Electronic Notes in Theoretical Informatics And Computer Science

ENTICS
https://entics.episciences.org

Volume 5
Proceedings of MFPS 2025

Initial Algebras of Domains via Quotient Inductive-Inductive Types

Simcha van Collem[a,1,4] Niels van der Weide[a,2,5] Herman Geuvers[a,b,3]

[a] *Institute for Computing and Information Sciences*
Radboud University
Nijmegen, The Netherlands

[b] *Faculty of Mathematics and Computer Science*
Technical University Eindhoven
The Netherlands

Abstract

Domain theory has been developed as a mathematical theory of computation and to give a denotational semantics to programming languages. It helps us to fix the meaning of language concepts, to understand how programs behave and to reason about programs. At the same time it serves as a great theory to model various algebraic effects such as non-determinism, partial functions, side effects and numerous other forms of computation.
In the present paper, we present a general framework to construct algebraic effects in domain theory, where our domains are DCPOs: directed complete partial orders. We first describe so called *DCPO algebras* for a signature, where the signature specifies the operations on the DCPO and the inequational theory they obey. This provides a method to represent various algebraic effects, like partiality. We then show that initial DCPO algebras exist by defining them as so called *Quotient Inductive-Inductive Types* (QIITs), known from homotopy type theory. A quotient inductive-inductive type allows one to simultaneously define an inductive type and an inductive relation on that type, together with equations on the type. We illustrate our approach by showing that several well-known constructions of DCPOs fit our framework: coalesced sums, smash products and free DCPOs (partiality and power domains). Our work makes use of various features of homotopy type theory and is formalized in Cubical Agda.

Keywords: domain theory, quotient inductive-inductive types, algebra, algebraic effects.

[1] Email: simcha.vancollem@ru.nl
[2] Email: nweide@cs.ru.nl
[3] Email: herman@cs.ru.nl

[4] The first author was supported by ERC grant COCONUT (grant agreement no. 101171349), funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.
[5] The second author was supported by the NWO project "The Power of Equality" OCENW.M20.380, which is financed by the Dutch Research Council (NWO).

Rec'd Apr 3,2025; Pub'd Dec 15,2025 Proceedings Available Online at
10.46298/entics.16512 ENTICS MFPS 41 Proceedings

1 Introduction

Domain theory was developed in the late 1960s by Dana Scott [21] as a mathematical theory of computation and to give a *denotational semantics* to programs and programming languages, one that abstracts away from the operational semantics of a program, and describes the semantics of a program in terms of the mathematical function it denotes. Domain theory allows one to give a high level meaning to programs and program constructions and to fix the meaning of concepts from programming languages. It also allows to reason about programs on a higher level of abstraction. Since the 60s, programming languages have evolved, introducing many new programming paradigms. To keep up with the rapid development of these new constructs in programming languages, and to fully understand how they work, we also need to be able to describe these using denotational semantics.

Algebraic effects [20] have been introduced to represent computational effects in programming, such as state, exceptions, nondeterminism, non-termination, input-output, and many more. They allow factoring out effectful computations from pure computations and can be composed easily. An effect is defined as a set of operations and an (in)equational theory these operations should obey. For example, non-termination has two operations; one operation represents the non-termination, while the other represents the possible returned value.

In domain theory these algebraic effects are a *directed complete partial order* (DCPO) D together with operations on D which obey an inequational theory, and we call these DCPO algebras. Of particular importance is the *initial DCPO algebra*. For example, for partiality, the inequational theory states that the non-termination operation should be smaller than all possible return values, as non-termination gives the least amount of information about the return value of a partial function. Another example is given by the *powerdomain* [19]. Its inequational theory is similar to that of a join-semilattice, and it is used to represent nondeterminism.

There are various constructions of initial DCPO algebras [13]. However, these constructions use power sets, and thus they are only suitable in impredicative foundations. In this paper, we present an alternative construction for initial DCPO algebras. The main idea is to use *quotient inductive-inductive types* (QIITs) [3]. Using QIITs, we can define a DCPO by specifying operations and inequalities. This idea was also used by Altenkirch, Danielsson, and Kraus [4] to construct the partiality monad. We use this mechanism to show that one can define initial algebras for a suitable notion of signature. Note that this construction is predicative, because it does not use power sets.

1.1 Contribution and Overview

In [4] a quotient inductive-inductive type (QIIT, [3]) is used to construct the non-termination effect in domain theory. A quotient inductive-inductive type is a construction from homotopy type theory that allows one to simultaneously define an inductive type with a quotient on it, in parallel with an inductively defined relation on this type. In the present paper, we extend the method of [4] and present DCPO algebras as a general framework for constructing algebraic effects in domain theory.

We start this paper by recalling quotient inductive-inductive types in Section 2. After that we describe and compare several methods that one can use to construct DCPO algebras in Section 3. We also discuss the main idea of our construction in that section. In Sections 4 and 5, we define a notion of signature and algebra for it, and we illustrate these notions using the Plotkin powerdomain. We construct an initial algebra for every signature in Section 6, and we present numerous examples in Section 7. Finally, we discuss related work in Section 8, and we conclude in Section 9.

1.2 Formalization

As a foundation, our work uses homotopy type theory (HoTT). In particular, we use QIITs and some of the arguments make use of function extensionality. We do not need to use the univalence axiom in our work. Our work is formalized using Cubical Agda [26]. The artifact [24] includes a file with references to

$$\frac{}{[] : \mathrm{FinSet}} \qquad \frac{x : \mathbb{N} \qquad xs : \mathrm{FinSet}}{x :: xs : \mathrm{FinSet}}$$

$$\textsc{Cons-Comm}\ \frac{x, y : \mathbb{N} \qquad xs : \mathrm{FinSet}}{x :: y :: xs = y :: x :: xs} \qquad \textsc{Cons-Dup}\ \frac{x : \mathbb{N} \qquad xs : \mathrm{FinSet}}{x :: x :: xs = x :: xs}$$

$$\textsc{FinSet-Set}\ \frac{xs, ys : \mathrm{FinSet} \qquad p, q : xs = ys}{p = q}$$

Fig. 1. Constructors for FinSet

the Agda code for all the definitions and theorems. [6]

2 Preliminaries on Quotient Inductive-Inductive Types

We start this paper by recalling *quotient inductive-inductive types* (QIIT) [3,15]. In essence, quotient inductive-inductive types combine the ideas of *inductive-inductive types* (IIT) [18] and *quotient types* (QIT) [23]. Inductive-inductive types are used to define a type A together with a type family on A, and quotient inductive types are used to define types by specifying constructors and equations between them.

To get an understanding of what quotient inductive-inductive types are, we consider an example, and we refer the reader for a formal definition to the literature [3]. In addition, we only look at how to specify QIITs and we do not give their elimination rules. Let us start with the type of "finite sets of natural numbers", which can be represented as a QIT in various ways, see [12]. Its constructors are given in Figure 1. The $[]$ and $x :: xs$ constructors are the same as one would expect for regular lists. Apart from the regular point constructors, we also have path constructors, which states that FinSet actually is the type of finite sets of natural numbers. First, Cons-Comm makes sure that the order of the elements in a finite set does not matter. Secondly, Cons-Dup says that adding an element is idempotent. Combining these two path constructors gives us the structure we would expect. However, by introducing these path constructor, we now have multiple paths between finite sets. For example, we can identify the list $x :: x :: []$ with $x :: []$ either by using Cons-Dup, or by the composition of Cons-Dup and Cons-Comm. These paths are not equal to each other. This introduces extra structure which is not present when we consider finite multisets. We therefore add the path constructor FinSet-Set. As this path constructor asserts that FinSet is a set, we typically write this by adding a rule with the conclusion isSet(FinSet).

Next up, we shift our focus to an IIT. As an example, we consider the type of sorted lists [18, Example 3.2]. Its constructors are given in Figure 2. The predicate $x \leq_L xs$ asserts that x is smaller than all elements of the sorted list xs. This predicate allows us to define the type SortedList. We have that $[]$ is sorted, and if xs is sorted and we have a proof p of the fact that $x \leq_L xs$, then we have that $x ::\langle p\rangle\, xs$ is again a sorted list. To define the predicate, we know that x is always smaller than all elements of the empty list. Furthermore, if p is a proof of the fact that $x \leq_L xs$, then from $n \leq x$ we can conclude that n is also smaller than all elements of $x ::\langle p\rangle\, xs$.

Finally, we combine the previous two example into a QIIT. This gives us sorted lists where any two consecutive elements are unequal. Its constructors are given in Figure 3. We again have the $[]$ and $x ::\langle p\rangle\, xs$ constructors to create sorted lists and corresponding constructors to show that these lists are sorted. We now also add a constructor which states that $x ::\langle q\rangle\, x ::\langle p\rangle\, xs$ and $x ::\langle p\rangle\, xs$ are equal. Again, like in the first example, we also add a set truncation constructor to remove the extra structure we are not interested in.

[6] A clickable html version can be found here: https://simchavc.github.io/artifacts/qiits/Paper.html.

$$\frac{}{[] : \text{SortedList}} \qquad \frac{x : \mathbb{N} \qquad xs : \text{SortedList} \qquad p : x \leq_L xs}{x ::\langle p\rangle\ xs : \text{SortedList}}$$

$$\frac{x : \mathbb{N}}{x \leq_L []} \qquad \frac{n, x : \mathbb{N} \qquad xs : \text{SortedList} \qquad p : x \leq_L xs}{n \leq x \to n \leq_L x ::\langle p\rangle\ xs}$$

Fig. 2. Constructors for SortedList

$$\frac{}{[] : \text{StrictSortedList}} \qquad \frac{x : \mathbb{N} \qquad xs : \text{StrictSortedList} \qquad p : x \leq_L xs}{x ::\langle p\rangle\ xs : \text{StrictSortedList}}$$

$$\frac{x : \mathbb{N} \qquad xs : \text{StrictSortedList} \qquad p : x \leq_L xs \qquad q : x \leq_L x ::\langle p\rangle\ xs}{x ::\langle q\rangle\ x ::\langle p\rangle\ xs = x ::\langle p\rangle\ xs} \qquad \frac{}{\text{isSet}(\text{StrictSortedList})}$$

$$\frac{x : \mathbb{N}}{x \leq_L []} \qquad \frac{n, x : \mathbb{N} \qquad xs : \text{StrictSortedList} \qquad p : x \leq_L xs}{n \leq x \to n \leq_L x ::\langle p\rangle\ xs}$$

Fig. 3. Constructors for StrictSortedList

3 Constructions of DCPOs

In this section, we discuss two constructions of *directed complete partial order* (DCPO) that can be used for DCPO algebras. The first way is given by *presentations* of DCPOs [13]. Intuitively, a presentation specifies a basis and relations on elements on that basis. From a presentation one can construct a DCPO in a way similar to the rounded ideal completion [2]. The other construction is by using quotient inductive-inductive types, following ideas by Altenkirch, Danielsson, and Kraus [3].

Let us start by explaining why we are interested in DCPOs rather than ω-CPOs. An ω-CPO is a partial order where every increasing sequence $(d_i)_{i\in\mathbb{N}}$, has a least upper bound. The notion of DCPO is stronger: here we require that every *directed* family has a least upper bound. Recall that a family $\alpha : I \to D$ is directed if I is inhabited, and for all $i, j : I$, there exists a $k : I$ such that $\alpha(i), \alpha(j) \sqsubseteq \alpha(k)$. Concretely, a DCPO is given by a partially ordered set D such that every directed family has a least upper bound. While every DCPO is also an ω-CPO, proving that every ω-CPO is a DCPO requires the index types I to be countable and the axiom of choice [16].

If one works predicatively with DCPOs, then one has to keep careful track of the universe levels involved [8,9]. The type of DCPOs is indexed by three universe levels. More specifically, we have a type $\text{DCPO}_{\mathcal{U}_1,\mathcal{U}_2,\mathcal{U}_3}$ that consists of DCPOs D where the carrier lives in $\mathcal{U}_1$ and the order $\sqsubseteq$ is valued in $\mathcal{U}_2$ (we have $\sqsubseteq : D \to D \to \mathcal{U}_2$) and such that every directed family indexed by a type $I : \mathcal{U}_3$ has a least upper bound. In the formalization the universe levels are explicitly recorded, but in this paper we leave them implicit.

3.1 The Rounded Ideal Completion and Presentations

We start by discussing presentations, and to introduce these, we first discuss the *rounded ideal completion*. The rounded ideal completion gives us a way to construct DCPOs by specifying an *abstract basis* [2]. Elements of the basis are the basic approximations for the elements in the DCPO that we construct.

Definition 3.1 A pair $(B, \prec)$, where $\prec : B \to B \to \Omega$, is an **abstract basis** if $\prec$ is transitive, and it has the following interpolation properties:

- if $x : B$, then there exists $y : B$ such that $y \prec x$;

- if $x_1, x_2, z : B$ such that $x_1, x_2 \prec z$, then there exists $y : B$ such that $x_1, x_2 \prec y \prec z$.

An instance of an abstract basis is given by the ordered rational numbers [7, Definition 106]. Formally, we define the set B to be the collection of rational numbers, and we say that $p \prec q$ if $p < q$. The first condition says that for every $p : \mathbb{Q}$ we have r such that $p < r$, and the second condition says that whenever we have rational numbers $p_1, p_2, q : \mathbb{Q}$ such that $p_1 < q$ and $p_2 < q$, we have another rational number r such that $p_1, p_2 < r < q$.

We now consider rounded ideals of such an abstract basis $(B, \prec)$. These ideals form the underlying type of the rounded ideal completion of B. Following the intuition that these ideals should approximate information, they are defined as directed lower-subsets of B.

Definition 3.2 Let $(B, \prec)$ be an abstract basis. A predicate $X \colon B \to \Omega$ [7] is a **rounded ideal** if X is an directed lower set. The type of rounded ideals, R-Idl$(B, \prec)$, forms a DCPO with subset inclusion as the ordering and union of subsets as the supremum.

The rounded ideal completion allows us to construct continuous DCPOs by specifying a basis. For instance, we can construct the Cantor and Baire space as the rounded ideal completion of lists ordered by the prefix relation [7, Definition 87], and intervals of real numbers as the rounded ideal completion of rational intervals ordered by reverse strict inclusion [7,25].

Note that an abstract basis only specifies basis elements and an order on them. We are thus unable to specify inequalities between basis elements and suprema of collections of basis elements, which means we cannot require Scott continuity. For this reason, we consider the more general notion of *DCPO presentations* [13].

Definition 3.3 A **DCPO presentation** consists of a preorder P and a binary relation $\triangleleft$ on P and directed subsets of P.

Intuitively, the preorder P contains the generator elements for the DCPO that we present. Whenever a generator p is covered by a directed subset U, written $p \triangleleft U$, we intuitively think of p being below the supremum of U in the presented DCPO. Bidlingmaier, Faissole, and Spitters use presentations, although for ω-CPO rather than DCPOs, to study the semantics of probabilistic programming languages [5, Assumption 1]. They assume the existence of the free ω-CPO as an axiom, and they discuss various justifications such as impredicativity and QIITs.

One way to justify the existence of the free DCPO $\overline{P}$ for a presentation, is via an impredicative construction similar to the rounded ideal completion. Specifically, we say that $I \subseteq P$ is an *ideal* if it is a lower set, and for each $U \subseteq I$ such that $p \triangleleft U$, we have $p \in I$. The type of ideals, Idl(P), has a complete lattice structure. For a subset $M \subseteq P$, we write $\langle M \rangle$ for the least ideal containing M. The presented DCPO $\overline{P}$ is defined to be the least sub-DCPO of Idl(P), containing all $\langle \{q \mid q \leq p\} \rangle$ for $p : P$. Note that $\overline{P}$ in general lives in a higher universe than P, unless we work in impredicative foundations.

3.2 Quotient Inductive-Inductive Types

Another way to construct DCPOs is by using quotient inductive-inductive types [3]. The reason why one can use QIITs to construct DCPOs, is because they allow us to simultaneously define a type A together with a relation R between A and A by specifying constructors for both A and R. Because A and R are defined simultaneously, the constructors of A and R may refer to each other.

This idea was used by Altenkirch, Danielsson, and Kraus to construct the partiality monad [4]. More specifically, they construct the free ω-CPO as a QIIT, and the constructors of this QIIT are given in Figure 4. In the remainder of this paper, we generalize their work to construct DCPO algebras.

We illustrate the idea behind our development using an example. Instead of constructing the free pointed DCPO as a QIIT, we show how to construct another example, namely the *powerdomain*. The powerdomain gives a more interesting QIIT, because it has recursive operations and one needs to add

[7] Note that we think of X as subset $X \subseteq B$.

$$\bot : A_\bot \qquad \frac{a : A}{\eta(a) : A_\bot} \qquad \frac{a \sqsubseteq b \qquad b \sqsubseteq a}{a = b} \qquad \frac{s : \mathbb{N} \to A_\bot \qquad p : \prod_{n:\mathbb{N}} s(n) \sqsubseteq s(n+1)}{\bigsqcup(s,p) : A_\bot}$$

$$x \sqsubseteq x \qquad \frac{x \sqsubseteq y \qquad y \sqsubseteq z}{x \sqsubseteq z} \qquad \bot \sqsubseteq x \qquad \frac{n : \mathbb{N}}{s(n) \sqsubseteq \bigsqcup(s,p)} \qquad \frac{\prod_{n:\mathbb{N}} s(n) \sqsubseteq x}{\bigsqcup(s,p) \sqsubseteq x}$$

Fig. 4. Constructors for $A_\bot$ and $\sqsubseteq$

constructors to the QIIT expressing the Scott continuity of each operation. In addition, one can construct the partiality monad for DCPOs in more elementary ways, see the work by Escardó and Knapp [11, Theorem 1] and by De Jong and Escardó [8, Theorem 25].

Given a DCPO D, the powerdomain $\mathcal{P}(D)$ over D is defined to be the free DCPO generated by constructors $\{-\} : D \to \mathcal{P}(D)$ and $\cup : \mathcal{P}(D) \to \mathcal{P}(D) \to \mathcal{P}(D)$ such that $\cup$ is associative, commutative, and idempotent. We construct the DCPO $\mathcal{P}(D)$ together with its order $\sqsubseteq$ as the quotient inductive-inductive type with the point and path constructors given in Figures 5, 6 and 7. We omit the constructors which state that $\{-\}$ and $\cup$ are monotone. Note that we write down the equations of the powerdomain as inequalities, because the formalism that we introduce in this paper, is based on inequalities rather than equalities.

There are a couple of interesting aspects to this quotient inductive-inductive type. First, let us note the key difference with the QIIT by Altenkirch, Danielsson, and Kraus [4]. Since we look at DCPOs, every directed family must have a supremum rather than only chains indexed by the natural numbers. For this reason, the constructor $\bigsqcup$ quantifies over all possible directed families, whose index type lives in some universe $\mathcal{U}$. The type $\mathcal{P}(D)$ thus lives in a larger universe than $\mathcal{U}$, and, if D also lives in $\mathcal{U}^+$, the powerdomain $\mathcal{P}(D)$ is a type in the successor universe $\mathcal{U}^+$.

The second interesting aspect of the definition of this QIIT is that, in essence, it is purely specified by describing the constructors and the inequalities. On top of the constructions and rules for a DCPO (Figure 5), the constructors of the powerdomain and their continuity are specified in Figure 6, and the inequalities are specified in Figure 7. If we were to specify any other algebraic structure on DCPOs, we would only need to modify the constructors in Figures 6 and 7. In the remainder, we generalize this construction, and we show how to use QIITs to construct a wide variety of algebraic structures on DCPOs.

4 Signatures

In this section we define a notion of signature for DCPO algebras, and we demonstrate this notion using the Plotkin powertheory for a fixed DCPO D [2,19]. Our notion of signature is inspired by universal algebra where algebraic structures are described by specifying operations and equations. However, our signatures are specified by operations and inequalities instead.

Let us start by defining how to specify operations. A single operation is represented using a monomial.

Definition 4.1 A **monomial** M consists of an arity $M_B : \mathcal{U}$ and a DCPO M_C. We define $[\![M]\!]$ as the functor which sends a DCPO X to the DCPO $(M_B \to X) \times M_C$. We refer to M_C as the constant of the functor $[\![M]\!]$.

A monomial M describes an operation $[\![M]\!](X) \to X$ on a DCPO X. The Plotkin powertheory of D has two operations, namely the inclusion operation $D \to X$ and the formal union operation $X \to X \to X$. The inclusion operation is represented by a monomial with arity $\bot$ and constant D, and formal union can be represented by a monomial with arity Bool and the unit DCPO as the constant.

To package multiple operations together, we consider families of monomials, called *presignatures*.

$$\frac{}{x \sqsubseteq x} \qquad \frac{x \sqsubseteq y \qquad y \sqsubseteq z}{x \sqsubseteq z} \qquad \frac{}{\mathrm{isProp}(x \sqsubseteq y)}$$

$$\frac{x \sqsubseteq y \qquad y \sqsubseteq x}{x = y} \qquad \frac{}{\mathrm{isSet}(\mathcal{P}(D))}$$

$$\bigsqcup : \textstyle\prod_{\alpha : I \to \mathcal{P}(D)} \mathrm{isDirected}(\alpha) \to \mathcal{P}(D)$$

$$\frac{\alpha : I \to \mathcal{P}(D) \qquad \delta : \mathrm{isDirected}(\alpha)}{\prod\limits_{i:I} \alpha(i) \sqsubseteq \bigsqcup(\alpha, \delta)}\ \bigsqcup\text{-Is-Sup-1} \qquad \frac{\alpha : I \to \mathcal{P}(D) \qquad \delta : \mathrm{isDirected}(\alpha)}{\prod\limits_{v:\mathcal{P}(D)} \mathrm{isUpperbound}(v, \alpha) \to \bigsqcup(\alpha, \delta) \sqsubseteq v}\ \bigsqcup\text{-Is-Sup-2}$$

Fig. 5. Constructors for the DCPO structure of $\mathcal{P}(D)$

$$\{-\} : D \to \mathcal{P}(D) \qquad \cup : \mathcal{P}(D) \to \mathcal{P}(D) \to \mathcal{P}(D)$$

$$\frac{\alpha_1, \alpha_2 : I \to \mathcal{P}(D) \qquad \mathrm{isDirected}(\alpha_1) \qquad \mathrm{isDirected}(\alpha_2)}{\bigsqcup \alpha_1 \cup \bigsqcup \alpha_2 = \bigsqcup(\lambda i.\, \alpha_1(i) \cup \alpha_2(i))} \qquad \frac{\alpha : I \to D \qquad \mathrm{isDirected}(\alpha)}{\{\bigsqcup_D \alpha\} = \bigsqcup(\{-\} \circ \alpha)}$$

Fig. 6. Operations of $\mathcal{P}(D)$ and their continuity

$$\frac{}{x \cup y \sqsubseteq y \cup x} \qquad \frac{}{(x \cup y) \cup z \sqsubseteq x \cup (y \cup z)} \qquad \frac{}{x \cup x \sqsubseteq x} \qquad \frac{}{x \sqsubseteq x \cup x}$$

Fig. 7. Inequalities involving operations of $\mathcal{P}(D)$

Definition 4.2 A **presignature** Σ : PreSig consists of a type of constructor names in $\Sigma_A : \mathcal{U}$ and a family of monomials, $\Sigma_M : \Sigma_A \to \mathrm{Monomial}$. We write $\Sigma_B(a)$ and $\Sigma_C(a)$ for the arity and the constant of $\Sigma_M(a)$ respectively.

As the Plotkin powertheory of D has two operations, we define the type of constructor names of its presignature to be a type with exactly two elements: $\widetilde{\{-\}}$ and $\widetilde{\cup}$. The family of monomials is constructed by sending both constructor names to their respective monomial we defined above.

Remark 4.3 The structure of a presignature looks similar to a container [1], which are used to define W-types [17], but for each $a : \Sigma_A$, there is an additional DCPO $\Sigma_C(a)$. For containers this is not needed, as one can introduce a new operation for each constant $c : \Sigma_C(a)$. However, this does not take into account the DCPO-structure of $\Sigma_C(a)$. We want the maps to be Scott continuous (as will be detailed in Section 5) and these should also be continuous with respect to the DCPO-structure of $\Sigma_C(a)$. For example, the monomial for $\widetilde{\{-\}}$ should represent a Scott continuous map $D \to \mathcal{P}(D)$, which is different from a family of constants $\{d\}$ for each $d : D$.

Inequalities are described by their left- and right-hand side, which are constructed using variables and the operations specified by a presignature. The notion of a *term* formalizes this concept.

Definition 4.4 Let Σ be a presignature and V a type of variables. The type $\mathrm{Term}_{\Sigma,V}$ of **terms** is

inductively generated by the following constructors.

$$\frac{x : V}{\mathrm{var}(x) : \mathrm{Term}_{\Sigma,V}} \qquad \frac{a : \Sigma_A \qquad f : \Sigma_B(a) \to \mathrm{Term}_{\Sigma,V} \qquad c : \Sigma_C(a)}{\mathrm{constr}_a(f, c) : \mathrm{Term}_{\Sigma,V}}$$

Each $\mathrm{var}(x)$ represents a variable, and $\mathrm{constr}_a(f, c)$ represents the application of the operation named a with arguments f, c.

For the Plotkin powertheory of D, terms are constructed using inclusion and formal union operations. Specifically, terms are constructed using the following derivation rules.

$$\frac{d : D}{\{d\} : \mathrm{Term}_{\Sigma,V}} \qquad \frac{t_1, t_2 : \mathrm{Term}_{\Sigma,V}}{t_1 \cup t_2 : \mathrm{Term}_{\Sigma,V}}$$

Definition 4.5 Let Σ be a presignature. A **formal inequality** for Σ consists of a type $V : \mathcal{U}$ and two terms $t_1, t_2 : \mathrm{Term}_{\Sigma,V}$. We write Ineq_Σ for the type of formal inequalities for Σ.

The type V represents the variables in an inequality. Typically, we write formal inequalities as $t_1 \mathrel{\widetilde{\sqsubseteq}}_V t_2$, and we omit V in case it is clear from the context.

In the Plotkin powertheory of D, we want the formal union to be commutative, associative and idempotent up to equality. We can describe equalities using antisymmetry and two formal equalities. However, it turns out that it is enough to only require the following formal inequalities.

$$x \cup y \mathrel{\widetilde{\sqsubseteq}} y \cup x \qquad (x \cup y) \cup z \mathrel{\widetilde{\sqsubseteq}} x \cup (y \cup z) \qquad x \cup x \mathrel{\widetilde{\sqsubseteq}} x \qquad x \mathrel{\widetilde{\sqsubseteq}} x \cup x$$

We now construct these formal inequalities. As all formal inequalities are constructed similarly, let us only look at the first one. It contains two free variables, so we take $V = \mathrm{Bool}$. We now define the terms representing the variables as $x = \mathrm{var}(\mathrm{false}), y = \mathrm{var}(\mathrm{true})$. We then combine these using $\cup\colon \mathrm{Term}_{\Sigma,V} \to \mathrm{Term}_{\Sigma,V} \to \mathrm{Term}_{\Sigma,V}$, to create both the left- and right-hand side of the equation.

Finally, we put everything together and we define the notion of a *signature*.

Definition 4.6 A **signature** $\Sigma : \mathrm{Sig}$ consists of a presignature $\Sigma_{\mathrm{pre}} : \mathrm{PreSig}$, a type of inequality names $\Sigma_E : \mathcal{U}$, and a family of formal inequalities $\Sigma_{\mathrm{ineq}} : \Sigma_E \to \mathrm{Ineq}_{\Sigma_{\mathrm{pre}}}$.

All in all, a signature $\Sigma : \mathrm{Sig}$ describes operations and inequalities. The operations are specifies by a type of constructor names, Σ_A, which specifies their names, and a monomial $\Sigma_M(a)$ for each $a : \Sigma_A$. The arity and constant of an operation $a : \Sigma_A$ is given by a type $\Sigma_B(a)$ and a DCPO $\Sigma_C(a)$, respectively. The inequalities are specified by a type Σ_E and a family Σ_{ineq} of formal inequalities over Σ_E.

We end this section by constructing a signature for the Plotkin powertheory of D. We already defined its presignature. We define Σ_E as a four element type, as we want four formal inequalities. The family of formal inequalities is defined using the formal inequalities we described above.

Remark 4.7 Note that our terms and inequalities do not involve suprema contrasting them to presentations (Definition 3.3). However, when we define the initial algebra in Section 6, we make use of the suprema in certain inequalities to require Scott continuity of the operations. Hence, an abstract basis (Example 3.1) is insufficient, whereas a presentation is more general. The reason behind the choice of our definition of terms and inequalities is its similarity to what one would write down in algebra.

5 Algebras

Next we define algebras for the notion of signature that we defined in the previous section. We do this in two steps. First we look at *prealgebras*, which consists of a DCPO together with the operations described by the signature.

Definition 5.1 Let Σ be a signature. A **prealgebra** for Σ consists of a DCPO X, called the underlying DCPO, and a Scott continuous map $\mathrm{op}_{X,a} : [\![\Sigma_M(a)]\!](X) \to X$ for each constructor name $a : \Sigma_A$. The type of prealgebras for Σ is denoted by PreAlg_Σ.

We can consider prealgebras as algebras for a functor. Each presignature Σ gives rise to a *polynomial* functor $[\![\Sigma]\!]$, which sends DCPOs X to $\sum_{a:\Sigma_A}(\Sigma_B(a) \to X) \times \Sigma_C(a)$. A prealgebra for Σ is the same as an algebra for the functor $[\![\Sigma]\!]$. The reason why we phrase it slightly differently in Definition 5.1, is because this way we do not need to mention type indexed coproducts of DCPOs.

Next we define morphisms between prealgebras.

Definition 5.2 Let X, Y be prealgebras for a signature Σ. An **algebra morphism** from X to Y is a Scott continuous map $f : X \to Y$ such that, for each $a : \Sigma_A$, we have $\mathrm{op}_{Y,a} \circ [\![\Sigma_M(a)]\!](f) = f \circ \mathrm{op}_{X,a}$.

Every signature Σ gives rise to a category whose objects are prealgebras and whose morphisms are algebra morphisms. Next we define *algebras*, which are prealgebras in which each formal inequality of the signature is satisfied. To do so, we first show how to interpret the terms of formal inequalities.

Definition 5.3 Let Σ be a presignature, V a type of variables, $t : \mathrm{Term}_{\Sigma,V}$ a term, X a prealgebra for Σ, and $\rho : V \to X$ a variable assignment. We define $[\![t]\!]_{X,\rho} : X$, the **interpretation** of t in X, by recursion.

$$[\![\mathrm{var}(x)]\!]_{X,\rho} = \rho(x) \qquad [\![\mathrm{constr}_a(f,c)]\!]_{X,\rho} = \mathrm{op}_{X,a}((\lambda b.\ [\![f(b)]\!]_{X,\rho}), c)$$

Variables get interpreted via ρ. The term $\mathrm{constr}_a(f,c)$, representing the application of the operation named a with arguments f and c, is interpreted by performing the operation named a in the prealgebra X. We often omit the subscript X, if it is clear from context which prealgebra we are working in.

Note that this interpretation is natural in the prealgebra X.

Proposition 5.4 *Let Σ be a presignature, V a type of variables, and $t : \mathrm{Term}_{\Sigma,V}$ a term. Furthermore, let X, Y be prealgebras for Σ and $f : X \to Y$ a prealgebra morphism. Then, for each variable assignment $\rho : V \to X$, we have that $[\![t]\!]_{Y,f\circ\rho} = f([\![t]\!]_{X,\rho})$.*

Proof. We prove this by structural induction on t. In the case that t is $\mathrm{var}(x)$, we conclude with reflexivity on $f(\rho(x))$. Otherwise, if t is $\mathrm{constr}_a(g,c)$, we have that

$$\begin{aligned}
[\![\mathrm{constr}_a(g,c)]\!]_{Y,f\circ\rho} &= \mathrm{op}_{Y,a}((\lambda b.\ [\![g(b)]\!]_{Y,f\circ\rho}), c) \\
&= \mathrm{op}_{Y,a}((\lambda b.\ f([\![g(b)]\!]_{X,\rho})), c) \\
&= \mathrm{op}_{Y,a}([\![\Sigma_M(a)]\!](f)((\lambda b.\ [\![g(b)]\!]_{X,\rho}), c)) \\
&= f(\mathrm{op}_{X,a}((\lambda b.\ [\![g(b)]\!]_{X,\rho}), c)) \\
&= f([\![\mathrm{constr}_a(g,c)]\!]_{X,\rho})
\end{aligned}$$

We can use the induction hypothesis on $g(b) : \mathrm{Term}_{\Sigma,V}$, since all $g(b)$ are structurally smaller than t. □

With this interpretation at hand, we define what it means for a formal inequality to be valid in a prealgebra. Intuitively, a formal inequality is valid, if the left-hand side is less than or equal to the right-hand side, for all possible values for the variables. This is exactly captured by the following definition.

Definition 5.5 Let $t_1 \mathrel{\widetilde{\sqsubseteq}}_V t_2$ be a formal inequality over Σ with variables $V : \mathcal{U}$. Furthermore, let X be a prealgebra for Σ. We say that this formal inequality is **valid** in X if, for each $\rho : V \to X$,

$$[\![t_1]\!]_\rho \sqsubseteq [\![t_2]\!]_\rho$$

Note that this is an inequality in the underlying DCPO of X. Whenever such a formal inequality is valid in X, we write it using the usual notation for validity: $X \vDash t_1 \mathrel{\widetilde{\sqsubseteq}}_V t_2$.

An algebra for a signature can now be defined as a prealgebra for that signature, where all the formal inequalities are valid.

Definition 5.6 Let Σ be a signature. An **algebra** for Σ is a prealgebra for Σ, such that for each $j : \Sigma_E$, we have that $X \vDash \Sigma_{\text{ineq}}(j)$. We define Alg_Σ to be the full subcategory of PreAlg_Σ, of those prealgebras where all $\Sigma_{\text{ineq}}(j)$ are valid.

6 Initial Algebra

We now construct the initial algebra for a signature as a *quotient inductive-inductive type*, QIIT. In Section 2 we have already motivated QIITs and in Section 3.2 we illustrated the use of QIITs as initial DCPO algebras using the example of the powerdomain in Figures 5, 6 and 7.

Our goal in this section is to adapt this construction to cover arbitrary signatures.

Recall that we subdivided the constructors of this QIIT into three groups. The first group, given in Figure 5, expresses that the type being constructed is a DCPO. More specifically, it says that we have a partial order and a supremum operation. We do not need to modify any of these constructors, because again we are constructing a DCPO. What we do need to change, however, are the constructors in Figures 6 and 7. In these two figures, the operations and inequalities of the signatures are described respectively. Specifically, we need to change these constructors so that instead we have the operations and inequalities described by the signature. Since QIITs give us initial algebras [3], it follows that the constructed DCPO algebra is indeed initial, which we prove in this section.

We start this section by constructing the initial algebra for a signature in Section 6.1. Then we show its initiality in Section 6.2 after showing the recursion and induction principles of the QIIT.

6.1 Construction of the Initial Algebra

To construct the initial algebra for a signature Σ, we simultaneously define a type Initial_Σ and an ordering relation $\sqsubseteq_\Sigma$ on Initial_Σ as a QIIT. We discuss their constructors in three steps. First, we show that we have constructors to define a DCPO structure on Initial_Σ. We then show that we have the constructors to lift this DCPO structure to a prealgebra structure for Σ. Finally, we lift this structure to an algebra structure for Σ.

Definition 6.1 We simultaneously define the type Initial_Σ, and its ordering relation $\sqsubseteq_\Sigma$ as a QIIT. Their constructors are given in Figures 8,9,10. The constructors of Initial_Σ are presented as terms with a type, while the constructors of $\sqsubseteq_\Sigma$ are given as inference rules. We quantify over directed families whose index types live in some universe $\mathcal{U}$, so both Initial_Σ and the truth values of $\sqsubseteq_\Sigma$ live in a larger universe than $\mathcal{U}$, and, if the arities and constants of the monomials in Σ live in $\mathcal{U}^+$, both Initial_Σ and the truth values of $\sqsubseteq_\Sigma$ also live in $\mathcal{U}^+$.

Compared to the powerdomain example from Section 3.2, the QIIT here is presented differently. In particular, the two constructors for the inclusion and union operations are replaced by app_a constructors for each constructor name $a : \Sigma_A$. Moreover, the continuity of each app_a is written down by expressing that app_a sends directed suprema to suprema in Initial_Σ. This notion of continuity [8, Definition 13], means we do not need to add path constructors to the QIIT to express continuity, like we did in Section 3.2.

Remark 6.2 If we unfold the definition of the interpretation of a monomial, we see that app has the following type: $\prod_{a:\Sigma_A}(\Sigma_B(a) \to \text{Initial}_\Sigma) \times \Sigma_C(a) \to \text{Initial}_\Sigma$. The introduction rule for W-types [17] is similar, but does not include $\Sigma_C(a)$. Like we noted in Remark 4.3, we cannot leave out $\Sigma_C(a)$, as this would result in a loss of structure. This is also apparent here, as otherwise we would not be able to state that app_a is Scott continuous with respect to the constant DCPO $\Sigma_C(a)$.

The constructors for the DCPO structure on Initial_Σ are given in Figure 8. The rules Leq-Refl, Leq-Trans and Leq-Prop make sure that $\sqsubseteq_\Sigma$ is reflexive, transitive and proposition valued. Further-

$$\textsc{Leq-Refl}\ \frac{}{x \sqsubseteq_\Sigma x} \qquad \textsc{Leq-Trans}\ \frac{x \sqsubseteq_\Sigma y \qquad y \sqsubseteq_\Sigma z}{x \sqsubseteq_\Sigma z} \qquad \textsc{Leq-Prop}\ \frac{}{\mathrm{isProp}(x \sqsubseteq_\Sigma y)}$$

$$\textsc{Leq-Antisym}\ \frac{x \sqsubseteq_\Sigma y \qquad y \sqsubseteq_\Sigma}{x = y} \qquad \textsc{Initial-Set}\ \frac{}{\mathrm{isSet}(\mathrm{Initial}_\Sigma)}$$

$$\bigsqcup_\Sigma : \prod_{\alpha : I \to \mathrm{Initial}_\Sigma} \mathrm{isDirected}(\alpha) \to \mathrm{Initial}_\Sigma$$

$$\bigsqcup_\Sigma\textsc{-Is-Sup-1}\ \frac{\alpha : I \to \mathrm{Initial}_\Sigma \qquad \delta : \mathrm{isDirected}(\alpha)}{\prod_{i:I} \alpha(i) \sqsubseteq_\Sigma \bigsqcup_\Sigma(\alpha, \delta)} \qquad \bigsqcup_\Sigma\textsc{-Is-Sup-2}\ \frac{\alpha : I \to \mathrm{Initial}_\Sigma \qquad \delta : \mathrm{isDirected}(\alpha)}{\prod_{v:\mathrm{Initial}_\Sigma} \mathrm{isUpperbound}(v, \alpha) \to \bigsqcup_\Sigma(\alpha, \delta) \sqsubseteq_\Sigma v}$$

Fig. 8. Constructors for the DCPO structure

$$\mathrm{app}_a : [\![\Sigma_M(a)]\!](\mathrm{Initial}_\Sigma) \to \mathrm{Initial}_\Sigma$$

$$\textsc{app-Is-Cont-1}\ \frac{a : \Sigma_A \qquad \alpha : I \to [\![\Sigma_M(a)]\!](\mathrm{Initial}_\Sigma) \qquad \delta : \mathrm{isDirected}(\alpha)}{\prod_{i:I} \mathrm{app}_a(\alpha(i)) \sqsubseteq_\Sigma \mathrm{app}_a(\bigsqcup_{[\![\Sigma_M(a)]\!](\mathrm{Initial}_\Sigma)} \alpha)}$$

$$\textsc{app-Is-Cont-2}\ \frac{a : \Sigma_A \qquad \alpha : I \to [\![\Sigma_M(a)]\!](\mathrm{Initial}_\Sigma) \qquad \delta : \mathrm{isDirected}(\alpha)}{\prod_{v:\mathrm{Initial}_\Sigma} \mathrm{isUpperbound}(v, \mathrm{app}_a \circ \alpha) \to \mathrm{app}_a(\bigsqcup_{[\![\Sigma_M(a)]\!](\mathrm{Initial}_\Sigma)} \alpha) \sqsubseteq_\Sigma v}$$

Fig. 9. Constructors for the prealgebra structure for Σ

$$\textsc{Ineq-Valid}\ \frac{j : \Sigma_E \qquad \Sigma_{\mathrm{ineq}}(j) = t_1 \mathrel{\widetilde{\sqsubseteq}_V} t_2 \qquad \rho : V \to \mathrm{Initial}_\Sigma}{[\![t_1]\!]_\rho \sqsubseteq_\Sigma [\![t_2]\!]_\rho}$$

Fig. 10. Constructors for the algebra structure for Σ

more, we guarantee that $\sqsubseteq_\Sigma$ is antisymmetric and $\mathrm{Initial}_\Sigma$ is a set, [8] by adding suitable path constructors. Finally, the $\bigsqcup_\Sigma$ constructor is supposed to give the supremum for directed families. This is guaranteed by $\bigsqcup_\Sigma$-Is-Sup-1 and $\bigsqcup_\Sigma$-Is-Sup-2. Note that we often write $\bigsqcup_\Sigma \alpha$, in case we do not care about the specific proof of directedness.

Lemma 6.3 *The type* $\mathrm{Initial}_\Sigma$ *has a DCPO structure.*

Next up, the constructors for the prealgebra structure are given in Figure 9. The app_a constructor

[8] We could do without the set truncation, as the underlying type of a partial order is always a set [10, Definition 4.1]. See also https://github.com/martinescardo/TypeTopology/blob/02add316f8a78bc79e5687e4249d9f0174dae1d2/source/UF/Hedberg.lagda#L90.

introduces, for each $a : \Sigma_A$, an operation corresponding to the monomial $\Sigma_M(a)$. Note that, by Lemma 6.3, we can actually pass Initial_Σ as an argument to the DCPO endo-functor $[\![\Sigma_M(a)]\!]$. To guarantee that each operation app_a is continuous, we have the rules app-Is-Cont-1 and app-Is-Cont-2. Combining these rules we get

$$\frac{a : \Sigma_A \qquad \alpha : I \to [\![\Sigma_M(a)]\!](\text{Initial}_\Sigma) \qquad \delta : \text{isDirected}(\alpha)}{\text{isLeastUpperbound}(\text{app}_a(\bigsqcup_{[\![\Sigma_M(a)]\!](\text{Initial}_\Sigma)} \alpha), \text{app}_a \circ \alpha)}$$

This is exactly what it means for app_a to be continuous.

Lemma 6.4 *The type* Initial_Σ *has a prealgebra structure for the signature* Σ*.*

Finally, we have one single constructor for the algebra structure for the signature Σ given in Figure 9. The rule Ineq-Valid guarantees that all the formal inequalities of the signature are valid. Note that, by Lemma 6.4, we have an algebra structure for Σ_{pre}, so we can indeed interpret the left- and right-hand side of the formal inequality in Initial_Σ.

Theorem 6.5 *The type* Initial_Σ *has an algebra structure for the signature* Σ*.*

6.2 Initiality

To show that Initial_Σ is initial in Alg_Σ, we need to construct a unique morphism $\text{Initial}_\Sigma \to X$ for arbitrary X. The existence of such a morphism comes from the recursion principle for Initial_Σ.

Theorem 6.6 *Let* Σ *be a signature and* X *an algebra for* Σ*. Then there exists a morphism* $\text{rec}_{\Sigma,X} : \text{Initial}_\Sigma \to X$*, with the following computation rules*[9]

$$\text{rec}_{\Sigma,X}(\text{app}_a(x)) \doteq \text{op}_{X,a}([\![\Sigma_M(a)]\!](\text{rec}_{\Sigma,X})(x))$$
$$\text{rec}_{\Sigma,X}(\textstyle\bigsqcup_\Sigma \alpha) \doteq \bigsqcup_X(\text{rec}_{\Sigma,X} \circ \alpha)$$

Proof. The two remaining constructors of Initial_Σ, Leq-Antisym and Initial-Set, get send to the proofs that X has an antisymmetric relation and is a set, respectively. The function $\text{rec}_{\Sigma,X}$ is well-defined, as it is monotone.[10] It is an algebra morphism, as by definition, it is continuous and commutes with all the operations. □

To prove the uniqueness of such a morphism, we need an induction principle for Initial_Σ. For this, we could use *displayed algebras* [22,14] in its general form, but rather, in this paper, we only consider our specific use case. First, let us define how a monomial acts on a family of types.

Definition 6.7 Let M be a monomial with arity B and constant DCPO C, X a DCPO, and $Y : X \to \mathcal{U}$ a family of types. The family Y lifts to a type family $\overline{M}(Y) : [\![M]\!](X) \to \mathcal{U}$, which is defined as

$$\overline{M}(Y)(f, c) = \left(\prod_{b:B} Y(f(b))\right) \times C$$

The action defined above, allows us to state the induction hypotheses for Initial_Σ. If we restrict our view to a predicate $Y : X \to \mathcal{U}$ and let $x : [\![M]\!](X)$, then $\overline{M}(Y)(x)$ expresses the fact that Y holds for

[9] Here, $\doteq$ means definitional equality.

[10] In Agda, we annotate the recursion principle with a `TERMINATING` pragma, as the termination checker believes there are non-structural recursive calls in the monotonicity proof. We think that the termination checker is too restrictive here, and that the recursion principle should be accepted. The problem seems to be that Agda does not unfold definitions enough to recognize that the recursive calls are correct. We explain this in more detail in the formalization (`src/SignatureAlgebra/Initial/Elimination.agda`).

all the subterms of type X in x. This is exactly what we need as the induction hypothesis in the case of $\mathrm{app}_a(x) : \mathrm{Initial}_\Sigma$.

Theorem 6.8 *Let $Y : \mathrm{Initial}_\Sigma \to \mathcal{U}$ be a family of types, such that*

- *each $Y(x)$ is a proposition;*
- *Y is* supremum preserving*: if $\alpha : I \to \mathrm{Initial}_\Sigma$ is a directed family, such that $Y(\alpha(i))$ holds for every $i : I$, then $Y(\bigsqcup_\Sigma(\alpha))$;*
- *Y is* app preserving*: if a is a constructor name, $x : [\![\Sigma_M(a)]\!](\mathrm{Initial}_\Sigma)$ and $\overline{\Sigma_M(a)}(Y)(x)$, then $Y(\mathrm{app}_a(x))$ holds.*

In that case $Y(x)$ holds for every $x : \mathrm{Initial}_\Sigma$, that is, we have a function $\mathrm{ind}_{\Sigma,Y} : \prod_{x:\mathrm{Initial}_\Sigma} Y(x)$.

Proof. The function $\mathrm{ind}_{\Sigma,Y}$ is defined by pattern matching. For the $\bigsqcup_\Sigma$ and app_a constructors, we use that fact that Y is supremum and app preserving. On the path constructors of $\mathrm{Initial}_\Sigma$, $\mathrm{ind}_{\Sigma,Y}$ is defined using the fact that Y is a proposition valued. □

With these elimination principles at hand, we show the initiality of $\mathrm{Initial}_\Sigma$.

Theorem 6.9 *Let Σ be a signature. The algebra $\mathrm{Initial}_\Sigma$ is initial in the category Alg_Σ.*

Proof. Let X be an arbitrary algebra for Σ. The unique algebra morphism $! : \mathrm{Initial}_\Sigma \to X$ is given by the recursion principle.

Now let $f : \mathrm{Initial}_\Sigma \to X$ be an arbitrary map of algebras. To show that $!$ and f are equal, it is enough to show that their underlying functions are equal. After applying function extensionality, it thus suffices to show the following for each $x : \mathrm{Initial}_\Sigma$

$$!(x) = f(x) \tag{1}$$

As the underlying type X is a set, we know that (1) is a proposition. We can thus use the induction principle of $\mathrm{Initial}_\Sigma$. First we prove that (1) is true for the supremum of a directed family. Let $\alpha : I \to \mathrm{Initial}_\Sigma$ be a directed family. By the computation rules of $!$ and the continuity of f, we need to show that the suprema in X of $! \circ \alpha$ and $f \circ \alpha$ are equal. This holds, because the families $! \circ \alpha$ and $f \circ \alpha$ are equal by the induction hypothesis.

Finally, we need to show that (1) is true for any $\mathrm{app}_a(x)$. By the computation rules of $!$ and the fact that f commutes with all operations, we have to show that

$$\mathrm{op}_{X,a}([\![\Sigma_M(a)]\!](!)(x)) = \mathrm{op}_{X,a}([\![\Sigma_M(a)]\!](f)(x))$$

This follows from the induction hypothesis, as (1) holds for all the subterms of x. □

7 Examples

Finally, we show various examples of signatures and their initial algebras in this section using Theorem 6.9. We start with several categorical constructions of DCPOs (coalesced sums, smash products, and coequalizers), and then we show examples of free DCPOs (partiality and powerdomains).

7.1 Coalesced Sum

Recall that a pointed DCPO is a DCPO with a least element. The categorical coproduct of pointed DCPOs is defined as their coalesced sum. Classically, one defines this by taking the union of the two pointed DCPOs, removing both their bottom elements, and adding a new bottom element. We define this DCPO as the initial algebra for a signature.

Example 7.1 Let D, E be pointed DCPOs. We define Σ_{D+E}. We add two operations, with constructor names $\tilde{\iota}_D, \tilde{\iota}_E$ and corresponding monomials $(\bot \to X) \times D$ and $(\bot \to X) \times E$. This allows us to write down terms of the form $\iota_D(d), \iota_E(e)$ for $d : D, e : E$. Finally, we add the following two inequalities.

$$\iota_D(\bot_D) \mathrel{\widetilde{\sqsubseteq}} \iota_E(\bot_E) \qquad \iota_E(\bot_E) \mathrel{\widetilde{\sqsubseteq}} \iota_D(\bot_D)$$

An algebra for Σ_{D+E} consists of a DCPO X, together with inclusion operations $\iota_D : D \to X$ and $\iota_E : E \to X$, such that $\iota_D(\bot_D) = \iota_E(\bot_E)$.

Proposition 7.2 *Let D, E be pointed DCPOs. Their coproduct is given by the initial algebra for Σ_{D+E}.*

Proof. By taking $\iota_D(\bot_D) = \iota_E(\bot_E)$ as the bottom element and using Theorem 6.8. □

7.2 Coequalizer

Let D, E be DCPOs and $f, g : D \to E$ continuous maps. We define the coequalizer by first defining a signature whose algebras correspond to coequalizer cocones, and then considering the initial algebra for this signature.

Example 7.3 Let D, E be DCPOs and $f, g : D \to E$. We define $\Sigma_{f \rightrightarrows g}$. It has a single operation which includes E, i.e. it is represented by the monomial $(\bot \to X) \times E$. This allows us to write terms of the form $\iota(e)$ for $e : E$. For each $d : D$, we add the following two inequalities.

$$\iota(f(d)) \mathrel{\widetilde{\sqsubseteq}} \iota(g(d)) \qquad \iota(g(d)) \mathrel{\widetilde{\sqsubseteq}} \iota(f(d))$$

An algebra for $\Sigma_{f \rightrightarrows g}$ is a DCPO X, together with a map $\iota : E \to X$, such that $\iota(f(d)) = \iota(g(d))$ for each $d : D$.

Proposition 7.4 *Let D, E be DCPOs and $f, g : D \to E$. Algebras for $\Sigma_{f \rightrightarrows g}$ correspond to coequalizer cocones of f and g. Furthermore, the initial algebra for $\Sigma_{f \rightrightarrows g}$ is the coequalizer of f and g.*

7.3 Smash Product

Let D, E be pointed DCPOs. Their smash product is defined to be the product $D \times E$, where we identify $(\bot_D, e) = (d, \bot_E)$ for every $d : D, e : E$. Concretely, we define the smash product as the initial algebra for a suitably chosen signature.

Example 7.5 Let D, E be pointed DCPOs. We define $\Sigma_{D \wedge E}$. It has a single operation which includes $D \times E$, i.e. it is represented by the monomial $(\bot \to X) \times (D \times E)$. This allows us to write terms of the form (d, e) for $d : D, e : E$. For each $d : D, e : E$, we add the following two inequalities.

$$(d, \bot_E) \mathrel{\widetilde{\sqsubseteq}} (\bot_D, e) \qquad (\bot_D, e) \mathrel{\widetilde{\sqsubseteq}} (d, \bot_E)$$

An algebra for $\Sigma_{D \wedge E}$ consists of a DCPO X, together with an inclusion $\iota : D \times E \to X$, such that $\iota(\bot_D, e) = \iota(d, \bot_E)$ for every $d : D, e : E$.

Definition 7.6 Let D, E be pointed DCPOs. Their **smash product** is the initial algebra for $\Sigma_{D \wedge E}$.

7.4 Free Algebra for a Signature

The next two examples that we consider are special cases of a more general construction, namely the free algebra, which we discuss here. More specifically, suppose that we have some signature Σ and some DCPO D. Our goal is to define a signature $\Sigma + D$ such that the initial algebra for $\Sigma + D$ gives the free Σ-algebra for D. We use this construction to construct a left adjoint $F : \mathrm{DCPO} \to \mathrm{Alg}_\Sigma$ for the forgetful functor $U : \mathrm{Alg}_\Sigma \to \mathrm{DCPO}$. The main idea is that $\Sigma + D$ is obtained by adding an operation $D \to UFD$ to Σ.

Definition 7.7 Let Σ be a signature and D a DCPO. We define a signature $\Sigma + D$, by extending the constructor names of Σ with a new element $\tilde{\iota}$, and assign $(\bot \to X) \times D$ as the monomial for $\tilde{\iota}$. We do not add any new inequalities, but note that we have to lift the original inequalities of Σ as we changed the type of constructor names. We leave these technical details to the formalization.

If $X : \mathrm{Alg}_\Sigma$ is an algebra and $f : D \to UX$, we construct an algebra for the signature $\Sigma + D$, by using f as the interpretation for the inclusion constructor name $\tilde{\iota}$. We write $X + f : \mathrm{Alg}_{\Sigma+D}$ for this algebra. Conversely, if $X : \mathrm{Alg}_{\Sigma+D}$ is an algebra for $\Sigma + D$, we define the algebra $X^- : \mathrm{Alg}_\Sigma$, by forgetting the interpretation of $\tilde{\iota}$. Now we define the free Σ-algebra for D to be $(\mathrm{Initial}_{\Sigma+D})^-$.

Theorem 7.8 *The operation $D \mapsto (\mathrm{Initial}_{\Sigma+D})^-$ lifts to a functor $F : \mathrm{DCPO} \to \mathrm{Alg}_\Sigma$, which is left adjoint to the forgetful functor $U : \mathrm{Alg}_\Sigma \to \mathrm{DCPO}$.*

Proof. We construct this adjunction using universal arrows. The unit of the adjunction, $\eta_D : D \to UFD$, is given by the inclusion operation $\mathrm{app}_{\tilde{\iota}}$. The universal arrows are constructed in the following way. Let $X : \mathrm{Alg}_\Sigma$ be an algebra and $f : D \to UX$. By the initiality of $\mathrm{Initial}_{\Sigma+D}$, we have a unique map $\mathrm{Initial}_{\Sigma+D} \to X + f$. By forgetting about the fact that this map commutes with the inclusion operation, we get a map $FD \to X$, whose uniqueness follows from the initiality of $\mathrm{Initial}_{\Sigma+D}$. □

7.5 Pointed DCPO

We instantiate the free algebra construction of Section 7.4 to two specific examples. The first example that we consider, is given by pointed DCPOs. The type of pointed DCPOs can be given as algebras for a suitably chosen signature.

Example 7.9 We define $\Sigma_{\mathrm{DCPO}_\bot}$, the signature for pointed DCPOs. We add one operation with constructor name $\tilde{\bot}$ and monomial $(\bot \to X) \times 1$. This allows us to write down the term $\bot_t$, which represents the least element. Finally, we add the formal inequality $\bot_t \mathrel{\widetilde{\sqsubseteq}} x$ to our signature.

An algebra for $\Sigma_{\mathrm{DCPO}_\bot}$ thus consists of a DCPO D together with a least element. A map between algebras D, E is a continuous map $D \to E$ which preserves the bottom element.

Proposition 7.10 *Algebras for $\Sigma_{\mathrm{DCPO}_\bot}$ correspond to pointed DCPOs.*

By using Theorem 7.8 for the signature $\Sigma_{\mathrm{DCPO}_\bot}$, we obtain an adjunction between the category of DCPOs and pointed DCPOs, and this adjunction gives rise to a monad. Note that this construction is similar to how Altenkirch, Danielsson, and Kraus constructed the partiality monad [4], except that we use DCPOs instead of ω-CPOs.

7.6 Power Algebras

Our final example is the Plotkin powerdomain for which we described a suitable signature in Section 4.

Example 7.11 We define Σ_{Power}, the signature for the Plotkin powertheory. We add one operation with constructor name $\widetilde{\cup}$ and monomial $(\mathrm{Bool} \to X) \times 1$. Given terms t_1, t_2, we write $t_1 \cup t_2$ for the term representing the formal union of t_1, t_2. Finally, we add the following formal inequalities, to guarantee that formal union is commutative, associative and idempotent.

$$x \cup y \mathrel{\widetilde{\sqsubseteq}} y \cup x \qquad (x \cup y) \cup z \mathrel{\widetilde{\sqsubseteq}} x \cup (y \cup z) \qquad x \cup x \mathrel{\widetilde{\sqsubseteq}} x \qquad x \mathrel{\widetilde{\sqsubseteq}} x \cup x$$

An algebra for Σ_{Power} consists of a DCPO D, and a continuous map $\cup : D \to D \to D$ which is commutative, associative and idempotent. A map between algebras D, E is a continuous map $D \to E$ which commutes with the formal union operations.

By using Theorem 7.8 for the signature Σ_{Power}, we get that every DCPO D gives rise to a free algebra for Σ_{Power}, which is the Plotkin powerdomain. This free algebra is constructed as the QIIT with the constructors given in Figure 6, 7, and 5.

8 Related Work

In this section, we briefly recall the related work and how it compares to the results in this paper. Our development is inspired by the work of Altenkirch, Danielsson, and Kraus [4] that discusses the partiality monad as a domain. We extend this by developing a general notion of signature for DCPOs to capture a large variety of domain constructions. A difference is that Altenkirch, Danielsson, and Kraus use ω-CPOs, whereas we use DCPOs. These two notions are equivalent if we assume the axiom of choice (and limit to countable index sets), but constructively the notion of DCPO is stronger. This difference complicates the specification of the QIIT in Section 6, because we have to quantify over all directed sets (and not just chains). Note that one can modify our work to obtain free ω-CPO algebras.

There are various other constructions of the partiality monad. We have already mentioned the work by Escardó and Knapp [11] and by De Jong and Escardó [8], who construct the lift of types and of DCPOs without using any quotient type. More specifically, the lift of a type A is defined to be the type $\sum_{P:\Omega} P \to A$ of partial elements of A, where Ω is the type of all propositions. Their construction gives rise to a DCPO, which is in fact the free pointed DCPO. Note that each of these constructions is isomorphic to each other, because they give rise to a left adjoint of the forgetful functor from the category of DCPOs to the category of sets. Chapman, Uustalu, and Veltri construct the free ω-CPO using a quotient inductive type, namely the free countably-complete semilattice [6]. Their definition is similar to the one used in the work by Escardó and Knapp and by De Jong and Escardó, except that Ω is replaced by the initial σ-frame. They use quotient inductive types to construct the initial σ-frame. Note that one can also construct the initial σ-frame using only function extensionality, propositional extensionality, and propositional truncations as shown by Escardó [11] .

Finally, there are various other constructions of free DCPO algebras in classical foundations. One way to construct these algebras is by using the free DCPO for a presentation [13]. This construction is impredicative because it uses the power set, and if one were to use this in predicative foundations, then it raises the universe level. Bidlingmaier, Faissole, and Spitters discuss three ways to justify the existence of free ω-CPOs for presentations constructively [5, Section 3], and they assume the existence of free ω-CPOs for presentations. One can justify their axiom by either assuming impredicativity, countable choice, or QIITs, and in this paper, we use the last way to justify the existence of free complete partial orders. Abramsky and Jung [2, Chapter 6] discuss equational theories of domains. Their notion of signature only includes operations with an arity, and they do not have constants. For this reason, one cannot construct the coequalizers, the coalesced sum, and the smash product as an initial algebra of their signatures.

9 Conclusions and Future Work

We have presented a general framework to describe and construct algebraic effects in domain theory. We describe the algebraic effects using a signature: a set of operations accompanied by an inequational theory these operations should obey. An algebra for such a signature is a DCPO with Scott continuous maps for each operation, such that the inequational theory is satisfied. We construct the initial algebra as a quotient inductive-inductive type (QIIT), extending [4] (where the partiality monad is constructed as a QIIT). Different from [13], this construction does not use power sets, so it is predicative. The initial algebra is used for interpreting algebraic effects in denotational semantics and we show that our framework captures a variety of well-known examples, like coalesced sums, smash products, coequalizers, partiality and power domains. Our work has all been formalized in Cubical Agda.

Potential future work is to also consider handlers for algebraic effects. We focus on the effects themselves, but their handlers are just as important, as a handler describes how a result of a particular effect should be handled [20]. To give the full denotational semantics of a language with algebraic effects, we also need to interpret the handlers in domain theory.

[11] https://github.com/martinescardo/TypeTopology/blob/master/source/NotionsOfDecidability/QuasiDecidable.lagda

References

[1] Abbott, M. G., T. Altenkirch and N. Ghani, *Containers: Constructing strictly positive types*, Theoretical Computer Science **342**, pages 3–27 (2005).
https://doi.org/10.1016/J.TCS.2005.06.002

[2] Abramsky, S. and A. Jung, *Domain theory*, in: S. Abramsky, D. M. Gabbay and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168, Clarendon Press (1994).

[3] Altenkirch, T., P. Capriotti, G. Dijkstra, N. Kraus and F. N. Forsberg, *Quotient inductive-inductive types*, in: C. Baier and U. D. Lago, editors, *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10803 of *Lecture Notes in Computer Science*, pages 293–310, Springer (2018).
https://doi.org/10.1007/978-3-319-89366-2_16

[4] Altenkirch, T., N. A. Danielsson and N. Kraus, *Partiality, revisited - the partiality monad as a quotient inductive-inductive type*, in: J. Esparza and A. S. Murawski, editors, *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 534–549 (2017).
https://doi.org/10.1007/978-3-662-54458-7_31

[5] Bidlingmaier, M. E., F. Faissole and B. Spitters, *Synthetic topology in homotopy type theory for probabilistic programming*, Mathematical Structures in Computer Science **31**, pages 1301–1329 (2021), ISSN 0960-1295,1469-8072.
https://doi.org/10.1017/s0960129521000165

[6] Chapman, J., T. Uustalu and N. Veltri, *Quotienting the delay monad by weak bisimilarity*, in: M. Leucker, C. Rueda and F. D. Valencia, editors, *Theoretical Aspects of Computing - ICTAC 2015 - 12th International Colloquium Cali, Colombia, October 29-31, 2015, Proceedings*, volume 9399 of *Lecture Notes in Computer Science*, pages 110–125, Springer (2015).
https://doi.org/10.1007/978-3-319-25150-9_8

[7] de Jong, T., *Apartness, sharp elements, and the Scott topology of domains*, Mathematical Structures in Computer Science **33**, pages 573–604 (2023).
https://doi.org/10.1017/S0960129523000282

[8] de Jong, T. and M. H. Escardó, *Domain Theory in Constructive and Predicative Univalent Foundations*, in: C. Baier and J. Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25-28, 2021, Ljubljana, Slovenia (Virtual Conference)*, volume 183 of *LIPIcs*, pages 28:1–28:18, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021).
https://doi.org/10.4230/LIPICS.CSL.2021.28

[9] de Jong, T. and M. H. Escardó, *Predicative aspects of order theory in univalent foundations*, in: N. Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference)*, volume 195 of *LIPIcs*, pages 8:1–8:18, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021).
https://doi.org/10.4230/LIPICS.FSCD.2021.8

[10] de Jong, T. and M. H. Escardó, *On small types in univalent foundations*, Log. Methods Comput. Sci. **19** (2023).
https://doi.org/10.46298/LMCS-19(2:8)2023

[11] Escardó, M. H. and C. M. Knapp, *Partial elements and recursion via dominances in univalent type theory*, in: V. Goranko and M. Dam, editors, *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden*, volume 82 of *LIPIcs*, pages 21:1–21:16, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017).
https://doi.org/10.4230/LIPICS.CSL.2017.21

[12] Frumin, D., H. Geuvers, L. Gondelman and N. v. d. Weide, *Finite sets in homotopy type theory*, in: *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2018, page 201–214, Association for Computing Machinery, New York, NY, USA (2018), ISBN 9781450355865.
https://doi.org/10.1145/3167085

[13] Jung, A., M. A. Moshier and S. Vickers, *Presenting Dcpos and Dcpo algebras*, Electronic Notes in Theoretical Computer Science **218**, pages 209–229 (2008), ISSN 1571-0661. Proceedings of the 24th Conference on the Mathematical Foundations of Programming Semantics (MFPS XXIV).
https://doi.org/https://doi.org/10.1016/j.entcs.2008.10.013

[14] Kaposi, A. and A. Kovács, *A syntax for higher inductive-inductive types*, in: H. Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*, pages 20:1–20:18, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018).
https://doi.org/10.4230/LIPICS.FSCD.2018.20

[15] Kaposi, A. and A. Kovács, *Signatures and induction principles for higher inductive-inductive types*, Logical Methods in Computer Science **16** (2020).
https://doi.org/10.23638/LMCS-16(1:10)2020

[16] Markowsky, G., *Chain-complete posets and directed sets with applications*, Algebra universalis **6**, pages 53–68 (1976).
https://api.semanticscholar.org/CorpusID:16718857

[17] Martin-Löf, P., *Intuitionistic type theory*, volume 1 of *Studies in proof theory*, Bibliopolis (1984), ISBN 978-88-7088-228-5.

[18] Nordvall Forsberg, F., *Inductive-inductive definitions*, Ph.D. thesis, Swansea University (2013).

[19] Plotkin, G. D., *A powerdomain construction*, SIAM J. Comput. **5**, pages 452–487 (1976).
https://doi.org/10.1137/0205035

[20] Plotkin, G. D. and M. Pretnar, *Handling Algebraic Effects*, Logical Methods in Computer Science **Volume 9, Issue 4** (2013).
https://doi.org/10.2168/LMCS-9(4:23)2013

[21] Scott, D., *Outline of a mathematical theory of computation* page 30 (1970).

[22] Sojakova, K., *Higher inductive types as homotopy-initial algebras*, in: S. K. Rajamani and D. Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 31–42, ACM (2015).
https://doi.org/10.1145/2676726.2676983

[23] Univalent Foundations Program, T., *Homotopy Type Theory: Univalent Foundations of Mathematics*, Institute for Advanced Study (2013).
https://homotopytypetheory.org/book

[24] van Collem, S., *Initial algebras of domains via quotient inductive-inductive types* (2025).
https://doi.org/10.5281/zenodo.15176521

[25] van der Weide, N. and D. Frumin, *The interval domain in homotopy type theory*, in: V. Capretta, R. Krebbers and F. Wiedijk, editors, *Logics and Type Systems in Theory and Practice: Essays Dedicated to Herman Geuvers on The Occasion of His 60th Birthday*, volume 14560 of *Lecture Notes in Computer Science*, pages 241–256, Springer Nature Switzerland, Cham (2024), ISBN 978-3-031-61716-4.
https://doi.org/10.1007/978-3-031-61716-4_16

[26] Vezzosi, A., A. Mörtberg and A. Abel, *Cubical Agda: A dependently typed programming language with univalence and higher inductive types*, J. Funct. Program. **31**, page e8 (2021).
https://doi.org/10.1017/S0956796821000034