterministically based on some unknown cause, unspecified state, or unpredictable decision procedure. It makes sense to describe this unknown with some indexing set I which captures a set of circumstances, for instance potential states, decisions, or transitions, and a function $f : I \to A$ which associates to each circumstance the result a program will produce.

$$\{f(i) \mid i \in I\} \qquad \text{with } f \text{ a function } I \to A \tag{2}$$

In other words, we define a *family* over A, which is an object in the *slice category* over A. These families should be endowed with a notion of equivalence, as different objects in the slice category could represent the same subset of A. The natural solution here lies in the morphisms of the slice category over A; an object is a subset of another object if there is a morphism between them, and they give the same subset if there are morphisms both ways. The subset property can be shown by *any* morphism, not just the injective ones, since we do not want to differentiate between families with different multiplicities of elements.

A nondeterministic function $f : A \to B$ is given by a map which associates to each argument an indexing set and a function from the indexing set to the set of results. This representation lands it firmly between two other representations of relations: *extensional relations* and *spans*.

$$A \to B \to \mathsf{Set} \qquad A \to \Sigma_{I:\mathsf{Set}}\,(I \to B) \qquad \Sigma_{I:\mathsf{Set}}\,(I \to A \times B) \tag{3}$$

In case of $A \to B \to \mathsf{Set}$, we see Set as a space of propositions, in which we will only care about whether the set is inhabited or not. In the other two statements, we use Set as a universe of indexing sets.

Each instance forms a *bicategory* [7], where a 2-morphism from one to another tells us that the relation modelled by one is a subrelation of the relation modelled by the other. We turn this into a *setoid-enriched category* [12], sometimes called a *$\mathcal{E}$-category* [3, 13], which is a category whose morphisms are endowed with an equivalence relation, and whose categorical equations hold up to that equivalence relation. In each instance, the equivalence on morphisms is given by the symmetrisation of the relation induced by the 2-morphisms; they are equivalent if there is a 2-morphism in both directions. E.g., two relations modelled by $R, S : A \to B \to \mathsf{Set}$ are equivalent if for each $a \in A$, and $b \in B$, $R\,a\,b$ is inhabited if and only if $S\,a\,b$ is inhabited. Similarly, two spans are deemed equivalent[1] if they point out the same set of pairs of A and B. Taking the quotient over the equivalence relation of morphisms in an $\mathcal{E}$-category, we get a category. Each of the three categories gives an equivalent representation of the category of relations Rel.

In this paper, we focus on the approach of the second category. Instead of using Set, we specify a smaller inductive universe of indexing sets $\mathbb{U}$, which is sufficient for modelling many nondeterministic processes. The collection of families over A will form a set if we restrict indexing sets to this universe $\mathbb{U}$, and as such we get an endofunctor on Set. Relative to the aforementioned equivalence relation on families, the endofunctor forms a monad, which acts like the powerset monad. The Kleisli category over this monad is what we call the category of *slice nondeterministic functions* $\mathsf{SNF}_{\mathbb{U}}$, and can be put in the form of $\mathcal{E}$-category.

[1] Not equivalent in the category of spans, but in the sense that they model the same relation.

The category $\mathsf{SNF}_{\mathbb{U}}$ forms a wide subcategory of Rel, since it is a substructure of the second representation of relations given in (3). This category still retains many of the properties of Rel, while deviating from it in ways more in-line with nondeterministic processes. In this paper, we shall explore the properties of $\mathsf{SNF}_{\mathbb{U}}$, see how it models a variety of nondeterministic processes, and we shall briefly compare this approach with the other two approaches from (3). In particular, the category $\mathsf{SNF}_{\mathbb{U}}$ is better at dealing with composition and iteration.

In programming theory, a prevalent way of representing nondeterministic computations is in the form of *algebraic effect* [19]. There we consider nondeterministic choices as operations in the programming language. These choices can be enumerated in the indexing set of the family, encoding how the choices should be resolved. Each index gives us a way of handling the algebraic operations in the sense of *effect handlers* [20].

The resulting denotation using slices is similar to historical models of nondeterminism in *domain theory* [9, 2]. Domains are sets with an additional preorder which captures a notion of computational approximation, and *powerdomains* [18] consider different nondeterministic extensions to the domain. These form more usual models for nondeterministic processes, as for instance done in *guarded powerdomains* [17]. Slice nondeterminism offers an alternative light-weight formalisation approach, allowing us to model a lot of nondeterministic processes.

We use category theory in this work, in particular focussing on emulating properties of the category of relations in our framework (see e.g. [16]). The contributions of this paper can be split into three parts; the presentation of a novel model for nondeterministic functions in terms of slices, an exploration of the properties of the associated category, and examples of concepts and processes which it can capture with relative ease.

This paper is formalised in Agda: `https://github.com/Voorn/Slice-Nondeterminism`. See in particular the `ITP-paper.agda` file for links to terms corresponding to the definitions and results of this paper. Most of the code uses only the Agda standard library, though there are some optional files linking the results to the Agda categories library.

2 Powerset via the Slice Category

The main idea is to use a family over A to represent a subset of A. Such a family consists of an indexing set I together with a function $h : I \to A$, and is used to represent the subset $\{a \in A \mid \exists i \in I.\, h(i) = a\}$. It is useful to put a limit on what kind of set I can be. To this end, we specify a universe of sets $\mathbb{U}$.

2.1 A Universe for Indexing Sets

We could simply take $\mathbb{U}$ to be the universe of sets itself. There are some drawbacks to this. The first being that the collection of families using any indexing set does not form a set itself, and hence cannot be used as a basis for an endofunctor in the category of sets. Secondly, the more indexing sets we allow the more cases we need to check when formalising general properties about them. Thirdly, nondeterministic programs do not tend to have the abilities that models using arbitrary sets have. For instance, a program cannot in general nondeterministically produce any result from its codomain, nor can it decide to produce results based on incomputable tests (like equality checking) on its argument.

Instead, we construct an inductive universe of sets which is a set itself, and is closed with respect to a few chosen constructions we would like to use.

▶ **Definition 1.** *The universe of sets $\mathbb{U}$ are defined inductively as:*

$$\frac{}{\bot : \mathbb{U}} \quad \frac{}{\top : \mathbb{U}} \quad \frac{}{\mathbb{N} : \mathbb{U}} \quad \frac{A, B : \mathbb{U}}{A \to B : \mathbb{U}} \quad \frac{A, B : \mathbb{U}}{A \uplus B : \mathbb{U}} \quad \frac{A, B : \mathbb{U}}{A \times B : \mathbb{U}} \quad \frac{A : \mathbb{U} \qquad f : A \to \mathbb{U}}{\Sigma_{a:A}\, f(a) : \mathbb{U}} \quad \frac{A : \mathbb{U} \qquad f : A \to \mathbb{U}}{\Pi_{a:A}\, f(a) : \mathbb{U}}$$

Here $\bot$ is the empty set, $\top$ the single element set, and $\mathbb{N}$ the natural numbers. $A \to B$ is the function space, as specified by the ambient logic (e.g. Agda functions), $A \uplus B$ is disjoint union, and $A \times B$ the Cartesian product on sets. Lastly, $\Sigma_{a:A} f(a)$ is a sigma type whose elements consists of pairs $(a : A, b : f(a))$, and $\Pi_{a:A} f$ a pi type whose elements are dependent functions $(a : A) \to f(a)$. It should be noted that a large portion of the development up to Section 5 works without function spaces. Hence the universe could be limited further, effectively only using *enumerable sets*. For elements of product sets and sigma sets, we write proj_1 and proj_2 for the projections into the first and second components respectively, and for elements of sum sets, we write inj_1 and inj_2 for the first and second injections respectively.

Note that with the inclusion of Sigma types and Pi types in $\mathbb{U}$, the property of whether a set of $\mathbb{U}$ is inhabited is not decidable, since Sigma and Pi types function as existential and universal quantifiers. So in the representation of nondeterministic function, it may not be decidable whether the function outputs a result.

In formalisation, $\mathbb{U}$ is simply a collection of names which each is given a denotation according to the above definitions. For simplicity, we shall write results in this paper directly in terms of the indexing sets, using the closure properties of $\mathbb{U}$ specified above.

▶ **Definition 2.** *The set of $\mathbb{U}$-families over A, denoted $\mathsf{SL}_{\mathbb{U}}\, A$ is given by: $\Sigma_{I:\mathbb{U}} (I \to A)$.*

Hence, a family $S = (I, m) \in \mathsf{SL}_{\mathbb{U}}\, A$ is given by an indexing set $I : \mathbb{U}$ together with a function $m : I \to A$. In other words, it specifies a domain which we may use for indexing elements of A, and a function which associates to each index a result. For a family S, we denote S_I and S_m for the associated indexing set and map respectively.

In the conclusion we will also consider other universes of indexing sets C, and consider families SL_C using those. However, the main development focusses on using $\mathbb{U}$. With this universe, we know that $\mathsf{SL}_{\mathbb{U}}\, A$ is in fact also a set. Hence, the construction is entirely contained within the universe of sets, and $\mathsf{SL}_{\mathbb{U}}$ can be made into an endofunctor on Set. As a functor, it sends a function $f : A \to B$ to a function on families $\mathsf{SL}_{\mathbb{U}}(f) : \mathsf{SL}_{\mathbb{U}}\, A \to \mathsf{SL}_{\mathbb{U}}\, B$ sending (S_I, S_m) to $(S_I, f \circ S_m)$. In the next subsection, we shall define an equivalence relation on $\mathsf{SL}_{\mathbb{U}}\, A$ telling us whether two families represent the same set.

2.2 Relations on Families

Though we work towards an alternative construction of relations for describing nondeterministic functions, we still require the use of relations as they are more traditionally represented for reasoning purposes. An *endorelation* $\mathcal{R}$ on X is a predicate $\mathcal{R} : X \to X \to \mathbb{P}$, where $\mathbb{P}$ is a suitable space of propositions, e.g. Set as used in Agda. We write $a\,\mathcal{R}\,b$ to say $\mathcal{R}$ relates a and b. For two relations $\mathcal{R}$ and $\mathcal{S}$ on X, we write $\mathcal{R} \subset \mathcal{S}$ if $a\,\mathcal{R}\,b$ implies $a\,\mathcal{S}\,b$. Given an endorelation $\mathcal{R}$ on X, we write $\overline{\mathcal{R}}$ for the symmetrisation of $\mathcal{R}$, which relates a and b if both $a\,\mathcal{R}\,b$ and $b\,\mathcal{R}\,a$.

We use the theory of *relators* [15, 21] to guide us in building relations on families. We limit ourselves to relators on endorelations, which is a common adaptation.

▶ **Definition 3.** *Given a relation $\mathcal{R}$ on X, we define a relation $\Gamma_{\mathbb{U}}(\mathcal{R})$ on $\mathsf{SL}_{\mathbb{U}}\, X$ by relating (I, a) and (J, b) if there is a function $h : I \to J$ such that $\forall i \in I.\ a(i)\ \mathcal{R}\ b(h(i))$.*

This construction has the following properties:

- If $\mathcal{R}$ is reflexive, then $\Gamma_{\mathbb{U}}(\mathcal{R})$ is reflexive.
- If $\mathcal{R}$ is transitive, then $\Gamma_{\mathbb{U}}(\mathcal{R})$ is transitive.
- If $\mathcal{R} \subset \mathcal{S}$, then $\Gamma_{\mathbb{U}}(\mathcal{R}) \subset \Gamma_{\mathbb{U}}(\mathcal{S})$.
- If $\forall x, y.(x\,\mathcal{R}\,y) \implies (f(x)\,\mathcal{S}\,g(y))$, then $a\ \ \Gamma_{\mathbb{U}}(\mathcal{R})\ b \implies \mathsf{SL}_{\mathbb{U}}(f)(a)\ \ \Gamma_{\mathbb{U}}(\mathcal{S})\ \ \mathsf{SL}_{\mathbb{U}}(g)(b)$.

As a consequence, if $\mathcal{R}$ is an equivalence relation, then $\overline{\Gamma_{\mathbb{U}}(\mathcal{R})}$ is an equivalence relation. Our notion of equality on families $\mathsf{SL}_{\mathbb{U}}\, X$ shall be given by taking the appropriate notion of equality $=$ on X and lifting it to an equivalence relation on $\mathsf{SL}_{\mathbb{U}}\, X$ given by $\overline{\Gamma_{\mathbb{U}}(=)}$. In formalisation, the $=$ will be instantiated by propositional equality $\equiv$. We will first observe some more general properties.

2.3 A Monad on Setoids

We can see $\mathsf{SL}_{\mathbb{U}}$ as an endofunctor on setoids, as well as an endofunctor on sets. A setoid is a pair $(X, \mathcal{R})$ consisting of a set X and an equivalence relation $\mathcal{R}$ on X. A morphism between setoids $(X, \mathcal{R})$ and $(Y, \mathcal{S})$ is a relation preserving function between X and Y. We have the slice function F on setoids sending $(X, \mathcal{R})$ to $(\mathsf{SL}_{\mathbb{U}}\, X, \overline{\Gamma_{\mathbb{U}}(\mathcal{R})})$.

Given a set X, we define a set function $\langle - \rangle_X : X \to \mathsf{SL}_{\mathbb{U}}\, X$ as given by sending x to $(\top, \lambda * . x)$. This forms a natural transformation in Set; given $f : X \to Y$, then $\langle - \rangle_Y \circ f = \lambda x.(\top, \lambda * . f(x)) = \mathsf{SL}_{\mathbb{U}}(f) \circ \langle - \rangle_X$. Moreover, given a relation $\mathcal{R}$ on X, then $a \,\mathcal{R}\, b$ implies $\langle a \rangle \,\Gamma_{\mathbb{U}}\, \langle b \rangle$, hence this forms a natural transformation on setoids as well.

We have a set function $\bigsqcup_X : \mathsf{SL}_{\mathbb{U}}(\mathsf{SL}_{\mathbb{U}}(X)) \to \mathsf{SL}_{\mathbb{U}}(X)$ which does the following. Given a family $(I, a) \in \mathsf{SL}_{\mathbb{U}}(\mathsf{SL}_{\mathbb{U}}(X))$, then for each $i \in I$ we write $(J_i, b_i) = a(i) \in \mathsf{SL}_{\mathbb{U}}(X)$. $\bigsqcup_X$ sends (I, a) to (K, c) where: $K = \Sigma_{i:I}\, J_i$, meaning a pair (i, j) such that $i \in I$ and $j \in J_i$, and $c(i, j) = b_i(j)$. Similar to $\langle - \rangle$, this forms a natural transformation in both set and setoid:

- Given $f : X \to Y$, $\bigsqcup_Y \circ \mathsf{SL}_{\mathbb{U}}(\mathsf{SL}_{\mathbb{U}}(f)) = \mathsf{SL}_{\mathbb{U}}(f) \circ \bigsqcup_X$.
- Given a relation $\mathcal{R}$ on X, $u \,\Gamma_{\mathbb{U}}(\Gamma_{\mathbb{U}}(\mathcal{R}))\, v$ implies $\bigsqcup_X(u) \,\Gamma_{\mathbb{U}}(\mathcal{R})\, \bigsqcup_Y(v)$.

Importantly, we get the following result.

▶ **Proposition 4.** *$(\mathsf{SL}_{\mathbb{U}}, \langle - \rangle, \bigsqcup)$ forms a monad in the category of setoids.*

Together with the results of Subsection 2.5, we see that the monad actually satisfies the axioms of a powertheory on setoids [18], and as such is a candidate for *powersetoid*.

It is difficult to formalise things in the category of setoids, since one needs to prove many coherences and use higher-order rewrite techniques. As such, the previous proposition is as far as we go in this direction. We can however use the result to make similar claims about the category of sets in the next subsection.

2.4 A Kleisli Triple on Sets

As discussed before, we can define an equivalence relation $\equiv^{\mathbb{U}}_X$ on $\mathsf{SL}_{\mathbb{U}}(X)$ as $\overline{\Gamma_{\mathbb{U}}(=)}$. Consider the endofunctor $\mathsf{SL}^{\equiv}_{\mathbb{U}}$ which sends a set X to the quotient $\mathsf{SL}_{\mathbb{U}}(X)/\equiv^{\mathbb{U}}_X$.

Proposition 4 implies that $\mathsf{SL}_{\mathbb{U}}(-)/\equiv^{\mathbb{U}}$ forms a monad in Set. The associated multiplication operation for X is defined on domain $\mathsf{SL}_{\mathbb{U}}(\mathsf{SL}_{\mathbb{U}}(X)/\equiv^{\mathbb{U}})/\equiv^{\mathbb{U}}$. A morphism whose domain is given by a quotient is defined as a morphism invariant under the equivalence relation of the quotient. In formalisation, this means the morphism would need to be equipped with a proof that they are well defined. In the spirit of getting flexibility in defining nondeterministic function, we bypass this necessity by avoiding the use of quotients in the domain of morphisms. Instead, we look at the Kleisli triple corresponding to the monad structure, which is as follows:

- The unit $\langle - \rangle_X : X \to \mathsf{SL}_{\mathbb{U}}(X)$ is as given before.
- The Kleisli lifting $(-)^* : (X \to \mathsf{SL}_{\mathbb{U}}(Y)) \to (\mathsf{SL}_{\mathbb{U}}(X) \to \mathsf{SL}_{\mathbb{U}}(Y))$ sends f to $\bigsqcup_Y \circ \mathsf{SL}_{\mathbb{U}}(f)$.

We see that this satisfies the appropriate properties:

- For $f, f' : X \to \mathsf{SL}_{\mathbb{U}}(Y)$, such that $\forall x.\, f(x) \equiv^{\mathbb{U}}_X f'(x)$, then $\forall a, a' \in \mathsf{SL}_{\mathbb{U}}(X).\, a \equiv^{\mathbb{U}}_X a' \implies f^*(a) \equiv^{\mathbb{U}}_Y f'^*(a')$.

- For all $a \in \mathsf{SL}_{\mathbb{U}}(X)$, $a \equiv^{\mathbb{U}}_X \langle - \rangle^*(a)$.
- For $f : X \to \mathsf{SL}_{\mathbb{U}}(Y)$ and $x \in X$, $f^*(\langle x \rangle) \equiv^{\mathbb{U}}_Y f(x)$.
- For $f : X \to \mathsf{SL}_{\mathbb{U}}(Y)$, $g : Y \to \mathsf{SL}_{\mathbb{U}}(Z)$ and $a \in \mathsf{SL}_{\mathbb{U}}(X)$, $g^*(f^*(a)) \equiv^{\mathbb{U}}_Z (g^* \circ f)^*(a)$.

2.5 Semilattice Structure

Before we continue to defining nondeterministic functions, we look at one last useful structure. We have an order $\sqsubseteq^{\mathbb{U}}$ on families over X given by applying $\Gamma_{\mathbb{U}}$ to the equality relation on X. This is a preorder which implements the subset relation discussed before, and whose symmetrisation gives the equivalence relation $\equiv^{\mathbb{U}}$ on families telling us that the two families model the same subset. Specifically, $(I, u) \sqsubseteq^{\mathbb{U}} (J, v)$ if for any $i \in I$ there is a $j \in J$ such that $u(i) = v(j)$. In other words, the images are subsets, $u(I) \subseteq v(J)$.

Alternatively, we may define a relation $\in^{\mathbb{U}}$ between X and $\mathsf{SL}_{\mathbb{U}}(X)$, which models inhabitance by relating x to S if there is an $i \in S_I$ such that $S_m(i) = x$. This gives rise to an external subset endorelation $\subseteq^{\mathbb{U}}$ on $\mathsf{SL}_{\mathbb{U}}(X)$ which relates S and Z if for any $x \in X$, $x \in^{\mathbb{U}} S \implies x \in^{\mathbb{U}} Z$. It turns out that the relations $\subseteq^{\mathbb{U}}$ and $\sqsubseteq^{\mathbb{U}}$ are equivalent.

Like for powersets, $\sqsubseteq^{\mathbb{U}}$ comes equipped with a join semi-lattice structure [9]. As such, we can see it as a model of *angelic* nondeterminism, or lower powertheory [10].

▶ **Proposition 5.** *For sets A and I, where I is from $\mathbb{U}$, and $f : I \to \mathsf{SL}_{\mathbb{U}}(A)$, there is an element $\bigvee f \in \mathsf{SL}_{\mathbb{U}}(A)$ giving the supremum of $f(I)$ under the order $\sqsubseteq^{\mathbb{U}}$. In other words, $\forall i \in I.\, f(i) \sqsubseteq^{\mathbb{U}} \bigvee f$, and for $S \in \mathsf{SL}_{\mathbb{U}}(A)$, if $\forall i \in I.\, f(i) \sqsubseteq^{\mathbb{U}} S$ then $\bigvee f \sqsubseteq^{\mathbb{U}} S$.*

Proof. We define $\bigvee f$ as $(\bigvee f)_I = \Sigma_{i:I} f(i)_I$ and $(\bigvee f)_m(i, j) = f(i)_m(j)$.

Then for any $i \in I$, $f(i) \sqsubseteq^{\mathbb{U}} \bigvee f$ since for any $j \in f(i)_I$, there is an index $k = (i, j) \in (\bigvee f)_I$ such that $f(i)_m(j) = (\bigvee f)_I(k)$.

Let $(K, u) \in \mathsf{SL}_{\mathbb{U}}(A)$ such that $\forall i \in I.\, f(i) \sqsubseteq^{\mathbb{U}} (K, u)$. Then for any $(i, j) \in (\bigvee f)_I$, since $i \in I$ there is a $k \in J$ such that $f(i)_m(j) = u(k)$, hence there is a $k \in J$ such that $(\bigvee f)_m((i, j)) = f(i)_m(j) = u(k)$. So $\bigvee f \sqsubseteq^{\mathbb{U}} S$. ◀

We distinguish three special kinds of joins.

- Let $!_A = \lambda() : \emptyset \to A$ be the unique function from the empty set to A. We define the *empty join* $\oslash_A \in \mathsf{SL}_{\mathbb{U}}(A)$ as $\oslash_A = (\emptyset, !_A)$. This is the smallest element of $\mathsf{SL}_{\mathbb{U}}(A)$, and it holds that $\oslash_A \equiv_A \bigvee !_{\mathsf{SL}_{\mathbb{U}}(A)}$.
- For $U, V \in \mathsf{SL}_{\mathbb{U}}(A)$, we define the *binary join* $U \vee V$ as $(U_I \uplus V_I, \lambda\{\mathsf{inj}_1(i) \mapsto U_m(i), \mathsf{inj}_2(j) \mapsto V_m(i)\})$. This is equivalent to $\bigvee f$ where $f : \{0, 1\} \to \mathsf{SL}_{\mathbb{U}}(A)$ with $f(0) = U$ and $f(1) = V$.
- For $f : \mathbb{N} \to \mathsf{SL}_{\mathbb{U}}(A)$ we can take the *countable join* $\bigvee f$.

Lastly, note that for $f : I \to J \to \mathsf{SL}_{\mathbb{U}}(A)$, $\bigvee(\lambda i.\, \bigvee f(i)) \equiv \bigvee(\lambda(i, j).\, f(i)(j))$. Due to the limitations of our universe of indexing sets, we do not have a complementing meet semilattice structure, which would require us to check for equality of results within indexing sets.

3 Nondeterministic Functions

To model nondeterministic functions, we use the Kleisli category associated to the Kleisli triple defined in Subsection 2.4. Since this is a Kleisli triple relative to an equivalence relation, what we end up constructing is a *Setoid-enriched category*, sometimes called an *$\mathcal{E}$-category*.

3.1 $\mathcal{E}$-categories

In an $\mathcal{E}$-category, we use setoids to describe collections of morphisms. A common interpretation of this is to see the setoid relation as a 2-morphism between these morphisms, creating a *bicategory* whose spaces of 2-morphisms have at most one element. We lay out precisely the conditions of such a structure.

▶ **Definition 6.** *An* $\mathcal{E}$-category $\mathcal{C}$ *is given by:*

- *A set of objects* O*.*
- *For any two objects* $X, Y \in O$*, a set of morphisms* $M(X,Y)$ *equipped with an equivalence relation* $\equiv$*.*
- *For each* $X \in O$ *an identity morphism* $id_X \in M(X,X)$*.*
- *For* $X, Y, Z \in O$*,* $f \in M(X,Y)$*,* $g \in M(Y,Z)$ *a morphism* $f\,g \in M(X,Z)$*.*

such that:

- *For any* $X, Y \in O$*,* $f \in M(X,Y)$*,* $id_X\, f \equiv f \equiv f\, id_Y$*.*
- $\forall X, Y, Z, W \in O$*,* $f \in M(X,Y)$*,* $g \in M(Y,Z)$*,* $h \in M(Z,W)$*,* $(f\,g)\,h \equiv f\,(g\,h)$*.*
- $\forall X, Y, Z \in O$*,* $f, f' \in M(X,Y)$*,* $g, g' \in M(Y,Z)$*, if* $f \equiv f'$ *and* $g \equiv g'$ *then* $f\,g \equiv f'\,g'$*.*

The $\mathcal{E}$-category method matches the approach of the Agda categories library. There, categories are formalised using equivalence relations on morphisms. By taking the quotient over the equivalence relations on the homsets, we get a category. Hence, we may see any $\mathcal{E}$-category as a category as well, and properties on the $\mathcal{E}$-category directly translate to properties on the resulting category. For instance, a functor between $\mathcal{E}$-categories forms a functor between their corresponding categories.

The focus of this paper is the following setoid-enriched category.

▶ **Definition 7.** *The* $\mathcal{E}$*-category of* slice nondeterministic functions*, denoted* $\mathcal{E}\mathsf{SNF}_{\mathbb{U}}$*, is the category consisting of:*

- *Objects are sets.*
- *The set of morphisms between set* A *and* B*, denoted* $A \multimap B$*, are functions* $A \to \mathsf{SL}_{\mathbb{U}}(B)$*. Two morphisms* $f, g : A \multimap B$ *are equivalent, denoted as* $f \sim g$*, if* $\forall a \in A.\, f(a) \equiv^{\mathbb{U}}_{B} g(a)$*.*
- *For a set* A*, the identity morphism is given by* $\langle - \rangle_A : A \multimap A$*.*
- *For two morphisms* $f : A \multimap B$ *and* $g : B \multimap C$*, the composition is given by* $f; g := f\, g^*$*.*

We write $\mathsf{SNF}_{\mathbb{U}}$ for the associated quotient category to the $\mathcal{E}$-category $\mathcal{E}\mathsf{SNF}_{\mathbb{U}}$.

The appropriate properties are satisfied as observed in the previous chapter. For $f : A \multimap B$, remember we may write $f(a)_I$ for the indexing set associated to $f(a)$, and for $i \in f(a)_I$, $f(a)_m(i)$ the element of B associated to i via $f(a)$.

3.2 Functors and Variations

Let us look at some variations on models of nondeterministic functions. First we revisit the category of relations Rel, which can be cast into the form of an $\mathcal{E}$-category as well.

▶ **Definition 8.** *The* $\mathcal{E}$-category of relations*, denoted* $\mathcal{E}\mathsf{Rel}$ *is given by:*

- *Objects are sets.*
- *Morphisms between* A *and* B *are predicates* $\mathcal{R} : A \to B \to \mathbb{P}$*, with equivalence relation:* $\mathcal{R} \equiv \mathcal{S}$ *if* $\forall a \in A, b \in B.\, a\,\mathcal{R}\,b \iff a\,\mathcal{S}\,b$*.*
- *The identity relation is the equality predicate.*
- *Composition is given by:* $a\,\mathcal{R}\mathcal{S}\,c$ *if* $\exists b.\, a\,\mathcal{R}\,b \wedge b\,\mathcal{S}\,c$*.*

Note that the type of propositions $\mathbb{P}$ needs to be flexible enough to contain both equality on sets and existential quantification. In Agda, this is instantiated by Set, with the double implication $\iff$ describing the existence of a function in both directions.

There exists a functor $[\![-]\!]$ from $\mathcal{E}\mathsf{SNF}_{\mathbb{U}}$ to $\mathcal{E}\mathsf{Rel}$, which preserves objects and sends a morphism $f : A \multimap B$ to $[\![f]\!]$ with the following property:

For $a \in A$, $b \in B$, $a \,[\![f]\!]\, b$ holds if and only if there is an $i \in f(a)_I$ such that $f(a)_m(i) = b$.

There is no functor going from Rel to $\mathsf{SNF}_{\mathbb{U}}$, since our universe of indexing sets $\mathbb{U}$ is not large enough to capture the two things that are necessary to formulate such a thing; it would need Sigma types over any set, and equality types on arbitrary sets. If however our universe $\mathbb{U}$ of indexing sets was taken to be Set itself, then Rel and $\mathsf{SNF}_{\mathsf{Set}}$ are equivalent.

In Subsection 3.3 we shall do a qualitative comparison between the two approaches of formalising nondeterministic functions. There we will see that when composing nondeterministic functions, it is easier to verify that the composite gives a certain result when it is formalised in $\mathsf{SNF}_{\mathbb{U}}$ than when it is formalised in Rel or the category of spans.

We look at four properties a slice nondeterministic function $f : A \multimap B$ could satisfy.

- f is *total* if for any $a \in A$ the indexing set $f(a)_I$ is inhabited.
  This is synonymous to saying that $\forall a \in A.\, \exists b \in B.\, a \,[\![f]\!]\, b$.
- f is *deterministic* (single-valued) if for any $a \in A$ and $i, j \in f(a)_I$, $f(a)_m(i) = f(a)_m(j)$.
  This is synonymous to saying that $\forall a \in A, b, b' \in B,\, a \,[\![f]\!]\, b \wedge a \,[\![f]\!]\, b' \implies b = b'$.
- f is *surjective* if for any $b \in B$, there are $a \in A$ and $i \in f(a)_I$ such that $f(a)_m(i) = b$.
  This is synonymous to saying that $\forall b \in B.\, \exists a \in A.\, a \,[\![f]\!]\, b$.
- f is *injective* (sometimes called modest) if $\forall a, a' \in A.\, i \in f(a)_I$, and $j \in f(a')_I$, if $f(a)_m(i) = f(a')_m(j)$ then $a = a'$.
  This is synonymous to saying that $\forall a, a' \in A, b \in B,\, a \,[\![f]\!]\, b \wedge a' \,[\![f]\!]\, b \implies a = a'$.

All four properties are satisfied by the identity morphism of $\mathsf{SNF}_{\mathbb{U}}$, are preserved by composition and invariant under equivalence of morphisms. Hence, any subset of properties specifies a *wide subcategory* of $\mathsf{SNF}_{\mathbb{U}}$.

If a morphism is surjective and deterministic, it is an epimorphism. If a morphism is total and injective, it is a monomorphism. Lastly, the wide subcategory of total and deterministic morphisms is equivalent to the category Set of sets.

3.3 Method Comparison

We can compare the method of formalising nondeterministic processes using $\mathsf{SNF}_{\mathbb{U}}$ with other representations of the category of relations. We do this with code from the Agda proof assistant. Let us consider a particular example. Suppose we toss five coins, and want to prove that it is possible to get three heads. So, starting with zero, if we nondeterministically add zero heads or one head a total of five times, it should be possible to get three heads. We compare extensional relations, spans and slice nondeterminism, and we first define a nondeterministic function on $\mathbb{N}$ representing a coin toss.

```
Erel-ℕ-toss : ℕ → ℕ → Set
Erel-ℕ-toss n m = (n ≡ m) ⊎ (suc n ≡ m)

Span-ℕ-toss : Σ Set λ I → I → ℕ × ℕ
Span-ℕ-toss = (ℕ × Bool) , λ {(n , false) → n , n ;
    (n , true) → n , (suc n)}

Slic-ℕ-toss : ℕ → Σ Set λ I → (I → ℕ)
Slic-ℕ-toss n = Bool , (λ { false → n ; true → suc n})
```

Note that though we use Set in the slice example, the indexing set is from $\mathbb{U}$. In each case, we construct a multi-toss operation inductively, by composing the identity relation with a number of toss operations specified by the argument. We look at the examples, and prove that 5 tosses may get 3 heads, using a sequence of two tails and three heads.

```
Erel-ℕ-example : Erel-ℕ-test (Erel-ℕ-multi-toss 5) 0 3
Erel-ℕ-example = 0 , (inj₁ refl , 0 , (inj₁ refl) , 1 , (inj₂ refl) ,
2 , (inj₂ refl) , 3 , (inj₂ refl) , refl)

Span-ℕ-example : Span-ℕ-test (Span-ℕ-multi-toss 5) 0 3
Span-ℕ-example = (((0 , false) , ((0 , false) , ((0 , true) , ((1 , true) ,
((2 , true) , 3) , refl) , refl) , refl) , refl) , refl) , refl

Slic-ℕ-example : Slic-ℕ-test (Slic-ℕ-multi-toss 5) 0 3
Slic-ℕ-example = (false , false , true , true , true , tt) , refl
```

In the first method, since composition is defined using existential quantification on the intermediate argument, we need to specify all the intermediate results in the sequence. So, after one toss we can keep 0, given that we choose a tail signified by the inj_1. Then still 0 after another tail, followed by 1, 2 and 3, each with a heads result signified by inj_2. For spans, a similar proof is necessary, though we specify transitions using both the input state and a truth value denoting the result of the coin toss. Though in a slightly different way, both endorelations and spans need the same information in their proof.

Only in the slice nondeterministic proof do we neither need to specify intermediate states, nor have to check for equality at each step. The proof there is simply a sequence of choices with a single final verification that it gives the right result. This example illustrates the relative ease of using slice nondeterminism in these situations.

Many nondeterministic processes are results of compositions of many subprocesses. Therefore, slice nondeterministic functions are particularly well suited to capture such processes. In the rest of the paper, we will see that it moreover still retains a lot of the useful structures and properties that the category of relations exhibits.

4 Categorical Structures

We look at some of the structures that can be found in $\mathsf{SNF}_{\mathbb{U}}$. Most of these reflect similar structures from the category of relations, though not all can be replicated. In most cases though, this inability to express some structures is more faithful to the limitations of nondeterministic programs compared to relations.

Some of the structures we look at are lifted from the category of sets using the unit $\langle - \rangle$ of $\mathsf{SL}_{\mathbb{U}}$. Explicitly, we have a functor $|-| : \mathsf{Set} \to \mathsf{SNF}_{\mathbb{U}}$ which keeps objects as is, and sends morphisms $f : A \to B$ to $|f| = \lambda x.\langle f(x) \rangle_B : A \multimap B$.

4.1 Morphisms with Daggers

Given a relation $\mathcal{R}$ between A and B, there is a relation $\mathcal{R}^\dagger$ between B and A, called the *dagger* of $\mathcal{R}$, such that $a \mathcal{R} b \iff b \mathcal{R}^\dagger a$. Such a reversing of morphisms does not exist for $\mathsf{SNF}_{\mathbb{U}}$. Though unfortunate, it is arguably not an unreasonable problem to have when thinking of nondeterministic functions as programs.

For a nondeterministic program P of type $\sigma \to \tau$, there is in general not another program of type $\tau \to \sigma$ which for input $V : \tau$ can nondeterministically give any term W of σ such that $P(W)$ may produce V. This is due to two general concerns:

- Equality in τ may not be checkable by a program, hence there may not be a way of verifying whether $P(W)$ can produce V.
- Programs may not have access to enumerations over terms of σ.

Both these restrictions align closely to the limitations of our universe $\mathbb{U}$.

We can however distinguish those slice nondeterministic functions that have a dagger.

▶ **Definition 9.** *$f : A \multimap B$ and $g : B \multimap A$ are each others dagger, denoted $f \dagger g$, if $[\![f]\!]^\dagger = [\![g]\!]$.*

If a morphism has a dagger, we call it *daggerable*. This property of having a dagger is preserved under composition and the equivalence relation on morphisms. Moreover, the identity morphism is a dagger of itself. We conclude that the daggerable morphisms form a wide subcategory of $\mathsf{SNF}_{\mathbb{U}}$.

Note that being daggerable does not mean that the daggered program is an inverse. We can say the following; let f g be eachother's dagger, then:

- f is total if and only if g is surjective.
- f is deterministic if and only if g is injective.
- If f is total and injective, then $f\,g$ is the identity morphism. In other words, g is a post inverse of f.
- If f is deterministic and surjective, then $g\,f$ is the identity morphism. In other words, g is a pre-inverse of f.

There is a slight difference between the notion of daggerability and the notion of *reversibility*. A nondeterministic function is normally called reversible if it is injective (see e.g. [11]). If we were to match this definition, the appropriate category to consider for reversible nondeterministic functions is the category of daggerable injective (maybe total) morphisms.

4.2 Products and Coproducts

Similar to the category of relations, the category of slice nondeterministic functions is both Cartesian and Cocartesian, with both structures mirroring each other exactly as in a *semi-additive category*.

Firstly, the initial and terminal object of the category is given by the empty set.

▶ **Lemma 10.** *For a set A, any two morphisms $\emptyset \multimap A$ are similar, and any two morphisms $A \multimap \emptyset$ are similar.*

Consider the bifunctor for disjoint union $\uplus$ on sets, with injections $\mathsf{inj}_1 : A \to A \uplus B$ and $\mathsf{inj}_2 : B \to A \uplus B$. We can lift $\uplus$ to the category of nondeterministic functions, where for $f : A \multimap B$ and $g : C \multimap D$, $f \uplus g : A \uplus B \multimap C \uplus D$ sends $\mathsf{inj}_1(a)$ to $(f(a)_I, \lambda i.\mathsf{inj}_1(f(a)_m(i)))$ and $\mathsf{inj}_2(b)$ to $(g(b)_I, \lambda i.\mathsf{inj}_2(g(b)_m(i)))$. Consider the following natural transformations.

- $\curlyvee_1 := |\mathsf{inj}_1| : A \multimap A \uplus B$ and $\curlyvee_2 := |\mathsf{inj}_2| : B \multimap A \uplus B$.
- Given $f : A \multimap C$ and $g : B \multimap C$, we define $(f \curlyvee g) : A \uplus B \multimap C$ as given by $(f \curlyvee g)(\mathsf{inj}_1(a)) = f(a)$ and $(f \curlyvee g)(\mathsf{inj}_2(b)) = g(b)$.

The following proposition establishes that $(\uplus, \curlyvee_1, \curlyvee_2)$ forms a coproduct in $\mathsf{SNF}_{\mathbb{U}}$.

▶ **Proposition 11.** *Let $f : A \multimap C$, $g : B \multimap C$, and $h : A \uplus B \multimap C$ such that $\curlyvee_1; h \sim f$ and $\curlyvee_2; h \sim g$, then $h \sim (f \curlyvee g)$.*

Proof. Let $a \in A$, then $(f \between g)(\mathsf{inj}_1(a)) = f(a)$ and $h(\mathsf{inj}_1(a)) = h^*(\langle \mathsf{inj}_1(a) \rangle) = h^*(\between_1(a)) = (\between_1; h)(a)$. Since $\between_1; h \sim f$, $f(a) \equiv^{\mathbb{U}} h(\mathsf{inj}_1(a))$. Similarly, for $b \in B$, $(f \between g)(\mathsf{inj}_2(b)) = g(b) \equiv^{\mathbb{U}} (\between_2; h)(b) = h(\mathsf{inj}_2(b))$, so $h \sim (f \between g)$. ◀

We define $\mathsf{merge}_A = (id_A \between id_A) : A \uplus A \to A$, so it holds that $(f \uplus g); \mathsf{merge}_C = (f \between g)$.

Remember that for each set A, we have a special kind of family called the *empty family* $\oslash_A \in \mathsf{SL}_{\mathbb{U}}(A)$ given by: $\oslash = (\emptyset, \lambda())$.

- $\between_1$ has a dagger $\pi_1 : A \uplus B \multimap A$ defined as: $\pi_1(\mathsf{inj}_1(a)) = \langle a \rangle$ and $\pi_1(\mathsf{inj}_2(b)) = \oslash_A$. Dually, $\between_2$ has a dagger $\pi_2 : A \uplus B \multimap B$ defined as: $\pi_2(\mathsf{inj}_1(a)) = \oslash_B$ and $\pi_2(\mathsf{inj}_2(b)) = \langle b \rangle$.
- Given $f : A \multimap B$ and $g : A \multimap C$, there is a function $(f \pi g) : A \multimap B \uplus C$ given by: $(f \pi g)(a) = (f(a)_I \uplus g(b)_I, \lambda\{\mathsf{inj}_1(i) \mapsto \mathsf{inj}_1(f(a)_m(i)), \mathsf{inj}_2(j) \mapsto \mathsf{inj}_2(g(b)_m(j))\})$.

Note that $\between_1; \pi_1 \sim id_A$ and $\between_2; \pi_2 \sim id_B$. Moreover, if $f^\dagger$ and $g^\dagger$ are daggers of f and g respectively, then $(f^\dagger \between g^\dagger)$ is a dagger of $(f \pi g)$. Now, $(\uplus, \pi_1, \pi_2)$ forms a product in $\mathsf{SNF}_{\mathbb{U}}$:

▶ **Proposition 12.** *Let $f : A \multimap B$, $g : A \multimap C$, and $h : A \multimap B \uplus C$ such that $h; \pi_1 \sim f$ and $h; \pi_2 \sim g$, then $h \sim (f \pi g)$.*

We can define $\mathsf{share}_A : A \multimap A \uplus A$ as $(id_A \pi id_A)$, so $\mathsf{share}_A; (f \uplus g) = (f \pi g)$.

4.3 Semilattice Enriched

As explored in Subsection 2.5, each family $\mathsf{SL}_{\mathbb{U}}(A)$ comes equipped with a join semilattice structure. This can be used to define a join semilattice structure on the spaces of morphisms as well, creating a *semilattice enriched* category [12].

Firstly, we define an order $\prec$ on morphisms $A \multimap B$, given by $f \prec g \iff \forall a \in A.\, f(a) \sqsubseteq^{\mathbb{U}} g(a)$. This is a preorder, and by definition $f \prec g \wedge g \prec f$ implies $f \sim g$. Hence, in $\mathsf{SNF}_{\mathbb{U}}$ as a category (the quotient over the $\mathcal{E}$-category), $\prec$ is antisymmetric. Similar to how $\sim$ is preserved over composition and products, $\prec$ is preserved over the same constructions in the $\mathcal{E}$-category $\mathcal{E}\mathsf{SNF}_{\mathbb{U}}$ and the associated category $\mathsf{SNF}_{\mathbb{U}}$.

We can recover the join operation from Subsection 2.5.

▶ **Corollary 13.** *For A, B and I sets, with I from $\mathbb{U}$, and $F : I \to (A \multimap B)$, there is a supremum $\bigvee F : A \multimap B$ of $F(I)$.*

This can be simply proven by taking $(\bigvee F)(a) = (\Sigma_{i:I} F(i,a)_I,\ \lambda(i,j).\, F(i,a)_m(j))$.

Given $F : I \to (A \multimap B)$ and $G : J \to (B \multimap C)$, where I and J from $\mathbb{U}$, then $(\bigvee F); (\bigvee G) \sim (\bigvee(\lambda(i,j) : I \times J.\, F(i); G(j)))$. Note that for $I = J$, $(\bigvee F); (\bigvee G)$ is not necessarily similar to $\bigvee(\lambda i.\, F(i); G(i))$. To get such a property, we look at ω-chains.

▶ **Definition 14.** *An ω-chain of morphisms is an enumeration of morphisms $F : \mathbb{N} \to (A \multimap B)$ such that for any $n \in \mathbb{N}$, $F(n) \prec F(n+1)$.*

Omega chains are helpful as they allow us to more uniformly use preservation of join over constructions like composition and products. Consider the following results, for example.

- For $F : \mathbb{N} \to (A \multimap B)$ and $G : \mathbb{N} \to (B \multimap C)$ two ω-chains, then $H = \lambda n.F(n); G(n) : \mathbb{N} \to (A \multimap C)$ is an ω-chain and $\bigvee H = (\bigvee F); (\bigvee G)$.
- For $F : \mathbb{N} \to (A \multimap B)$ and $G : \mathbb{N} \to (C \multimap D)$ two ω-chains, then $H = \lambda n.F(n) \uplus G(n) : \mathbb{N} \to (A \uplus C \multimap B \uplus D)$ is an ω-chain and $\bigvee H = (\bigvee F) \uplus (\bigvee G)$.

As before, we can have a specific binary join operation which takes $f, g : A \multimap B$ and produces $f \vee g : A \multimap B$. It holds that $(f \vee g) \sim (\mathsf{share}_A; (f \uplus g); \mathsf{merge}_B)$.

4.4 Monoidal

The Cartesian structure on Rel and $\mathsf{SNF}_{\mathbb{U}}$ is different from the Cartesian structure on Set. Set uses the bifunctor $\times$ which collects pairs of elements, with projections $\mathsf{proj}_1 : A \times B \to A$ and $\mathsf{proj}_2 : A \times B \to B$. We can lift this to the monoidal structure $(\times, |\mathsf{proj}_1|, |\mathsf{proj}_2|)$ on $\mathsf{SNF}_{\mathbb{U}}$ using the functor $|-| : \mathsf{Set} \to \mathsf{SNF}_{\mathbb{U}}$. Similarly, we can lift the comonoid structure on $\times$ in Set as well:

- The unique map $u_A : A \to \{*\}$ is lifted to $|u_A| : A \multimap \{*\}$.
- The map $c_a : A \to A \times A$ which duplicates the argument is lifted to $|c_A| : A \multimap A \times A$.

For any morphism $f : A \multimap B$, $f; |u_B| \prec |u_A|$ and $f; |c_b| \prec |c_a|; (f \times f)$. If f is total, then $f; |u_B| \sim |u_A|$, and if f is deterministic, then $f; |c_b| \sim |c_a|; (f \times f)$.

We cannot equip $\times$ with a monoidal closed structure. The best approximation would be to take $(A \Rightarrow B) = (A \multimap B)$, in which case we can do the following: For $F : A \multimap (B \Rightarrow C)$, let $F' : (A \times B) \multimap C$ be the map $F'(a, b) = (\Sigma_{i:F(a)_I} F(a)_m(i, b)_I,\ \lambda(i, j).\, F(a)_m(i, b)_m(j))$. Only in this direction is the similarity relation preserved. We cannot go into the other direction, since in $(B \Rightarrow C)$ we distinguish between similar but syntactically different families.

5 Inductive Nondeterministic Structures

One of the main applications of the formalisation via slice functions is the relative ease in which it can be used to give a specification for a variety of inductive nondeterministic structures. Suppose in particular we have some inductive structure generated over some signature, for instance generated by algebraic effect operations [19] or a container [1]. Such a signature forms a free monad in the category of sets, which we can lift to $\mathsf{SNF}_{\mathbb{U}}$.

A $\mathbb{U}$-*container* C is an element of the sigma type $\Sigma_{O:\mathsf{Set}}(O \to \mathbb{U})$. It consists of a set of operations O, and a function $ar : O \to \mathbb{U}$ which associates to every operation and arity.

▶ **Definition 15.** *Given a $\mathbb{U}$-container $C = (O, ar)$, the* free monad over C *in Set, is a monad which sends each set A to the set $F_S A$ defined inductively:*

- *For each $a \in A$, there is an element $\mathsf{leaf}(a) \in F_S A$.*
- *For each $\sigma \in O$ and $c : ar(\sigma) \to F_S A$, there is an element $\mathsf{node}_\sigma(c) \in F_S A$.*

It sends a function $f : X \to Y$ to a function $F_S(f) : F_S(X) \to F_S(Y)$ replacing the values at the leaves. The monad unit is given by $\eta^C_A : A \to F_C A$ sending a to $\mathsf{leaf}(a)$, and the monad multiplication $\mu^C_A : F_S F_S A \to F_S A$ is defined inductively as:

- $\mu^C_A(\mathsf{leaf}(t)) = t$.
- $\mu^C_A(\mathsf{node}_\sigma(c)) = \mathsf{node}_\sigma(\lambda i.\mu^C_A(c(i)))$.

We will show that this monad can be lifted to $\mathsf{SNF}_{\mathbb{U}}$ in the next subsection.

5.1 Distributivity

In Set, the free monad distributes over the powerset monad, in the sense that there is a distributivity law between monads [6]. This can be adapted to show that the free monad distributes over the monad $\mathsf{SL}_{\mathbb{U}}$ using a natural transformation $F_S(\mathsf{SL}_{\mathbb{U}} A) \to \mathsf{SL}_{\mathbb{U}}(F_S A)$. We construct this by specifying a set of leaf-positions for each element of $F_S A$, following ideas from the theory of containers [1, 4]. An element $t \in F_S(\mathsf{SL}_{\mathbb{U}} A)$ has at each of its positions a leaf which contains a set of values (a family). If we make a choice of element for each of the leaves of t, we can construct an element of $F_S(A)$. Such a combination of choices is given by associating to each position a choice, which we can do with a dependent map.

We use this idea to lift the Set-endofunctor F_S to an $\mathsf{SNF}_{\mathbb{U}}$-endofunctor. We need a way to transform a morphism $f : A \to \mathsf{SL}_{\mathbb{U}}B$ to $F_SA \to \mathsf{SL}_{\mathbb{U}}(F_SB)$, which accepts a term t as input and replaces each leaf $\mathsf{leaf}(a)$ with some choice value from $f(a)$. We specify a set of indices which collects all possible combinations of choices that can be made.

▶ **Definition 16.** *Given a set A and a function $f : A \to \mathsf{Set}$ assigning a set of choices to each element $a \in A$, we inductively define a function $\mathsf{Pos}(f) : F_SA \to \mathsf{Set}$ assigning to an element $t \in F_SA$ a set of combinations of choices:*

- $\mathsf{Pos}(f)(\mathsf{leaf}(a)) = f(a)$.
- $\mathsf{Pos}(f)(\mathsf{node}_\sigma(c)) = (\Pi_{i:ar(\sigma)}\, \mathsf{Pos}(f)(c(i)))$.

With this function, we can specify our endofunctor on $\mathsf{SNF}_{\mathbb{U}}$, which we call T_C. On objects, T_CA is given by F_CA.

▶ **Definition 17.** *Given a function $f : A \multimap B$, we define $T_Cf : T_CA \multimap T_CB$, with indexing set $T_Cf(t)_I = \mathsf{Pos}(\lambda a.f(a)_I)(t)$ and indexing map where:*

- $T_Cf(\mathsf{leaf}(a))_m = \lambda i : f(a)_I.\, \mathsf{leaf}(f(a)_m(i))$.
- $T_Cf(\mathsf{node}_\sigma(c))_m = \lambda h : ((i : ar(\sigma)) \to T_Cf(c(i))_I).\, \mathsf{node}_\sigma(\lambda i.\, T_Cf(c(i))_m(h(i)))$.

With this definition, T_C forms an endofunctor in $\mathsf{SNF}_{\mathbb{U}}$.

Let us consider whether the above recipe gives us what we want. We independently specify a relation which tells us whether a term is a result of making nondeterministic choices at the leaves. This is akin to how one would lift the free monad to the category of relations. Then, we can check soundness and completeness; meaning the result of our endofunctor contains precisely the choices specified by the relation.

Given $f : A \to \mathsf{SL}_{\mathbb{U}}B$, the *$f$-choice relation* is a relation between T_CA and T_CB inductively defined as follows:

- It relates $\mathsf{leaf}(a)$ to $\mathsf{leaf}(b)$ if b is a result of $f(a)$.
- It relates $\mathsf{node}_\sigma(c)$ to $\mathsf{node}_\sigma(d)$ if for every $i \in ar(\sigma)$, it relates $c(i)$ to $d(i)$.

We can show that our endofunctor T_C is complete over the choice relation.

▶ **Theorem 18.** *Given $f : A \to \mathsf{SL}_{\mathbb{U}}B$, $t \in T_CA$ and $r \in T_CB$, then:*
$\exists i \in T_Cf(t)_I.\, T_Cf(t)_m = r$ if and only if t is related to r by the f-choice relation.

5.2 Monad and Comonad Structure

The free functor forms a monad in the category of relations. The appropriate natural transformations are constructed by lifting the unit and multiplication map of the free monad from Set to $\mathsf{SNF}_{\mathbb{U}}$ by composing it with the unit transformation $\langle - \rangle$ for families. Unit and multiplication are given by $T_C^\eta = |\eta_A^C| : A \multimap T_CA$ and $T_C^\mu = |\mu_A^C| : T_C(T_CA) \multimap T_CA$.

▶ **Proposition 19.** *For each $\mathbb{U}$-container C, (T_C, T_C^η, T_C^μ) is a monad in $\mathsf{SNF}_{\mathbb{U}}$.*

It is possible to reverse the structure of this monad, and construct a comonad.

- Let $T_C^\varepsilon : T_CA \multimap A$ be the morphism sending $\mathsf{leaf}(a)$ to $\langle a \rangle$, and $\mathsf{node}_\sigma(c)$ to $\oslash_A$, then $T_C^\eta \dagger T_C^\varepsilon$.
- Let $T_C^\delta : T_CA \multimap T_C(T_CA)$ be the morphism sending $\mathsf{leaf}(a)$ to $\langle \mathsf{leaf}(\mathsf{leaf}(a)) \rangle$, and $\mathsf{node}_\sigma(c)$ to $\langle \mathsf{leaf}(\mathsf{node}_\sigma(c)) \rangle \vee N$, where $N = (N_I, N_m)$ is given by $N_I = \Pi_{i:ar(\sigma)}\, T_C^\delta(c(i))_I$ and $N_m = \lambda(i,j).\, \mathsf{node}_\sigma(T_C^\delta(c(i))_m(j))$. Then $T_C^\mu \dagger T_C^\delta$.

Hence $(T_C, T^\varepsilon, T^\delta)$ is a comonad. Suppose for instance $T_C A$ describes a set of computational processes, with η creating a process that immediately terminates and gives a result, and μ merges a two-staged process into a single process. Then ε extracts from a process any result that is immediately produced, and δ nondeterministically splits a process into two stages. The monad and comonad structure interacts in interesting ways.

- $T^\eta_C; T^\varepsilon_C \sim id$; we can extract a result from a process that immediately terminates.
- $T^\delta_C; T^\mu_C \sim id$; when we split a process into two stages, and merge the stages, we get the original process back.
- $T^\mu_C; T^\varepsilon_C \sim T^\varepsilon_C; T^\varepsilon_C$; extracting results from a process is like extracting from its stages.
- $T^\eta_C; T^\delta_C \sim T^\eta_C; T^\eta_C$; splitting a terminating process gives two terminating stages.

6 Example Processes

We look at some examples of nondeterministic processes we can represent with slice nondeterministic functions.

6.1 Interleaving Concurrency

Let us first look at *interleaving concurrency*, which can be modelled by the interleaving operation on actions. Consider lists X^* over X inductively defined as $[] \in A^*$ and for $x \in X$ and $l \in X^*$, $x :: l \in X^*$.

▶ **Definition 20.** *The* interleaving operations *on lists* $\|, \|^l, \|^r \colon X^* \times X^* \multimap X^*$ *are inductively defined as:*

- $(a \parallel b) = (a \parallel^l b) \vee (a \parallel^r b)$.
- $([] \parallel^l a) = \langle a \rangle = (a \parallel^r [])$.
- $(x :: a \parallel^l b) = \mathsf{SL}_{\mathbb{U}}(\lambda c.x :: c)(a \parallel b) = (a \parallel^r x :: a)$.

The definition of parallel operation is split into three clauses in order to satisfy Agda's termination checker. In the axioms of *process algebra* (see e.g. [8]), the parallel operation was already split into two clauses corresponding to our $\parallel$ and $\parallel^l$, in order for the induction principle to apply. Here, we separately define $\parallel^r$, since without it, a non-trivial order on arguments need to be established in order to prove termination of the function.

The parallel operation satisfies some interesting properties, which can be summarised in the following proposition.

▶ **Proposition 21.** $(\|\colon X^* \times X^* \multimap X^*, e : \top \multimap X^*)$ *where* $e(*) = \langle [] \rangle$ *forms a commutative monoid in* $\mathsf{SNF}_{\mathbb{U}}$

▶ Remark. The approach of using slices for modelling nondeterminism was used at first in order to prove properties like associativity of the parallel operation. Earlier efforts to model interleaving concurrency used the representation of the finite powerset monad as a free idempotent commutative monoid. Proving equations of the interleaving operation required precise rewriting of terms using associativity and commutativity of the monoid. Nondeterminism via slices on the other hand uses a case analysis on the elements of the indexing set, which in this case represents the exact nondeterministic choices made by the parallel operation. Though the proof of associativity is still not simple, the technique of using slice nondeterminism greatly reduces the level of bureaucracy necessary.

It can be verified that this model satisfies the equations of process algebra as for instance specified in [8]. The development in Agda also contains a formalisation of previous work from the author [24] which considers a categorical model for interleaving algebraic effects using $\mathsf{SNF}_{\mathsf{Set}}$.

6.2 Iterated Processes

As our second example, we study a simple state automata which is given by a state space, and a nondeterministic transition function into the set of states and outputs. We can iterate over chains of transitions using a natural number, and then take the supremum over this natural number to get a single collection of all possible outputs.

We define the iteration function $\mathsf{Iter}_{A,B} : (A \multimap B \uplus A) \to \mathbb{N} \to (A \multimap B)$, which takes a morphism $H : A \multimap B \uplus A$ and a natural number $n \in \mathbb{N}$, and iterates H n-times, gathering all results from B it produces. Defined inductively on $n \in \mathbb{N}$, it does the following:

- $\mathsf{Iter}_{A,B}(H, 0) = \lambda a.\oslash_B$.
- $\mathsf{Iter}_{A,B}(H, n+1) = H; (id_B \uplus \mathsf{Iter}_{A,B}(H, n))$.

$\mathsf{Iter}_{A,B}(H, n+1)(a)$ looks at $H(a)$, which is a collection of elements of $B \uplus A$. It keeps all results from B and joins them with elements of $\mathsf{Iter}_{A,B}(H, n)(a')$ for any result a' from A. The function $\mathsf{Iter}_{A,B}(H, -)$ forms an ω-chain.

We define $\mathsf{Iter}\omega_{A,B} : (A \multimap B \uplus A) \to (A \multimap B)$ by taking the countable join of $\mathsf{Iter}_{A,B}$. We have the following results:

- Both $\mathsf{Iter}_{A,B}$ and $\mathsf{Iter}\omega_{A,B}$ preserve the similarity relation $\sim$ on morphisms, and hence are well defined as functions on morphisms in $\mathsf{SNF}_{\mathbb{U}}$.
- For $f : A \multimap B$, $H : B \multimap C \uplus A$, and $n \in \mathbb{N}$, $\mathsf{Iter}_{A,B}(f; H, n) \sim f; (\mathsf{Iter}_{A,B}(H; (id_C \uplus f), n))$, hence $\mathsf{Iter}\omega_{A,B}(f; H) \sim f; (\mathsf{Iter}\omega_{A,B}(H; (id_C \uplus f)))$.
- For $f : B \multimap C$, $H : A \multimap B \uplus A$, and $n \in \mathbb{N}$, $\mathsf{Iter}_{A,B}(H; (f \uplus id_A), n) \sim \mathsf{Iter}_{A,B}(H, n); f$, and hence $\mathsf{Iter}\omega_{A,B}(H; (f \uplus id_A), n) \sim \mathsf{Iter}\omega_{A,B}(H, n); f$.
- For $H : A \multimap B \uplus A$, $\mathsf{Iter}\omega_{A,B}(H) \sim H; (id_B \uplus \mathsf{Iter}\omega_{A,B}(H, n))$.
- For $H : A \multimap B \uplus A$ and $K : B \multimap C \uplus B$, $\mathsf{Iter}\omega_{A,B}(H); \mathsf{Iter}\omega_{B,C}(K) = \bigvee(\lambda n.\, \mathsf{Iter}_{A,B}(H, n); \mathsf{Iter}_{B,C}(K, n))$.
- $\mathsf{Iter}\omega_{A,A}(\mathsf{inj}_1) \sim \langle - \rangle_A$.

The $\mathsf{Iter}\omega$ operation can be used as a basis for a traced monoidal [5] operation for $\mathsf{SNF}_{\mathbb{U}}$, and can moreover be used to model recursive processes using ω-chains. As such, it could form a basis for *domain theoretic* denotations using streams as in previous work from the author [23]. There, such streams were used to define denotational equivalence of recursive effectful programs.

6.3 Labelled transition systems

We consider a common nondeterministic process called the *labelled transition system*. Here each possible transition is given a corresponding label designating some input, and a decidable predicate on states specifying final states. We leave the choice of initial state as a parameter.

▶ **Definition 22.** *A* labelled transition system *(lts) over a set of labels $A : \mathbb{U}$ is specified by a triple (S, t, e) consisting of a set of states S, a transition map $t : S \times A \multimap S$ and a function checking for ending states $e : S \to \mathbb{B}$.*

We denote the set of labelled transition systems over A as $\mathtt{LTS}(A)$. We are often interested in checking which series of labels would lead to a final state. A list of labels l is *accepted* by an lts (S, t, e) on some initial state $s \in S$ if there is a path of labelled transitions corresponding to the labels of l, from s to some final state from $e^{-1}(\mathsf{true})$. There are two ways to specify which lists of labels are accepted. We can either directly enumerate them using accepted paths, or we can inductively check whether a list is accepted.

We can generate a collection of paths which will lead to termination inductively as a dependent function $\mathsf{LTSCol}(A) : ((S, t, r) : \mathsf{LTS}(A)) \to \mathbb{N} \to (S \multimap \mathsf{SL}_{\mathbb{U}}(A^*))$, which for a natural number $n \in \mathbb{N}$ collects all accepted lists of length n.

- $\mathsf{LTSCol}(A)(S, t, e)(0)(s) = \begin{cases} \langle [] \rangle & \text{if } (e(s) = \mathsf{true}) \\ \varnothing_{A^*} & \text{if } (e(s) = \mathsf{false}) \end{cases}$
- $\mathsf{LTSCol}(A)(S, t, e)(n + 1)(s) = (I, h)$ where I is a set of triples consisting of $a \in A$, $i \in t(a, s)_I$ and $j \in \mathsf{LTSCol}(A)(S, t, e)(n)(t(s, a)_m(i))_I$, and $h(a, i, j) = a :: \mathsf{LTSCol}(A)(S, t, e)(n)(t(s, a)_m(i))_m(j)$.

We then define $\mathsf{LTSCol}\omega$ as the supremum over $\mathbb{N}$ of LTSCol.

Alternatively, we can define a predicate which checks whether a certain list is accepted. $\mathsf{LTSaccept}(A) : ((S, t, e) : \mathsf{LTS}(A)) \to S \to A^* \to \mathsf{Set}$ where:

- $[]$ is accepted on initial state s if $e(s) = \mathsf{true}$.
- $a :: l$ is accepted on initial state s if there is a state z in the collection $t(s, a)$ such that l is accepted on initial state z.

▶ **Proposition 23.** *The collection of lists generated by* $\mathsf{LTSCol}\omega$ *consists exactly of the lists accepted according to* $\mathsf{LTSaccept}$.

7 Conclusions

We have looked at a formalism for constructively representing denotations of nondeterministic processes. These models are inherently intensional as they send inputs to outputs dependent on concrete nondeterministic choices, as opposed to giving an external predicate for checking whether an output is possible. The model is moreover directional, as only a subset of well-behaved morphisms have daggers. As such, the model stays closer to natural occurrences of computational nondeterministic processes. Nondeterminism is often guided by a source of unpredictability, like sampling spaces in probability theory. In terms of this model, the nondeterministic sampling space takes the form of an indexing set of a family. Potential extensions to probabilistic models could be considered in the future.

We focused mainly on a specific universe of indexing sets $\mathbb{U}$. Earlier versions of the work used the collection of sets as a universe. Other possible universes may be explored in the future too. Varying the universe allows us to model different categories, for instance:

- Taking the universe $\{\top\}$, we effectively retrieve the category of sets Set.
- Taking the universe $\{\bot, \top\}$, we model the category of partial functions Par.
- Taking finite sets as a universe, we get finite nondeterministic functions.
- Excluding functions and Π constructions from $\mathbb{U}$, we get countably nondeterministic functions, e.g. programs which can sample natural numbers nondeterministically.

If there is an inclusion of universes, there is a functor between their respective categories. There are more variations to consider, since we can restrict our sets of morphisms to those satisfying a chosen subset of properties; total, deterministic, surjective, injective, daggerable.

Note that in the above formulation of partial functions, it is decidable whether a function gives a result or not. If you want to capture undecidability, you should instead use the subcategory of deterministic slice functions, where inhabitance of an indexing set is undecidable. This does however require you to show that all possible results are equal.

The slice nondeterminism framework is in some aspects much easier to work with then algebraic representations of nondeterminism. For instance, equations like idempotency, associativity and commutativity of nondeterministic choice (implemented by the join operation) are easily proven using case distinctions on the indexing set. Moreover, proofs that normally need such equations can be easily shown without needing to explicitly call these equations, avoiding the need to specify a concrete sequence of rewrites. In particular, the formalisation of properties for interleaving concurrency was hampered by the need of intensive equational reasoning when working with a monoidal representation of the finite powerset monad. But switching over to slices, many of these former difficulties were easily avoided.

7.1 Agda Categories Library

The formalisation of this paper is grounded in the framework of setoid enriched categories ($\mathcal{E}$-categories), like the development of the Agda Categories library. The core of the formalisation only uses Agda's standard library, for ease of adaptability. In separate files, some of the results are linked up with the Agda Categories library, and concrete properties are shown. These properties are as follows.

We have shown the following things concerning $\mathsf{SNF}_{\mathbb{U}}$.

- $\mathsf{SNF}_{\mathbb{U}}$ is a category.
- $\mathsf{SNF}_{\mathbb{U}}$ is symmetric monoidal with respect to the product and disjoint union bifunctor.
- $\mathsf{SNF}_{\mathbb{U}}$ is Cartesian and Cocartesian

Secondly, we have shown that the endofunctor $\mathsf{SL}_{\mathbb{U}}$ can be used to form a monad in the category of setoids. This is the perfect candidate for a powerset monad over setoids, which could be called the *powersetoid* monad, since it is a model of the powertheory. Formalisation in terms of setoids specifically may be a future avenue for research.

Lastly, consider the slice category with indexing sets ranging over Set instead of $\mathbb{U}$, then:

▶ **Theorem 24.** *The category $\mathit{SNF}_{\mathsf{Set}}$ is equivalent to the category of relations.*

7.2 Denotational Semantics

Some examples were considered in Section 6, regarding finite automata and process algebra. An example for the future is to formalise functional nondeterministic languages as studied by Lassen [14]. Given the fact that families are closed under taking limits (see iterated processes in Section 6), it should be possible to create a denotational model for nondeterministic functional languages, like the one employed in previous work using streams [23]. A simple example has been worked out, providing a denotational semantics for an untyped call-by-value lambda calculus with added nondeterminism.

More generally, using a specification by nondeterministic runner [22, 24], we can model a variety of stateful and nondeterministic effects. When described in this framework, we use an indexing set to enumerate all possible handlers [20] for the effect, which can then be used to extract a set of possible results of the computation. Part of the theory surrounding this is formalised in the current development, and studying such algebraic models using the formalisation presented in this paper is subject to future research.

References

1 Michael Gordon Abbott, Thorsten Altenkirch, and Neil Ghani. Containers: Constructing strictly positive types. *Theoretical Computer Science*, 342(1):3–27, 2005. doi:10.1016/j.tcs.2005.06.002.

2 Samson Abramsky. Domain theory in logical form. *Ann. Pure Appl. Log.*, 51(1–2):1–77, 1991. doi:10.1016/0168-0072(91)90065-T.

3 Peter Aczel. Galois: a theory development project, 1993. In: A report on work in progress for the Turin meeting on the Representation of Logical Frameworks.

4 Thorsten Altenkirch, Neil Ghani, Peter Hancock, Conor Mcbride, and Peter Morris. Indexed containers. *Journal of Functional Programming*, 25:e5, 2015. doi:10.1017/S095679681500009X.

5 Joyal Andre, Ross Street, and Dominic Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119:447–468, April 1996. doi:10.1017/S0305004100074338.

6 Jon Beck. Distributive laws. In B. Eckmann, editor, *Seminar on Triples and Categorical Homology Theory*, pages 119–140, Berlin, Heidelberg, 1969. Springer Berlin Heidelberg. doi:10.1007/BFb0083084.

7 Jean Bénabou. Introduction to bicategories. In *Reports of the Midwest Category Seminar*, pages 1–77, Berlin, Heidelberg, 1967. Springer Berlin Heidelberg.

8 J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985. doi:10.1016/0304-3975(85)90088-X.

9 Gerhard Gierz, Karl Heinrich Hofmann, Klaus Keimel, Jimmie Lawson, Michael Mislove, and Dana S. Scott. *Continuous Lattices and Domains*. Cambridge University Press, Cambridge, 2003.

10 C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, August 1978. doi:10.1145/359576.359585.

11 Markus Holzer and Martin Kutrib. Reversible nondeterministic finite automata. In *International Workshop on Reversible Computation*, pages 35–51, May 2017. doi:10.1007/978-3-319-59936-6_3.

12 G. Kelly. The basic concepts of enriched category theory. *Reprints in Theory and Applications of Categories [electronic only]*, 2005, January 2005.

13 Yoshiki Kinoshita. A bicategorical analysis of E-categories. *Mathematica Japonica*, 47:157–170, 1998.

14 Søren B. Lassen. *Relational Reasoning about Functions and Nondeterminism*. PhD thesis, University of Aarhus, 1998.

15 Paul Blain Levy. Similarity quotients as final coalgebras. In Martin Hofmann, editor, *FoSSaCS 2011*, volume 6604 of *LNCS*, pages 27–41. Springer, 2011. doi:10.1007/978-3-642-19805-2_3.

16 Saunders MacLane. *Categories for the Working Mathematician*. Springer-Verlag, New York, 1971. Graduate Texts in Mathematics, Vol. 5.

17 Rasmus Ejlers Møgelberg and Andrea Vezzosi. Two guarded recursive powerdomains for applicative simulation. In Ana Sokolova, editor, *Proceedings of 37th Conference on Mathematical Foundations of Programming Semantics, MFPS 2021*, volume 351 of *Electron. Proc. in Theor. Comput. Sci.*, pages 200–217, 2021. doi:10.4204/EPTCS.351.13.

18 Gordon D. Plotkin. A powerdomain construction. *Siam J. Comput.*, 5(3):452–487, 1976. doi:10.1137/0205035.

19 Gordon D. Plotkin and John Power. Adequacy for algebraic effects. In Furio Honsell and Marino Miculan, editors, *FoSSaCS 2001*, volume 2030 of *LNCS*, pages 1–24, 2001. doi:10.1007/3-540-45315-6_1.

20 Gordon D. Plotkin and Matija Pretnar. Handling algebraic effects. *Log. Methods Comput. Sci.*, 9(4, article 23):1–36, 2013. doi:10.2168/lmcs-9(4:23)2013.

21 Albert Marchienus Thijs. *Simulation and Fixpoint Semantics*. PhD thesis, University of Groningen, 1996. URL: `https://research.rug.nl/en/publications/simulation-and-fixpoint-semantics`.

22 Tarmo Uustalu. Stateful runners of effectful computations. *Electron. Notes Theor. Comput. Sci.*, 319:403–421, 2015. `doi:10.1016/j.entcs.2015.12.024`.

23 Niccolò Veltri and Niels F. W. Voorneveld. Streams of approximations, equivalence of recursive effectful programs. In Ekaterina Komendantskaya, editor, *Mathematics of Program Construction - 14th International Conference, MPC 2022, Tbilisi, Georgia, September 26-28, 2022, Proceedings*, volume 13544 of *Lecture Notes in Computer Science*, pages 198–221. Springer, 2022. `doi:10.1007/978-3-031-16912-0_8`.

24 Niels F. W. Voorneveld. Runners for interleaving algebraic effects. In Helmut Seidl, Zhiming Liu, and Corina S. Pasareanu, editors, *Theoretical Aspects of Computing – ICTAC 2022*, pages 407–424, Cham, 2022. Springer International Publishing.