ELSEVIER Theoretical Computer Science 177 (1997) 3–25

Theoretical Computer Science

A representable approach to finite nondeterminism

S.O. Anderson, A.J. Power *,1

Department of Computer Science, University of Edinburgh, King's Buildings, Edinburgh EH9 3JZ, UK

Abstract

We reformulate denotational semantics for nondeterminism, taking a nondeterministic operation $\vee$ on programs, and sequential composition, as primitive. This gives rise to binary trees. We analyse semantics for both type and program constructors such as products and exponential types, conditionals and recursion, in this setting. In doing so, we define new category-theoretic structures, in particular premonoidal categories. We also account for equivalences of programs such as those induced by associativity, symmetry and idempotence of $\vee$, and we study finite approximation by enrichment over the category of ω-cpos with least element. We also show how to recover the classical powerdomains, especially the convex powerdomain, as three instances of a general, computationally natural, construction.

0. Introduction

Over the past few years, monads, equivalently Kleisli triples, have been advocated as an appropriate mathematical structure with which to model "notions of computation" [6]. The seeds of that idea, and its leading example, lie in Gordon Plotkin's attempt to model nondeterminism with a powerdomain in [7]. However, there is no *need* to model nondeterminism that way. In fact, there are computationally natural reasons, as we explain later, to use different category-theoretic primitives, and to derive the powerdomain from them via a simple category-theoretic construction.

Here we reformulate semantics for nondeterminism in a principled way, the structures we develop forming a basis and leading example for modelling notions of computation more generally, with that goal motivating some of our specific choices. In fact, our semantics does generalise and we can recover all computational monads in the same way as we recover the powerdomain here.

We seek to understand nondeterminism in denotational semantics. Classically, three powerdomains, the lower, upper, and convex powerdomains, have been used. The structure of a powerdomain consists of, in the first instance,

* Corresponding author. E-mail: ajp@dcs.ed.ac.uk.

[1] Supported by EPSRC grant GR/J84205: Frameworks for programming language semantics and logic.

- for a countable set S of states, a set $\mathscr{P}(S)$, which is then endowed with the structure of an ω-cpo with least element;
- an "extension", i.e., a function that takes any function $f : X \to \mathscr{P}(Y)$ with X and Y countable to a unique map $f^\dagger : \mathscr{P}(X) \to \mathscr{P}(Y)$ satisfying certain properties;
- a "singleton" function $\{\ \} : X \to \mathscr{P}(X)$ (a property of extension being that it commutes with singleton); and
- a function $\cup : \mathscr{P}(X) \times \mathscr{P}(X) \to \mathscr{P}(X)$ called binary union, which is to satisfy some properties.

The partial order on $\mathscr{P}(X)$ is used to define extension, which is in turn used to define composition of the semantics of nondeterministic programs, where the semantics of a nondeterministic program is given by a map from X to $\mathscr{P}(Y)$. The partial order on $\mathscr{P}(X)$ induces a partial order on $X \to \mathscr{P}(X)$, which is used to define the semantics of iteration or other recursion constructs.

It is also observed that if the first three pieces of data are given for all sets, or more generally for all ω-cpos with least element, then that corresponds precisely to giving a Kleisli triple, or equivalently a monad, on the category of all sets or that of all ω-cpos with least element respectively. The basic theory of powerdomains is developed in [7, 13, 15, 8]. More recently, there has been substantial development of this theory. For instance, Di Giantonio et al. [4] have made delicate use of ω_1-enrichment to model countable nondeterminism, and Alex Simpson [12], based upon the work herein, has modelled nondeterminism with a distinction between intensional and extensional properties.

We wish to reformulate this semantics for the following reason: in the development of semantics, sequential composition of programs is typically treated as a primitive operation on programs, and its denotational semantics has reflected that; so we should like to continue with that rather than relegate the semantics of sequential composition to the complex construction involving extension. In addition, the three classical powerdomains have typically been treated as separate constructions. Our approach allows us to see them as three instances, indeed the only three instances, of a general, construction applied to one generic formalism for nondeterminism.

We take as primitive constructions *sequential composition* of programs and *nondeterministic* $\vee$. In order to ensure that this is not studied in isolation, we shall do it in the context of **if**, **while**, and later **print**, to be sure we capture the spirit of ω-cpos with least element, specifically the idea of finite approximation of input, of output, and of programs. We also account for simple type constructors: products, exponentials, and positively recursively defined types.

There are at least two ways in which nondeterminism arises: one, the action of an external agent, e.g., the human dealing with email. Here a deterministic program is modelled by a nondeterministic operator because it is a useful simplification. Also this illustrates how finite approximation need not be recursive: it could be given by human interaction. This will be our leading example of nondeterminism through the course of the paper. Second, from concurrency. We do not address that here as it involves extra considerations, in particular that of nonobservable behaviour, such as in weak bisimulation or testing equivalence; the classical powerdomains do not handle such equivalences anyway.

Nondeterminism is a complex and subtle phenomenon. A full conceptual analysis is beyond the scope of this short paper. The leading example mentioned above, although simple, provides sufficient detail to motivate and illustrate our general approach.

In Section 1 we introduce results we need on $\mathscr{V}$-categories with tensors, and explain how we may recover powerdomains in general from our formalism. In Section 2, we give semantics for the discrete part of our language, i.e., everything except **print**. Sections 3 and 4 account for simple type constructors: the former for product types, in order to account for programs of several variables; the latter for recursively defined and higher-order types. In Section 5 we show how to account for factoring out equivalences such as associativity, commutativity and idempotence in this setting. Section 6 shows how to recover the three classical powerdomains. Finally, in Section 7, we account for finite approximation via enrichment over cpos, enriching all the previous analysis with one substantial modification.

An earlier version of this work was presented at MFPS'92 in Oxford.

1. Category-theoretic preliminaries

A *monoidal category* $\mathscr{V}$ consists of a category $\mathscr{V}_0$ together with a functor $\otimes : \mathscr{V}_0 \times \mathscr{V}_0 \to \mathscr{V}_0$ that is associative up to coherent isomorphism, with a two-sided unit up to coherent isomorphism. A monoidal category is *symmetric* if the operation $\otimes$ is symmetric up to coherent isomorphism, and is *closed* if for each object X of $\mathscr{V}_0$, the functor $- \otimes X : \mathscr{V}_0 \to \mathscr{V}_0$ has a right adjoint.

The leading example of a symmetric monoidal closed category is the cartesian closed category Set of small sets and functions: the monoidal structure is given by finite product, and the closed structure is given by exponentials. For another example, the category $\mathscr{O}$ of ω-cpos with least element and maps preserving the least element also has a symmetric monoidal closed structure, which is of fundamental importance in domain theory. The monoidal structure is given as follows: given A and B, strip their least elements, take the product, and add a least element again; equivalently, take the product of A and B and identify any two elements of which one coordinate is the least element. For details of the definition and for a range of examples used extensively in mathematics, see Max Kelly's book [5].

Given a symmetric monoidal closed category $\mathscr{V}$, there are well-understood notions of $\mathscr{V}$-category, $\mathscr{V}$-functor and $\mathscr{V}$-natural transformation, giving a 2-category $\mathscr{V}$-Cat. The idea of a $\mathscr{V}$-category is that in the definition of category, one has a set of objects and, for each pair of objects A and B, a homset of arrows from A to B; in a $\mathscr{V}$-category $\mathscr{C}$, one still has a set of objects, but for each pair of objects A and B, one has a homobject $\mathscr{C}(A,B)$ of $\mathscr{V}$. The monoidal structure of $\mathscr{V}$ allows one to express a notion of composition in a $\mathscr{V}$-category, generalising that in the definition of ordinary category. From the definition of $\mathscr{V}$-category, it is routine to define the notion of

$\mathscr{V}$-functor; that of $\mathscr{V}$-natural transformation requires just a little more thought as one needs to define the notion of arrow in a $\mathscr{V}$-category: an arrow from A to B is defined to be an arrow in $\mathscr{V}$ from the unit I to the homobject $\mathscr{C}(A,B)$. The details are all spelt out at the start of Kelly's book [5].

For example, for the cartesian closed category **Set**, **Set-Cat** is precisely **Cat**, because to give a homobject $\mathscr{C}(A,B)$ of **Set** is precisely to give a homset. In the case of $\mathscr{O}$, an $\mathscr{O}$-category is a category together with, for each pair of objects A and B, an ω-cpo structure with least element on the set of arrows from A to B, such that composition preserves the structure. Observe that for any symmetric monoidal closed category $\mathscr{V}$, we may consider $\mathscr{V}$ as a $\mathscr{V}$-category: homobjects are given by the closed structure of $\mathscr{V}$. The standard reference for $\mathscr{V}$-categories is Kelly's book [5].

Given the definitions of $\mathscr{V}$-category, $\mathscr{V}$-functor and $\mathscr{V}$-natural transformation, one immediately has the notion of $\mathscr{V}$-adjunction: a $\mathscr{V}$-functor $H : \mathscr{D} \to \mathscr{C}$ has a right adjoint K if there is a family of isomorphisms of homobjects between $\mathscr{D}(D,KC)$ and $\mathscr{C}(HD,C)$, that is $\mathscr{V}$-natural in C and D.

One can similarly, routinely, generalise many of the basic notions of ordinary category theory to enriched categories. One construction that is important to us becomes more vivid in this setting: it is the notion of a tensor, as follows.

Definition 1. A $\mathscr{V}$-category $\mathscr{C}$ has *tensors* if for every object A of $\mathscr{C}$ and every object X of $\mathscr{V}$, the $\mathscr{V}$-functor $[X,\mathscr{C}(A,-)] : A \to \mathscr{V}$ is representable, i.e., if there is an object $X \otimes A$ of $\mathscr{C}$ together with a collection of isomorphisms $[X,\mathscr{C}(A,B)] \cong \mathscr{C}(X \otimes A,B)$, natural in B.

If a $\mathscr{V}$-category $\mathscr{C}$ has tensors, then it follows that the collection of isomorphisms in the definition is not only natural in B, but also natural in X and A. To have tensors is a mild cocompleteness condition on a $\mathscr{V}$-category.

If $\mathscr{V}$ is **Set**, then the tensor of a set X with an object A of a category $\mathscr{C}$ is precisely the coproduct of X copies of A. For $\mathscr{V} = \mathscr{O}$, the situation is more delicate. However, in the particular $\mathscr{O}$-category $\mathscr{O}$, which is our leading example, tensors are simple: the tensor of an object X with another object Y is as above, i.e., the cartesian product of X and Y factored by $(x,\bot) \sim (\bot,y)$ for all x and y. The universal property can be reexpressed as the assertion that $X \otimes Y$ classifies bistrict maps out of X and Y.

A $\mathscr{V}$-functor is said to preserve tensors when it preserves this construction. It is routine to verify that if $H : \mathscr{B} \to \mathscr{C}$ and $K : \mathscr{C} \to \mathscr{D}$ preserve tensors, then so does the composite KH.

Proposition 2. *Let $\mathscr{C}$ be a tensored $\mathscr{V}$-category. Then, a $\mathscr{V}$-functor $H : \mathscr{V} \to \mathscr{C}$ has a right adjoint if and only if H preserves tensors.*

Proof. One way follows since tensors, being colimits, are preserved by left adjoints. Alternatively, one can verify the result directly from the definition of tensor: we need to show that $H(X \otimes A)$ satisfies the defining condition for $X \otimes HA$, i.e., that $\mathscr{C}(H(X \otimes A),B)$ is isomorphic to $[X,\mathscr{C}(HA,B)]$; but that may be proved by two applications of the adjunction isomorphism together with the definition of tensor. For the other direction, the right adjoint is given by $\mathscr{C}(HI,-)$, where I is the unit of the monoidal category $\mathscr{V}$. That this is a right adjoint is an immediate consequence of the definition of tensor and the fact that H preserves tensors. □

Corollary 3. *For any $\mathscr{V}$-category $\mathscr{C}$, an identity on objects $\mathscr{V}$-functor $H : \mathscr{V} \to \mathscr{C}$ has a right adjoint if and only if H preserves tensors.*

Proof. In order to apply the previous result, we need only check that $\mathscr{C}$ has tensors. Given X an object of $\mathscr{V}$ and A an object of $\mathscr{C}$, since the objects of $\mathscr{C}$ are equal to those of $\mathscr{V}$, it follows that A is an object of $\mathscr{V}$. The $\mathscr{V}$-category $\mathscr{V}$ has tensors given by the tensor in $\mathscr{V}$ regarded as a monoidal category, so we have the tensor $X \otimes A$ in $\mathscr{V}$. If H is assumed to preserve tensors, then since H is the identity on objects, it follows that $X \otimes A$ acts as the desired tensor in $\mathscr{C}$, and if H is assumed to have a right adjoint, then again $X \otimes A$ is the tensor in $\mathscr{C}$, with proof given as in the first part of the proof of the proposition. □

This is all we need about enriched categories in this paper. We next turn to the basic category theory we need to show the relationship between our account of nondeterminism and the traditional one based on powerdomains. We recalled the definition of powerdomain in the introduction. More recently, rather than spelling out the structure of a powerdomain as in the introduction, the tendency has been to identify the structure of a powerdomain with that of a monad P on a category $\mathscr{C}$, together with a natural transformation with components $\cup : PX \times PX \to PX$, subject to laws for associativity, commutativity, and idempotence. A monad may be defined as follows.

Definition 4. A *monad* (P,m,j) on a category $\mathscr{C}$ consists of a functor $P : \mathscr{C} \to \mathscr{C}$ together with natural transformations with components $m_X : PPX \to PX$ and $j_X : X \to PX$ subject to one equation expressing associativity of m with respect to P and two equations to the effect that j provides a two-sided unit for m.

One would typically write P for the monad, taking the natural transformations in the definition as implicit. It is straightforward to verify that a powerdomain, as defined in the introduction, gives rise to monad together with a natural transformation as above. For more detail on monads in this regard, see [6].

The usual way to obtain a monad is from an adjunction: given any adjunction, $H \dashv K$, with unit η and counit ε, it is routine to verify that the triple $(KH, K\varepsilon H, \eta)$ is a monad. One can give a converse to that in two canonical ways. The one of primary interest to us is given by Kleisli as follows.

Proposition 5. *To give a monad (P,m,j) on a category $\mathscr{C}$ is to give an identity on objects functor $J : \mathscr{C} \to \mathscr{K}$ that has a right adjoint.*

Proof. Given (P,m,j), define $\mathscr{K}$ to be the Kleisli category for the monad: $\mathscr{K}$ has the same objects as $\mathscr{C}$, with $\mathscr{K}(A,B)$ defined to be $\mathscr{C}(A,PB)$, and with composition defined by use of the natural transformation m. There is a functor J from $\mathscr{C}$ to $\mathscr{K}$ defined by the identity on objects, and on arrows, by composing an arrow from A to B with the B-component of j. The functor J has right adjoint given on objects by P. Conversely, given J, the monad arises from the adjunction. It is routine to verify that these constructions are mutually inverse. □

For a general study of the relationship between identity on objects functors and monads, see [14, Section 5]. We do not need such generality in this paper, so we restrict our attention to the simple structures we do need.

Proposition 6. *Let $\mathscr{C}$ be any category with finite products, and let P be any monad on it. Then, to give $\cup : PX \times PX \to PX$ is to give for each object Y of $\mathscr{C}$, a function $\vee : \mathscr{K}(Y,X) \times \mathscr{K}(Y,X) \to \mathscr{K}(Y,X)$ natural in Y regarded as an object of $\mathscr{C}$, where $\mathscr{K}$ is the Kleisli category for P. Moreover, $\cup$ is associative if and only if $\vee$ is, and similarly for commutativity and idempotence.*

Proof. This is a consequence of the Yoneda lemma. First note that, by the definitions of finite products and adjunctions, $\mathscr{K}(Y,X) \times \mathscr{K}(Y,X) = \mathscr{C}(Y, PX \times PX)$ and $\mathscr{K}(Y,X) = \mathscr{C}(Y,PX)$. So, given $\cup$, one can define $\vee$ by composition. Also, having noted the above, the converse construction, i.e., that of $\cup$ from $\vee$, is a direct instance of the Yoneda lemma. The last line of the theorem also follows from the Yoneda lemma: associativity of $\cup$ is preserved by composition with any map from Y to PX, thus yielding associativity of $\vee$, and the Yoneda lemma provides the converse; similarly for commutativity and idempotence. □

We shall take as our primitive data to define a denotational semantics for nondeterminism:

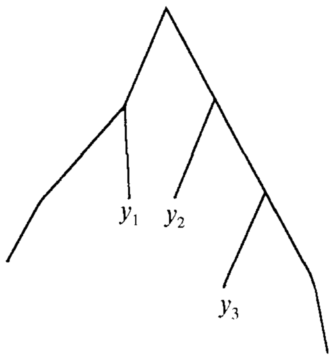
- a category $\mathscr{C}$ to provide denotational semantics for the deterministic part of the language,
- an identity on objects functor (or $\mathscr{O}$-functor later) $J : \mathscr{C} \to Nondet(\mathscr{C})$ with the latter $\mathscr{O}$-category providing our full semantics. The inclusion J represents taking a deterministic program and regarding it trivially as a nondeterministic program. It is evident that one would want such a J to preserve composition and identities; it will be further evident, when we discuss enrichment, that we will want J to enrich too. In order to model nondeterminism, there is no inherent reason to insist that J be the identity on objects. However, initially there is no reason to alter the objects of $\mathscr{C}$ in extending the semantics. Thus for simplicity we choose an identity on objects functor. Of course, $Nondet(\mathscr{C})$ will be equipped with a binary operation on each hom to model $\vee$.

We shall use Propositions 5 and 6 to recover the powerdomains.

2. The basic case

As a first attempt to model nondeterminism, we assume we have a language as outlined in the introduction except for **print**, with a denotational semantics of its deterministic totally defined part in the category **Set**. We now extend **Set** to a new category *Nondet*(**Set**) in order to model the language including nondeterministic or. In principle, we freely adjoin a binary operation $\vee$ to each hom. Formally, it is not quite so simple. The main reason is that in order to account for recursion in the guise of **while**, we need to allow application of $\vee$ infinitely many times. This inelegance may be resolved by passing to enrichment over $\mathcal{O}$. That works best when the base category is $\mathcal{O}$ rather than **Set**, as in Section 7, as $\mathcal{O}$ is $\mathcal{O}$-enriched, while **Set** is not. However, for ease of exposition, we first restrict ourselves to **Set**, then extend to $\mathcal{O}$. This follows Plotkin's exposition in [8]: the use of $\mathcal{O}$ allows a more elegant treatment, but one that is also more complex. Formally, *Nondet*(**Set**) is given by freely making **Set** $\mathcal{O}$-enriched, and equipping each hom with a binary operator $\vee$ that respects the $\mathcal{O}$-structure and is natural in its domain, i.e., that is given pointwise. In the category *Nondet*(**Set**), we model **if** by use of coproducts, sequential composition by composition, **while** by iteration or equivalently by the $\mathcal{O}$-structure, and nondeterministic or by $\vee$.

We now describe *Nondet*(**Set**), together with an elementary explanation of its computational motivation. An object of *Nondet*(**Set**) is a set. An arrow from X to Y is the assignment to each $x \in X$ of a possibly infinite nonempty binary tree with leaves labelled by elements of Y.



The idea is that a program p with input x starts doing a deterministic part of p. Either it stops with output y, or it never stops, or it reaches a nondeterministic choice. In the first case, we would have a tree consisting of one point labelled y. In the second, we will write the infinite leafless tree. In the last, the program may choose left or right: so we would have an initial branching, then continue inductively.

Observe that there are two ways of failing to halt on input x:

1. one could reach a deterministic part of p that fails to halt. For instance, p could be a deterministic program that does not halt on input x.

2. one may follow an infinite branch, each piece of which is deterministic. For instance, one could consistently follow the left branch of **while** $x = 0$ **do** $(x = 0 \vee x = 1)$ on input 0.

In our semantics, we wish to identify the two ways to fail to halt on the grounds that, in both cases, there is no observable output: that is why we did not introduce the

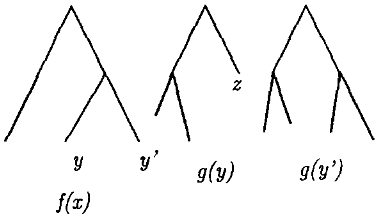


Fig. 1. Component trees.

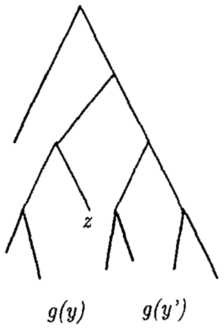


Fig. 2. Composite tree.

empty tree in our second case of the previous paragraph, as we wish to identify that case with that which would yield the infinite leafless tree.

An arrow in *Nondet*(Set) from X to Y may be described algebraically as a partial function f from $[\omega,2] \times X$ into $\omega \times Y$, such that if $f(s,x) = (n,y)$, then if t agrees with s on 0 to $n-1$, then $f(t,x) = (n,y)$. This algebraic definition has computational significance in that the argument $s : \omega \rightarrow 2$ amounts to a choice function, or an oracle, which tells the computation which way to proceed at every stage i. It may appear that our algebraic formulation asserts that we insist upon an oracle to model nondeterminism and that we insist upon it acting in a particular way, but that appearance is only incidental as the definition in terms of trees shows.

We define composition in *Nondet*(Set) as follows: given $f : X \rightarrow Y$, $g : Y \rightarrow Z$, and $x \in X$, define $gf(x)$ to be the tree given by replacing each leaf in $f(x)$ by the labelled tree determined by g evaluated at that label of the leaf. For instance, if $f(x)$, $g(y)$ and $g(y')$ are as shown in Fig. 1, then $(gf)(x)$ would be as shown in Fig. 2.

Algebraically, this may be described as follows: given $f : [\omega,2] \times X \rightharpoonup \omega \times Y$ and $g : [\omega,2] \times Y \rightharpoonup \omega \times Z$, suppose $f(s,x) = (n,y)$, and denote by $s(n)$ the sequence $(s_n, s_{n+1}, \ldots)$. Then, if $g(s(n),y) = (m,z)$, we have $gf(s,x) = (n+m,z)$. If either $f(s,x)$ or $g(s(n),y)$ is undefined, then $gf(s,x)$ is undefined.

The idea of this composition is evident from the ideas of sequential composition of programs and of the maps in *Nondet*(Set). Observe that *Nondet*(Set) is enriched over $\mathcal{O}$, with the partial order on the homs given by pointwise definedness.

There is an evident functor $J : \mathsf{Set} \to \mathit{Nondet}(\mathsf{Set})$, and it has a right adjoint given by

$$PY = \{\text{possibly infinite nonempty binary trees labelled by elements of } Y\},$$

or, equivalently,

$$PY = \{f : [\omega, 2] \to \omega \times Y \mid \text{if } f(s) = (n, y) \text{ and } t \text{ agrees with } s \text{ on } 0 \text{ to } n-1, \text{ then } f(t) = (n, y)\}.$$

Observe that our programming language does have a semantics in $\mathit{Nondet}(\mathsf{Set})$: the semantics of **while** is given by iteration; that of $\vee$ is evident. Observe also that the construction $\vee : \mathit{Nondet}(\mathsf{Set})(X, Y) \times \mathit{Nondet}(\mathsf{Set})(X, Y) \to \mathit{Nondet}(\mathsf{Set})(X, Y)$

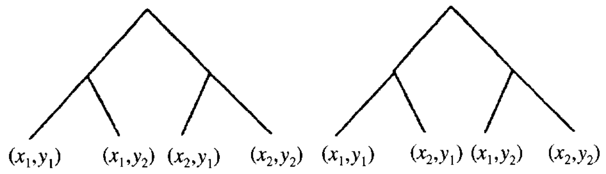
$$(f, g) \mapsto (f \vee g) : (s, a) \mapsto \begin{cases} f(s(1), a) & \text{if } s_0 = 0, \\ g(s(1), a) & \text{if } s_0 = 1 \end{cases}$$

is natural in X as an object of Set. So the hypotheses of Propositions 5 and 6 are satisfied, and hence $\vee$ is represented by $\cup$. Observe that $\cup$ respects the $\mathcal{O}$-structure given by definedness: that fact will follow automatically from our formalism of Section 7.

We shall adopt $\mathit{Nondet}(\mathsf{Set})$ as our primitive category in which to model nondeterminism. However, we have not yet tackled fundamental issues. First, if our programming language is typed, we want an account of programs with several variables. We should also like accounts of recursively defined types and higher-order types: these lead us to move beyond $\mathit{Nondet}(\mathsf{Set})$. Second observe that $\vee$ on $\mathit{Nondet}(\mathsf{Set})$ is neither associative, nor commutative, nor idempotent; but in our leading example of the action of an external agent, the programming $\vee$ is associative, commutative, and idempotent. So we seek an account of equivalence of arrows in $\mathit{Nondet}(\mathsf{Set})$. Third, we want a study of finite approximation to incorporate **print**. Finally, we shall see how to recover the classical powerdomains. These tasks constitute the remaining sections of the paper.

3. Programs with several variables

In this section we begin the study of types. The category Set is cartesian closed and cocomplete, so we can speak of (finite) product types, sums, and positively recursively defined types. Binary sums are used to model conditional statements, and $J : \mathsf{Set} \to \mathit{Nondet}(\mathsf{Set})$ preserves sums: that is why the semantics of "**if** b **then** p **else** q" where p and q are deterministic, extends in a modular fashion to nondeterministic programs. Product types are used to model programs with several variables such as "**if** $x > 0$ **then** x **else** y". However, the functor J does not preserve finite products; and indeed, finite products on Set do not even extend naturally to a monoidal structure on $\mathit{Nondet}(\mathsf{Set})$: consider the nondeterministic programs $x_1 \vee x_2 : 1 \to X$ and $y_1 \vee y_2 : 1 \to Y$. Given $f : A \to B$ and $g : C \to D$ in a monoidal category the two composites from $A \odot C$ to $B \odot D$ given by $(f \odot D)(A \odot g)$ and $(B \odot g)(f \odot C)$ would be equal. So if J sent the binary product in **Set** to a monoidal structure on *Nondet*(**Set**), it would follow that the two trees



would agree: but they do not, and hence binary product on **Set** does not extend naturally to monoidal structure on *Nondet*(**Set**).

This anomaly can be resolved by recourse to a new category-theoretic definition, that of symmetric predistributive category. For our leading example of a nondeterministic programming language, we have adopted a simple imperative language with a nondeterministic operator. To model that specific language, the notion of symmetric predistributive category is not obviously significant: our nondeterministic operator in that language is associative, commutative, and idempotent, and when one factors *Nondet*(**Set**) by a congruence that forces $\vee$ to satisfy those equations, then the predistributive structure collapses to a distributive structure. However, that is not typical. For instance, if we had a functional language with a nondeterministic operator and we allowed side effects in our language, then the predistributive structure would not collapse under associativity, commutativity and idempotence to a distributive structure. See [10] for details. The same remarks would hold if we were modelling the expressions of an imperative language with nondeterministic expresssions. However, for simplicity, we will not introduce such complex languages in detail solely to motivate the definition of symmetric predistributive category. Instead, we will illustrate how such structure arises before one factors out the various congruences, with this side remark that the structure typically does not collapse under interesting congruence.

We begin by introducing some subsidiary definitions.

Definition 7. A *binoidal category* is a category $\mathscr{C}$ together with, for each object X of $\mathscr{C}$, functors $H_X : \mathscr{C} \to \mathscr{C}$ and $K_X : \mathscr{C} \to \mathscr{C}$ such that for each pair (X, Y) of objects of $\mathscr{C}$, $H_X Y = K_Y X$.

We will denote the joint value of $H_X Y$ and $K_Y X$ by $X \odot Y$.

Definition 8. An arrow $f : X \to Y$ in a binoidal category is called *central* if for each arrow $g : X' \to Y'$, the diagrams

$$\begin{tikzcd}
X \odot X' \arrow[r, "K_{X'} f"] \arrow[d, "H_X g"'] & Y \odot X' \arrow[d, "H_Y g"] \\
X \odot Y' \arrow[r, "K_{Y'} f"] & Y \odot Y'
\end{tikzcd}$$

and

$$\begin{tikzcd}
X' \odot X \arrow[r, "K_X g"] \arrow[d, "H_{X'} f"'] & Y' \odot X \arrow[d, "H_{Y'} f"] \\
X' \odot Y \arrow[r, "K_Y g"] & Y' \odot Y
\end{tikzcd}$$

commute.

Definition 9. A *premonoidal* category is a binoidal category together with a family of central isomorphisms $a : (X \odot Y) \odot Z \to X \odot (Y \odot Z)$ natural in each argument, an object I, and families of central isomorphisms $l : X \to I \odot X$ and $r : X \to X \odot I$ natural in X, such that

$$\begin{tikzcd}
((W\odot X)\odot Y)\odot Z \arrow[rr, "a\odot Z"] \arrow[d, "a"'] & & (W\odot(X\odot Y))\odot Z \arrow[d, "a"] \\
(W\odot X)\odot(Y\odot Z) \arrow[dr, "a"'] & & W\odot((X\odot Y)\odot Z) \arrow[dl, "W\odot a"] \\
& W\odot(X\odot(Y\odot Z)) &
\end{tikzcd}$$

and

$$\begin{tikzcd}
& X\odot Y \arrow[dl, "r\odot Y"'] \arrow[dr, "X\odot l"] & \\
(X\odot I)\odot Y \arrow[rr, "a"'] & & X\odot(I\odot Y)
\end{tikzcd}$$

commute.

It is trivially true that every monoidal category is a premonoidal category, with $H_X = X \odot \_$ and $K_X = \_ \odot X$.

Definition 10. A *symmetry* on a premonoidal category is a natural family of central isomorphisms $c : X \odot Y \to Y \odot X$ such that $c^2 = 1$ and

$$\begin{tikzcd}
(X \odot Y) \odot Z \arrow[r, "a"] \arrow[d, "c \odot Z"'] & X \odot (Y \odot Z) \arrow[d, "c"] \\
(Y \odot X) \odot Z \arrow[d, "a"'] & (Y \odot Z) \odot X \arrow[d, "a"] \\
Y \odot (X \odot Z) \arrow[r, "Y \odot c"'] & Y \odot (Z \odot X)
\end{tikzcd}$$

commutes. A *symmetric premonoidal* category is a premonoidal category together with a symmetry on it.

There are coherence theorems for premonoidal and symmetric premonoidal categories generalising those for monoidal and symmetric monoidal categories (see [11]): these show that "all diagrams commute", and so we may pass from two to many variables unambiguously.

A *strong premonoidal* functor is a functor that preserves premonoidal structure up to coherent natural central isomorphism. It is called *strict* if the isomorphisms are identities. One may define strong and strict symmetric premonoidal functors along the same lines.

Definition 11. A *predistributive* category is a premonoidal category with finite sums, for which H_X and K_X preserve finite sums. A *symmetric predistributive* category is a predistributive category together with a symmetry.

It is evident how to define strong predistributive functors and variants: they are strong premonoidal functors that preserve finite sums and variants.

Proposition 12. *Nondet*(Set) *is a symmetric predistributive category and* J : Set $\to$ *Nondet*(Set) *is a strict symmetric predistributive functor.*

Proof. This is a routine calculation, but also may be deduced in greater generality from the fact that *Nondet*(Set) is the Kleisli category for a monad on Set. First observe that any monad on Set has a unique strength, as it has a unique, in fact trivial, enrichment over Set. Any strength on a monad on a cartesian closed category with finite sums yields a symmetric predistibutive structure on the Kleisli category, with J strict symmetric predistributive: see [11]. □

The symmetric predistributive structure on *Nondet*(Set) allows a treatment of nondeterministic programs of several variables, a program with input a pair of variables (x, y) of types X and Y being modelled by a map with domain $X \odot Y$, the premonoidal product of X and Y. The condition that $X \odot \_$ preserves finite sums ensures that the semantics of programs with several variables behave well in the presence of conditionals, cf. Walters and colleagues' work on distributive categories [3]. The fact that J is a strict symmetric predistributive functor means that the analysis extends that for deterministic programs of several variables in the evident simple way.

4. Recursively defined types and higher-order types

In order to account for positively recursively defined types, we need to take colimits in our category. The reason is as follows. Consider a positive recursive domain equation, for instance $X = 1 + X$. First observe that simply to define this equation, we have used binary coproducts; so we want the existence of binary coproducts in our category of domains. Now, to obtain a solution to this domain equation, one first takes the initial object 0 of the category: so we want an initial object in our category in addition to binary coproducts, hence all finite coproducts. Next, we apply the functor $1 + (\,)$ to the initial object, and we continue to apply $1 + (\,)$ inductively, yielding a sequence X_n of objects of the category, with, for each n, a map from X_n to X_{n+1}: for $n = 0$, it is the unique such map, and for greater n, it is given inductively by $1 + (\,)$ applied to that map. Finally, we take the colimit X_ω of the sequence: so we want the existence of such ω-directed colimits in our category. This colimit is preserved by $1 + (\,)$, so provides a solution to our domain equation. That domain equation is one for the natural numbers, so one certainly wants an account of such constructions. In making this construction, we have used finite coproducts and ω-directed colimits. They do not generate all colimits, but they do generate a large class of them. So we seek to add colimits to our semantic category.

Since we need to add colimits, the least intrusive way to do so is to add them freely to *Nondet*(Set). In passing to this new cocomplete category, the free cocompletion of *Nondet*(Set), we still want to maintain the modularity of our programs; i.e., we want to preserve the program and type connectives we already have in *Nondet*(Set), namely sequential composition, sums, and the premonoidal structure. We also do not want to identify the semantics of any two programs whose semantics have not already been identified. Such requirements are delicate but achievable modulo a mild size condition: if we let *Nondet*(Set_c) denote the full subcategory of *Nondet*(Set) determined by the countable sets we have:

Theorem 13. *The functor*

$$\begin{aligned} \mathcal{Y} &: \mathit{Nondet}(\mathsf{Set}_c) \to \mathrm{FP}(\mathit{Nondet}(\mathsf{Set}_c)^{\mathrm{op}}, \mathsf{Set}), \\ X &\longmapsto \mathit{Nondet}(\mathsf{Set}_c)(\_, X) \end{aligned}$$

is fully faithful and exhibits the category $\mathrm{FP}(Nondet(\mathsf{Set}_c)^{\mathrm{op}}, \mathsf{Set})$ *of finite product preserving functors from* $Nondet(\mathsf{Set}_c)^{\mathrm{op}}$ *to* Set *as the free cocompletion of* $Nondet(\mathsf{Set})$ *that preserves those finite sums that exist in* $Nondet(\mathsf{Set}_c)$.

The point of this construction is that, if we seek to model positively recursively defined types, we cannot a priori do so in the category $Nondet(\mathsf{Set})$ because it does not have colimits. So this construction gives us a category $\mathrm{FP}(Nondet(\mathsf{Set}_c)^{\mathrm{op}}, \mathsf{Set})$ in which we can give all semantics as we have in previous sections, together with a semantics for positively recursively defined types, in such a way that the original semantics extends while respecting modularity, and without making any further identifications of semantic maps. The freeness in the construction is not really essential: it is just a least intrusive way of getting such a semantic category.

A proof appears of a more general situation than the above theorem in [5], Theorem 6.11. The result gives us what we need, i.e., colimits, for an account of positively recursively defined types, while preserving the existing structure. In fact, one only requires initial objects and colimits of ω-chains for inductive definitions that do not involve binary coproducts in their definition, but since we already require finite coproducts to model the semantics of **if**, we seek countable colimits. We are unaware of an attractive characterization of the free addition only of countable colimits, while respecting some already existing ones.

This construction is not quite as useful as we should like. For any object X of $Nondet(\mathsf{Set}_c)$, the functor $\_ \odot X$ extends to $\mathrm{FP}(Nondet(\mathsf{Set}_c)^{\mathrm{op}}, \mathsf{Set})$, so we obtain some account of programs of several variables. However, we see no way to obtain a premonoidal structure on the whole category $\mathrm{FP}(Nondet(\mathsf{Set}_c)^{\mathrm{op}}, \mathsf{Set})$. So although we extend all our original semantics and have an account of recursively defined types, our account of semantics for programs of several variables does not extend to all pairs of types.

The above construction also gives us a limited account of higher-order structure: each $\_ \odot X$ on $Nondet(\mathsf{Set}_c)$ preserves finite sums, so lifts to a functor on $\mathrm{FP}(Nondet(\mathsf{Set}_c)^{\mathrm{op}}, \mathsf{Set})$ which has a right adjoint $(X \longrightarrow \_)$, which acts as a higher-order type. This is not a complete analysis of higher-order types for two reasons: first, we only have $(X \longrightarrow \_)$ for those X in $Nondet(\mathsf{Set}_c)$, and cannot iterate the process; and second, we have not yet accounted for equivalences of programs, and such a treatment will necessarily fundamentally affect the choice of a higher-order type, as well as affecting everything else. We will return to this in the next section.

Finally, it follows from countable versions of Barr and Wells' [2] Theorem 4.3.5 and Example 4.3 (SSFP) that we have

Theorem 14. *The functor* $\mathscr{Y} : Nondet(\mathsf{Set}_c) \longrightarrow \mathrm{FP}(Nondet(\mathsf{Set}_c)^{\mathrm{op}}, \mathsf{Set})$ *factors through fully faithful* $\mathscr{Y}_c : Nondet(\mathsf{Set}_c) \longrightarrow \mathrm{CP}(Nondet(\mathsf{Set}_c)^{\mathrm{op}}, \mathsf{Set})$, *where the latter category is the full subcategory of* $\mathrm{FP}(Nondet(\mathsf{Set}_c)^{\mathrm{op}}, \mathsf{Set})$ *consisting of those functors that preserve countable products. Moreover, it is equivalent to the category*

PJ-Alg *of PJ-algebras with PJ the monad on* Set *determined by the adjunction of Section* 1.

To understand the idea of PJ-algebras, recall that, in Section 1, we mentioned that there are two canonical ways in which one can obtain an adjunction from a monad. The first way, which has been of primary interest to us here, is the Kleisli construction, as explained in Section 1. The second is by construction of the category of algebras. That construction is studied in depth in [2]. We will not develop that further in this paper, but mention the result for those readers who already know the construction. The reason we mention the above result is that, on occasions, as for instance in Plotkin's LICS talk [9], when one has modelled nondeterminism with a powerdomain P, the category of algebras has been used as a semantic category: so this result shows how the category of algebras appears as a special case of our construction freely adding colimits while preserving some existing ones.

Upon generalising to the category of ω-cpos with least element, as we shall do later, passing to the category of algebras allows an account not merely of recursion on programs but also of a recursion operator as explained in Plotkin's LICS talk [9]. That is sometimes useful, as nondeterminism may arise from a recursive oracle, for instance in complex systems. However, it is not always useful, as an oracle may arise nonrecursively, for instance by human intervention in a program; so we do not regard a recursion operator as essential to modelling nondeterminism. In summary, for a recursion operator, one seems to need algebras, but a recursion operator need not exist even while recursion does exist. That is why we prefer to take the category of algebras as a derived construction, as we have done in this section, rather than as a primitive construction as has been done in the past.

5. Factoring out equivalences

The connective $\vee$ on *Nondet*(Set) is neither associative, nor commutative, nor idempotent; but in our leading example of the action of an external agent, the programming $\vee$ is associative, commutative and idempotent. So we seek to identify arrows in *Nondet*(Set) to take account of that. More generally, we seek an account of factoring out any reasonable equivalence on arrows in *Nondet*(Set).

In factoring out equivalences, we want to preserve the denotational semantics we have already given in *Nondet*(Set), so we need to preserve sequential composition, premonoidal structure and finite sums, and respect $\vee$, i.e., we want a strong symmetric predistributive functor that respects $\vee$. In preserving sequential composition and finite sums, it is natural to restrict attention to those equivalences $\sim$ such that $\sim$ is a congruence, i.e., such that if $f \sim g$, then for any composable h and k, we have $kfh \sim kgh$; and such that $f \sim g$ if and only if for all $x \in X, fx \sim gx$. This is equivalent to $fh \sim gh$ for all h composable with f. For any congruence, $f \sim g$ implies $fx \sim gx$ for all x: we additionally demand the converse. This is an extensionality condition that is sufficient, but not necessary, to preserve finite sums. It is not true of an arbitrary congruence, as one could have $a \vee b \sim b \vee a$ but also $(a,b) \vee (b,a) \not\sim (a,a) \vee (b,b) : \mathbf{2} \to Y$. But we do want our congruences to satisfy this extensionality condition. For instance, suppose f was **if** $x = 1$ **then** $(a \vee b)$ **else** $(b \vee a)$ and g was **if** $x = 1$ **then** $(a \vee b)$ **else** $(a \vee b)$. If we put $a \vee b \sim b \vee a$, we would have $fx \sim gx$ for all $x \in X$. We want to conclude that f is congruent to g, and we believe that any reasonable congruence should allow us to do so. So we say

Definition 15. An *extensional congruence* on *Nondet*(Set) consists of an equivalence relation $\sim$ on each homset *Nondet*(Set)(X, Y) such that $f \sim g$ if and only if for all $x \in X$, $fx \sim gx$.

Extensionality is an inherently natural condition, and it is particularly convenient, as we have

Proposition 16. *For any extensional congruence $\sim$ on Nondet*(Set) *the composite*

$$\mathsf{Set} \to \mathit{Nondet}(\mathsf{Set}) \to \mathit{Nondet}(\mathsf{Set})/\sim$$

has right adjoint $P^{\sim}$.

Proof. Since $\sim$ is extensional, $(\ )^{\sim} : \mathit{Nondet}(\mathsf{Set}) \to \mathit{Nondet}(\mathsf{Set})/\sim$ preserves coproducts, and hence tensors, so we may apply Proposition 5. For a map in *Nondet*(Set)/$\sim$ from X to Y is an equivalence class of maps in *Nondet*(Set), but an element of such is a set of X maps from $\mathbf{1}$ to Y, and any two are equivalent if and only if each of the X pairs is equivalent, so to give the map is to give X maps in *Nondet*(Set)/$\sim$ from $\mathbf{1}$ to Y. □

It follows from Proposition 16 that $(\ )^{\sim}$ is necessarily strong symmetric predistributive, because the premonoidal structure on *Nondet*(Set) agrees with the cartesian product of Set, which is necessarily preserved as a premonoidal structure by the proof of Proposition 12.

Corollary 17. *For any extensional congruence $\sim$ on Nondet*(Set) *that respects $\vee$, the operation $\cup : PX \times PX \to PX$ is sent to a defined operation $\cup^{\sim}$.*

Proof. This follows from the Yoneda lemma. The value of $(\ )^{\sim}$ at X is given by taking equivalence classes under $\sim$ applied to the set of maps from 1, and $\cup^{\sim}$ is determined by its value on elements of the equivalence classes. □

In general, it does not seem possible to give explicit definitions of the above powerdomains.

This result is mildly misleading, as we have ignored the $\mathcal{O}$-structure of *Nondet*(Set). In computational terms, we have ignored recursion. It does return us a monad, and it does give us a convenient partial result; but both our assumptions and our conclusions are mildly weaker than those we ultimately seek. Informally, we have freely adjoined a binary operation on terms in Section 2, and now we have factored it by equations. The category of algebras for the induced monad here is essentially, but not precisely, the category of algebras for those operations and equations. That statement becomes precise when we have $\mathcal{O}$ as our base category and we consider only categories, functors, and operations that are $\mathcal{O}$-enriched, as we do in Section 7.

Consideration of such operations and equations is a general situation applying not only to nondeterminism but also to two nondeterministic operators together with an equational relationship between them, and more generally to any system of operations and equations. However, it does not apply to all monads used as notions of computation by Moggi as they do not all arise from operations and equations in this way. For a specific example, consider the addition of side effects or the study of partiality.

6. Recovering the classical powerdomains

In this section, we use the techniques of Section 5, specifically Corollary 17, to rediscover the convex powerdomain. This section is the fundamental section of the paper. First, it shows precisely how our setting generalises the use of a powerdomain. Second, it gives a formal account of how one might discover the convex powerdomain from a more primitive analysis of nondeterminism. Formally, recovering the upper and lower powerdomains is similar. However, the convex powerdomain is clearly illustrated as more primitive.

First we recall the traditional powerdomains, and the thinking behind their development as explained in [8]. The idea is that one starts with a countable set of states S. Then, for a given initial state, a nondeterministic program has a set of possible outcomes. Taking recursion into account, a program may or may or may not terminate, and if not, may or may not produce any output: we investigate that possibility more closely in the next section. If the set of possible outcomes is infinite, then since we only allow finite branching, it follows by Konig's lemma that nontermination must have been a possibility. So the convex powerdomain $P_c(S)$ is defined to be the set of all nonempty subsets of $S_\perp$ which are either finite or contain $\perp$. This set is then given a natural order, yielding an ω-cpo with least element. This construction has been generalised in several ways to a more general class of domains, but the situation for a countable set is the basic one. For convenience, we will take it, for general X, to be the set of all countable nonempty subsets of $X_\perp$ which are either finite or contain $\perp$. We will do similarly for the other powerdomains too.

For the other powerdomains, one has a slightly less refined analysis: in one case, one ignores nontermination and just asks for the set of possible outcomes, hence $P(S)$; this may be seen as taking the convex powerdomain and deleting $\perp$ from any set that contains it. This identifies any finite set not containing $\perp$ with the union of that set with $\{\perp\}$. For the other case, one identifies all sets containing $\perp$ on the basis that

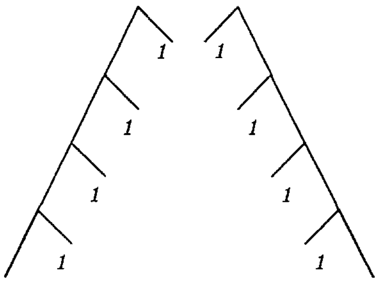


Fig. 3. Two computations.

no output can be guaranteed of them. These powerdomains are again given natural orderings making them into ω-cpo's with least element.

Note that they are motivated by less dynamic arguments than we have made: the emphasis being on a set of possible outcomes rather than a tree of possible behaviours. Nevertheless, we find it striking how easy it is to recover the convex powerdomain by putting a very mild equivalence on our set of trees. The argument is as follows.

As mentioned in Section 5, the programming $\vee$ is associative, commutative and idempotent. So we wish to factor out the minimal equivalence to force the semantic $\vee$ to be likewise. However, we also want **while** $x = 0$ **do** $(x = 0 \vee x = 1)$ to have the same meaning as **while** $x = 0$ **do** $(x = 1 \vee x = 0)$, and this is not assured by ordinary commutativity and associativity of $\vee$, i.e., we want to identify the two computations of Fig. 3.

This is simply resolved by insisting that our congruences respect the $\mathcal{O}$-structure of *Nondet*(**Set**), and $\vee$ continues to respect $\mathcal{O}$-structure after being factored, cf. [1].

We then have

Theorem 18. *Let $\sim$ be the minimal extensional congruence on Nondet*(**Set**) *that respects the $\mathcal{O}$-structure and makes $\vee$ associative, commutative and idempotent. Then $(P^{\sim}, \cup^{\sim})$ is the convex powerdomain.*

Proof. By extensionality, one need only consider maps with domain 1. So consider those maps from 1 to X. For finite trees, it follows by induction that two trees are identified if and only if the sets of labels on their leaves agree. For infinite trees, respect for the $\mathcal{O}$-structure applied to the definitions of commutativity and idempotence force the same result but with a least element to represent undefinedness added. Thus we have the convex powerdomain as restricted to sets, i.e., the set of those subsets of $X_{\perp}$ consisting of the finite subsets together with those countable subsets containing $\perp$, see [8, Section 8, p. 6]. It is routine to verify that $\cup^{\sim}$ amounts to union. $\square$

The upshot of this result is that factoring by extensional $\mathcal{O}$-congruences satisfying the conditions of the theorem reduces the trees to the set-of-possible-outputs view advocated in the traditional literature on powerdomains. We think this fact helps to justify the primacy of the convex powerdomain in the study of nondeterminism, while giving scope for modelling more intensional properties, as for instance done by Simpson [12].

We can go further than this and show that the various powerdomains are the *only* nontrivial constructions one obtains by factoring by congruences subject to mild, natural conditions. We regard this result as a strong vindication of the study of the power domains.[2]

Theorem 19. *There are precisely three nontrivial extensional $\mathcal{O}$-congruences $\sim$ on Nondet*(Set) *that respect $\vee$, and make it associative, commutative and idempotent. These yield the upper, lower and convex powerdomains.*

Proof. Take any congruence $\sim$ on *Nondet*(Set). By Theorem 18, we may consider $\sim$ as a congruence on the Kleisli category for the convex powerdomain. Assuming it is nontrivial, then it follows from the fact that composition with any map from the one point set must preserve $\sim$, that for some set X, two different subsets A and B of $X_\perp$, regarded as maps from 1 to X, are identified. Postcomposing with the characteristic function for one of them, it follows that $\sim$ identifies at least two of the three maps from 1 to 1: choosing 1, $\perp$, or the union. Identifying either of the first two with the last yields the other two classical powerdomains. One identifies all sets containing $\perp$, and the other ignores $\perp$. If one identifies 1 with $\perp$, then since $\sim$ is a congruence, it follows by postcomposition that all maps from 1 to any X are identified with that choosing $\perp$, and so by extensionality, any two maps are identified. □

Finally, observe that forcing $\vee$ to be associative, commutative and idempotent while respecting the $\mathcal{O}$-structure forces the premonoidal structure on *Nondet*(Set) to be sent to a monoidal structure. With premonoidal replaced by monoidal, the discussion about higher-order types at the end of Section 4 returns to the realm of known category theory: $\mathrm{FP}((\mathit{Nondet}(\mathsf{Set}_c)/\sim)^{\mathrm{op}}, \mathsf{Set})$ is then symmetric monoidal closed, allowing iteration of higher-order types respecting equivalences.

7. Finite approximation

There is only one substantial change in moving from sets to ω-cpos to account for finite approximation: equality is no longer the primitive relation between the functional behaviour of programs. The partial order is. For instance, to show that $[\![p]\!] = [\![q]\!]$, we show that $[\![p_n]\!] \sqsubseteq [\![q]\!]$ and $[\![q_m]\!] \sqsubseteq [\![p]\!]$ given $(p_n) \to p$ and $(q_m) \to q$. So we no longer use a congruence $\sim$, but we insert a partial order. In doing so, we introduce new maps!

The reason for introducing new maps is as follows: suppose we wish to refine a hom ω-cpo by forcing $g \sqsubseteq h : A \to B$ (e.g., $x \vee y \sqsubseteq y \vee x$). This may produce a new increasing sequence (f_n). We need a limit for this sequence, but there is no

[2] We should like to thank Gordon Plotkin for pointing this result out to us.

computational reason to force any existing map to be that limit: so we add a new map to act as the limit.

When we introduced and factored out congruences, we considered extensional congruences, i.e., those equivalence relations $\sim$ for which $f \sim g$ if and only if for all x, $fx \sim gx$. The reason was that programs could be observed only by observing their behaviour on input. For the same reason, we need an account of extensionality here.

Mathematically, this can be expressed simply by a strong extensionality condition: we want every map $f : A \longrightarrow B$, new or old, to be determined by giving an A-indexed family of maps $f_a : \mathbf{1}_\perp \longrightarrow B$, subject to the partial order and limiting information of A. This is the assertion, in categorical terms, that each $A \in \mathit{Nondet}(\mathcal{O})/\sqsubseteq$ is the tensor of A, regarded as an object of $\mathcal{O}$, with $\mathbf{1}_\perp$, regarded as an object of $\mathit{Nondet}(\mathcal{O})/\sqsubseteq$. (See Section 1). So, we will insist that $(\ )^{\sqsubseteq} : \mathit{Nondet}(\mathcal{O}) \to \mathit{Nondet}(\mathcal{O})/\sqsubseteq$ preserves tensors (equivalently, tensors with $\mathbf{1}_\perp$).

This condition was redundant in the discrete setting because it was a consequence of restricting to extensional congruences and not introducing new maps. Adding new maps allows us to model **print**, and preservation of tensors ensures preservation of conditionals, because sums can be treated as tensors here. We introduce **print** following Plotkin [8]. The idea, in a simple setting, is that a program prints a succession of 0's and 1's, and that is the output of the program. The reason this affects the modelling of nondeterminism is that the printing need not all happen when the program halts, but may occur during the running of the program. So in particular, a program may never halt but may have an output that is gradually generated. So in order to model **print** adequately, one needs to model the notion of finite approximation of output, and implicitly dataflow composition, whereas our setup so far has been directed towards modelling sequential composition. It was the semantics of **print** that motivated the binarily generable trees of [8], and it is therefore binarily generable trees that we wish to rediscover here.

Aside from this, we routinely generalise the discrete case with evident modifications. First observe that $\mathcal{O} = \omega$-cpos with least elements, being an $\mathcal{O}$-category may be used to define the semantics of **while**. We identify $\mathcal{O}$ with the category of ω-cpos and partial maps. So in order to allow the semantics of **while** on the deterministic part of our language to lift to $\mathit{Nondet}(\mathcal{O})$, we require that $\mathit{Nondet}(\mathcal{O})$ be an $\mathcal{O}$-category and J be an $\mathcal{O}$-functor. Moreover, we want $\mathit{Nondet}(\mathcal{O})$ and J to restrict to **Set** as in Section 2, and we want $f \sqsubseteq g$ if and only if $f(x) \sqsubseteq g(x)$ for all $x \in A$. So we define a map in $\mathit{Nondet}(\mathcal{O})$ to be a continuous partial function $f : [\omega, \mathbf{2}] \times A \rightharpoonup \omega \times B$ with $[\omega, \mathbf{2}]$ and ω regarded as discrete, such that if $f(s,a) = (n,b)$ and t agrees with s on $0, 1, \dots, n-1$ then $f(t,a) = (n,b)$. Moreover $f \sqsubseteq g$ if and only if whenever $f(s,a) = (n,b)$, then $g(s,a) = (n,b')$ for some b' with, $b \sqsubseteq b'$. In terms of trees this means that $f \sqsubseteq g$ if and only if for all $x \in A$, if f stops at a leaf with output y, then g stops at the same leaf with output greater than or equal to y. This enriches $\mathit{Nondet}(\mathcal{O})$, makes J $\mathcal{O}$-enriched, and extends the discrete case. Our technique now is to successively strengthen $\sqsubseteq$, i.e., to make things more related.

Observe that the semantics for **while** can be given in terms of either fixpoints or iteration: it is easy to verify that they agree, and we can duly give semantics to all our language other than **print** in $Nondet(\mathscr{C})$.

Definition 20. An *insertion* in $Nondet(\mathscr{C})$ consists of, for each A and B in $\mathscr{C}$, a partial order $\sqsubseteq$ on $Nondet(\mathscr{C})(A,B)$, which is preserved by pre- and post-composition in $Nondet(\mathscr{C})$.

Definition 21. An insertion is called *extensional* when $f \sqsubseteq g$ if and only if $f(x) \sqsubseteq g(x)$ for all $x \in A$.

Definition 22. An *insertion quotient* is the universal tensor-preserving identity on objects $\mathscr{C}$-functor $(\ )^{\sqsubseteq} : Nondet(\mathscr{C}) \longrightarrow Nondet(\mathscr{C})/\sqsubseteq$ that inserts the insertion and preserves $\vee$ as a family of continuous maps.

This means that an insertion quotient is, in a precise sense, the minimal way to add the given insertion to the partial order, and recover the structure of an ω-cpo with least element, and keep $\vee$ well defined. One just freely adds whatever data is needed to each of the partial orders to ensure that all the conditions are satisfied. It follows from abstract category theory that the functor giving the quotient has a right adjoint.

Theorem 23. *For any insertion quotient, the $\mathscr{C}$-functor$(\ )^{\sqsubseteq}J : \mathscr{C} \longrightarrow Nondet(\mathscr{C})/\sqsubseteq$ has a right adjoint given by $Nondet(\mathscr{C})/\sqsubseteq((J1)^{\sqsubseteq}, \_)$. Moreover, $\vee^{\sqsubseteq}$ is representable.*

Proof. The first claim is an instance of Corollary 3. The second holds because $\vee$ is preserved as a family of continuous maps and by the Yoneda lemma, by the same argument as in several of the proofs of Section 1. □

We consider two specific examples of extensional insertions and quotients. First consider the least extensional congruence generated by adding $f \sqsubseteq f \vee f$ for every f. This means that $f \sqsubseteq g$ if and only if for all $x \in A$, if f stops at a leaf with output y, then g stops at that leaf or along every path that extends that leaf, each time with a value at least that of y. We will denote this insertion by $\sqsubseteq_i$. Then we have

Theorem 24. *The insertion quotient of $\sqsubseteq_i$ gives the binarily generable trees subject to the equivalence relation making labelled trees only dependent upon their shapes, the labels of the leaves and the limiting values.*

Proof. This is implicit in [7, 13]. A binarily generable tree is a binary branching tree in which each node is labelled by an element of the target ω-cpo, with the sequence of elements increasing along any branch. So each such tree is a limit of a sequence of our generic trees, the difference being that binarily generalable trees may have a limit along an infinite branch, whereas our generic trees do not; but adding this insertion allows such limits. One then factors so that only the limit along each branch is counted, rather than the specific sequence that leads to the limit. □

Observe that this now allows us to give semantics to **print** as desired.

Now consider the least extensional insertion generated by making $\vee$ idempotent, associative and commutative. Note that this strengthens the above example. Let us denote the insertion by $\sqsubseteq_c$. There are several definitions of convex powerdomain on ω-cpos which agree on SFP domains. For the definition of convex powerdomain as a free monoid with extra structure, we have

Theorem 25. *The insertion quotient of $\sqsubseteq_c$ determines the convex powerdomain. Moreover, up to finite approximation, it is the Egli–Milner order on $Nondet(\mathcal{O})$.*

Proof. The first claim is proved in [7, 13]. For the second, modulo finite approximations, $f \sqsubseteq_c g$ if and only if for all $x \in A$, for all s there exists a t such that $f(s,x) \sqsubseteq g(t,x)$ and for all t there exists an s such that $f(s,x) \sqsubseteq g(t,x)$. □

The lower and upper powerdomains are recovered by strengthening the insertion appropriately.

8. Further work

In considering giving semantics for the concurrent execution of $f: X \to Y$ and $g : X' \to Y'$ we can introduce an operation $f \mid g : X \odot X' \to Y \odot Y'$. The tree for $f \mid g$ is constructed by considering all interleavings of the trees for f and g. Observe that this does *not* give a monoidal structure, or even a premonoidal structure on $Nondet(\mathbf{Set})$ because $f|g$ is not the composite of $(f|\mathbf{1})$ and $(\mathbf{1} \mid g)$. However, $\mid$ is evidently associative with an identity and respects sequential composition in $Nondet(\mathbf{Set})$.

For an account of concurrency we need a parallel operation and some means of pruning unwanted paths in the trees. We believe we can introduce such operations on $Nondet(\mathbf{Set})$. Such operations would provide the basis for an account of some features of concurrent systems, in particular synchronisation, and deadlock. Then using our treatment of equivalences in Section 5 one can acount for various equivalences arising from concurrent computation.

9. Conclusions

We have reformulated the semantics of nondeterminism, taking sequential composition and $\vee$ as primitive, doing so in the presence and with an account of the familiar program and type constructors. We have also accounted for equivalences and finite approximation by general, computationally natural constructions and we have indicated how one may commence a treatment of concurrency based on nondeterminism and appropriate equivalences. In particular, we have formulated a generic semantics for nondeterminism and a single general construction to account for equivalences: this gives a description of all powerdomains, illustrating the three classical powerdomains as three instances of the same mathematical construction.

References

[1] K.R. Apt and G.D. Plotkin, A Cook's tour of countable nondeterminism, in: *Proc. 9th ICALP*, Lecture Notes in Computer Science, Vol. 115 (Springer, Berlin, 1981) 479–494.
[2] M. Barr and C. Wells, *Toposes, Triples and Theories* (Springer, Berlin, 1985).
[3] A. Carboni, S. Lack and R.F.C. Walters, Introduction to extensive and distributive categories, *J. Pure Appl. Algebra* **84** (1993) 145–158.
[4] P. Di Giantonio, F. Honsell and G. Plotkin, Uncountable limits and the λ-calculus, *Nordic J. Comput.* **2** (1995) 126–145.
[5] G.M. Kelly, *Basic Concepts of Enriched Category Theory*, London Math. Soc. Lecture Notes Ser. 64 (Cambridge Univ. Press, Cambridge, 1982).
[6] E. Moggi, Computational lambda-calculus and monads, in: *4th LICS Conf.* (1989) 14–23.
[7] G.D. Plotkin, A powerdomain construction, *SIAM J. Comput.* **5** (1976) 452–486.
[8] G.D. Plotkin, Domains, Edinburgh University Department of Computer Science Postgraduate Theory Course Notes, 1983.
[9] G.D. Plotkin, Type theory and recursion (extended abstract), in: *8th LICS Conf.* (IEEE Computer Society Press, 1993) 374.
[10] A.J. Power and E.P. Robinson, On the categorical semantics of types, draft, 1994.
[11] A.J. Power and E.P. Robinson, Premonoidal categories and notions of computation, in: *LDPL '95, Math. Structures in Computer Science*, to appear.
[12] A.K. Simpson, The convex powerdomain in a category of posets realized by cpos, in: *Proc. CTCS '95*, Lecture Notes in Computer Science, Vol. 953 (Springer, Berlin, 1995) 117–145.
[13] M.B. Smyth, Power Domains, *J. Comput. System Sci.* **16** (1978) 23–36.
[14] R. Street and R. Walters, Yoneda structures on 2-categories, *J. Algebra* **50** (1978) 350–379.
[15] G. Winskel, Powerdomains and modality, in: *Foundations of Computation Theory*, Lecture Notes in Computer Science, Vol. 158 (Springer, Berlin, 1983) 505–514.