Formalized, Effective Domain Theory in Coq

Robert Dockins

Portland State University
rdockins@pdx.edu

Abstract. I present highlights from a formalized development of domain theory in the theorem prover Coq. This is the first development of domain theory that is *effective*, *formalized* and that supports all the usual constructions on domains. In particular, I develop constructive models of both the unpointed profinite and the pointed profinite domains. Standard constructions (e.g., products, sums, the function space, and powerdomains) are all developed. In addition, I build the machinery necessary to compute solutions to recursive domain equations.

1 Introduction and Related Work

The term "domain theory" refers to a class of mathematical techniques that are used to develop computational models suitable for reasoning about the semantics of general purpose programming languages. Proofs done in domain theory often have an elegant, compositional nature, and the much of the modern thinking about programming languages (especially functional languages) can be traced to domain-theoretic roots. Domain theory has a long lineage, starting from the classic work of Dana Scott on continuous lattices [22]. Domain theory was intensively studied in the 1970's and 80's, producing far more research than I have space here to review; see Abramsky and Jung's account for a good overview and further sources [2].

Unfortunately, using domain theory for language semantics requires specialized mathematical knowledge. Many proofs that are purely "domain theoretic" in nature must be done before domain theory can be used on a language problem of interest. Furthermore, the wealth of academic literature on domain theory is actually a liability in some ways: the sheer volume of knowledge on the subject and the myriad minor variations can be overwhelming to the novice.

It would thus be desirable to package up the difficult mathematics underlying domain theory into a library of facts and constructions that can be pulled off the shelf and used by those interested in programming languages, but not in domain theory *per se*. Such a library should choose, from among the many varieties and presentations in the literature, a selection of domain theoretic techniques that are suitable for practical use. The development of domain theory described in this paper is my attempt to develop such a library for the Coq proof assistant.

My own effort is far from the first attempt to make domain theory more accessible. One of the earliest mechanical proof systems, Milner's Logic for Com-

putable Functions (LCF) [15, 16], is a syntactic presentation of a logic for reasoning about functions defined via least-fixed-point semantics in Scott domains.[1]

In terms of the mathematics formalized, the most closely related work is probably the formalization of bifinite domains in Isabelle/HOL by Brian Huffman [13], building on the HOLCF system by Müller *et al.* [17]. One major difference between this work and that of Huffman, besides the choice of theorem prover, is that my development of domain theory is *effective*. There are two different senses in which this statement is true. First, the objects and constructions in the theory are effective; I formalize categories of effective algebraic domains (whose bases are countable and have decidable ordering), similar to those considered by Egli and Constable [9]. Second, the *metalogic* in which these constructions and proofs are carried out is the purely constructive metalogic of Coq with no additional axioms. As a result, I can forgo the explicit development of recursive function theory, relying instead on the innate "effectiveness" of functions definable in type theory. In contrast, HOL is a classical logic with strong choice principles. There are also some differences in the proof strategy — Huffman uses definitions of bifinite domains based on finitary deflations, whereas I use properties of Plotkin orders. I found it difficult to obtain appropriately constructive definitions of deflations, whereas Plotkin orders proved more amenable to constructive treatment. In addition, the uniform presentation of pointed and unpointed domains I used (discussed below) is novel.

In the Coq community, the most closely-related work appears to be the development of constructive complete partial orders (CPOs) by Benton *et al.* [7], which is based on earlier work by Paulin-Mohring [18]. The CPOs built by Benton *et al.* are constructive; however, they lack bases and thus are not actually *domains* according to the usage of the term used by some authors [2]. This line of work develops a constructive version of the lift construction by using coinductive ε-streams. In contrast, the main tool I use for achieving effective constructions is the enumerable set. Proofs using enumerable sets are, in my opinion, both easier to understand and easier to develop than those based on coinductive structures (`cofix`, with its syntactic guardedness condition, is especially troublesome).

My development of domain theory is able to handle some constructions that Benton *et al.* cannot handle; for example, they report difficulty defining the smash product, which presents no problem for me. Powerdomains can also be constructed in my system, whereas they cannot be on basic CPOs.

The main contribution of this paper is a formalization of the pointed profinite and unpointed profinite domains, together with common operations on them like products, sums, function space, and powerdomains. I also build the machinery necessary to solve recursive domain equations. The constructions and proofs largely follow the development of the profinite domains given by Carl Gunter in his thesis [12]; however, all the constructions are modified to be "effectively given," similar to [9].

The primary novelty of my approach is related to proof-engineering advances that both clarify and simplify portions of the proof. The main instance of this is

[1] Milner [16] credits the design of LCF to an unpublished note of Dana Scott.

a unification of the constructions for the pointed profinite domains (those with a least element) and the unpointed profinite domains (those *not necessarily* containing a least element). These two categories of domains are very similar; in my proof development, this similarity is aggressively exploited by constructing both categories of domains at the same time as two instantiations of a single parametrized construction. Common operations on domains (sums, products, function space, powerdomains) are also constructed uniformly for both categories. This saves a great deal of otherwise redundant work and highlights the tight connection between these categories of domains. This unification is made possible by a minor deviation from the usual definitions dealing with algebraic domains and compact elements, as explained later in this paper.

A second proof-engineering advance is the fact that I do not formalize domains in the usual way (as a subcategory of complete partial orders with Scott-continuous functions), but instead develop the category formed by the *bases* of profinite domains with approximable relations as arrows. This category is equivalent to the category of profinite domains via ideal completion. I can save some effort because rather than building constructions on CPOs as well as on their bases, I need only perform constructions on bases. With the notable exception of the function space, constructions on the bases of algebraic domains are simpler than corresponding constructions on CPOs (especially in the pointed case). This formalization of profinite domains via their bases (the Plotkin orders) mirrors the development of Scott domains via the Scott information systems [25]. I am not aware of any mechanized development of Scott information systems.

The goal of this work is ultimately to provide a solid foundation of domain theoretic constructions and theorems that can be used for practical program semantics in Coq. My focus on constructive mathematics, however, stems from a personal philosophy that values constructive methods *per se*; this perspective is especially well-justified for domain theory, which aims to be a mathematical model of computation. Nonetheless, I derive a practical benefit, which is that my development of domain theory asserts *no additional axioms at all* over the base metatheory of Coq — this makes my development compatible with any axiomatic extension of Coq, even unusual ones (say, anticlassical extensions).

In this paper, I will explain the high-level ideas of the proof and present the main definitions. The interested reader is encouraged to consult the formal proof development for details; it may be found at the author's website.[2] Knowledge of Coq is not required to understand the main ideas of this paper; however, a novice grasp of category theory concepts will be required for some sections.

2 Basic Definitions

We start with a key difference between my proof development and textbook references on domain theory. Usually, domain theory is concerned with certain classes of *partial orders*: types equipped with an order relation that is transitive,

[2] http://rwd.rdockins.name/domains/

reflexive and antisymmetric. I will be working instead with the *preorders*: types equipped with an order relation that is merely reflexive and transitive. The antisymmetry condition is dropped. This leads us to working with *setoids*, sets equipped with an equivalence relation, and setoid homomorphisms, functions that respect setoid equivalence. Using setoids is a common technique in Coq because quotients are not readily available [5].

Definition 1 (Setoid). *Suppose A is a type, and $\approx$ is a binary relation on A. We say $\langle A, \approx\rangle$ is a setoid provided $\approx$ is reflexive, transitive and symmetric.*

Throughout this paper (and the vast majority of the formal proof), the only notion of equality we will be interested in is the $\approx$ relation of setoids. In the formal proof, many of the definitions require axioms stating that various operations preserve $\approx$ equality. In this paper, I will elide such axioms and concerns about the preservation of equality.

Definition 2 (Preorder). *Suppose A is a type and $\sqsubseteq$ is a binary relation on A. We say $\langle A, \sqsubseteq\rangle$ is a preorder provided $\sqsubseteq$ is reflexive and transitive. Furthermore, we automatically assign to A a setoid where $x \approx y$ iff $x \sqsubseteq y \wedge y \sqsubseteq x$.*

Every preorder automatically induces a setoid; because we work up to $\approx$, we obtain "antisymmetry on preorders" by convention.

Definition 3 (Finite set). *Suppose $\langle A, \sqsubseteq\rangle$ is a preorder.*[3] *Then we can consider the type* $\mathsf{list}\ A$ *to be a preorder of* finite sets of A. *We say that an element x is in the finite set l, and write $x \in l$ provided $\exists x'.\ \mathsf{In}\ x'\ l \wedge x \approx x'$. Here* In *refers to the Coq standard library predicate for list membership. We can then equip finite sets with the inclusion order where $l_1 \subseteq l_2$ iff $\forall x.\ x \in l_1 \to x \in l_2$.*

Note that finite set membership is defined so that it respects $\approx$ equality.

The difference between unpointed directed-complete partial orders (which might not have a least element) and pointed DCPOs (those having a least element) can be expressed entirely in terms of whether certain finite sets are allowed to be empty or not. Likewise, the Scott-continuous functions are distinguished from the *strict* Scott-continuous functions; and the profinite domains from the pointed profinite domains. Therefore, we make the following technical definition that allows us to uniformly state definitions that are valid in both settings.

Definition 4 (Conditionally inhabited). *Suppose A is a preorder, l is a finite set of A and h is a boolean value. Then we say l is* conditionally inhabited *and write $\mathsf{inh}_h\ l$ provided either $h = false$ or $h = true \wedge \exists x.\ x \in l$.*

When h is false, l need not be inhabited; however, when h is true, $\mathsf{inh}_h\ l$ requires l to be inhabited. This strange little definition turns out to be the key to achieving a uniform presentation of pointed and unpointed domains.

[3] It suffices to suppose A is a setoid, but the objects of interest are always preorders, so this is mildly simpler. Likewise for the enumerable sets below.

Definition 5 (Enumerable set). *Suppose A is a preorder. Then the functions $\mathbb{N} \to \mathsf{option}\ A$ are* enumerable sets of A. *If X is an enumerable set and x is an element of A, we write $x \in X$ iff $\exists x'\ n.\ X(n) = \mathsf{Some}\ x' \wedge x \approx x'$. As with finite sets, we write $X \subseteq Y$ iff $\forall x.\ x \in X \to x \in Y$.*

Here $\mathbb{N}$ is the Coq standard library type of binary natural numbers and $\mathsf{option}\ A$ is an inductive type containing either $\mathsf{Some}\ x$ for some x in A or the distinguished element None. If we interpret Coq's metalogic as a constructive logic of computable functions then the enumerable sets defined here are a good analogue for the recursively enumerable sets (recall a set is recursively enumerable iff it is the range of a partial recursive function). I avoid using the terms *recursive* or *recursively enumerable* so as not to invoke the machinery of recursive function theory, which I have not explicitly developed.

Throughout this development of domain theory, the enumerable sets will fill in where, in more classical presentations, "ordinary" sets would be. The use of enumerable sets is one of the primary ways I achieve an effective presentation of domain theory. In the rest of this paper, I will freely use set-comprehension notation when defining sets; the diligent reader may wish to verify for himself that sets so defined are actually represented by some concrete enumeration function.

Definition 6 (Directed set). *Suppose A is a preorder, h is a boolean value, and X is an enumerable set of A. We say that X is* directed *(or h-directed), and write $\mathsf{directed}_h\ X$, if every finite subset $l \subseteq X$ where $\mathsf{inh}_h\ l$ has an upper bound in X.*

This definition differs from the standard one. My definition agrees when $h = false$; but when $h = true$, only the *inhabited* finite subsets must have upper bounds in X. As a consequence, when $h = false$, $\mathsf{directed}_h\ X$ implies X is nonempty (because the empty set must have an upper bound in X); but when $h = true$, $\mathsf{directed}_h\ X$ holds even when X is empty (in which case the condition holds vacuously because there are no finite inhabited subsets of the empty set).

Definition 7 (Directed-complete partial order). *Suppose A is a preorder and let h be a boolean value. Let $\bigsqcup X$ be an operation that, for every h-directed enumerable set X of A, calculates the least upper bound of X. Then we say $\langle A, \bigsqcup \rangle$ is a directed-complete partial order (with respect to h). The category of directed-complete partial orders is named $\mathbf{DCPO}_h$ when paired with the Scott-continuous functions.*

Definition 8 (Scott-continuous function). *Suppose A and B are in $\mathbf{DCPO}_h$ and $f : A \to B$ is a function from A to B. Then we say f is Scott-continuous if f is monotone and satisfies the following for all h-directed sets X:*

$$f(\bigsqcup X) \sqsubseteq \bigsqcup(\mathsf{image}\ f\ X)$$

Pause once again to reflect on the role of the parameter h. When $h = false$ the empty set is not considered directed, and thus may not have a supremum;

this gives unpointed DCPOs. On the other hand when $h = true$, the empty set must have a supremum, which is *a fortiori* the least element of the DCPO. Likewise for Scott-continuous functions, $h = false$ gives the standard Scott-continuous functions on unpointed DCPOs; whereas $h = true$ gives the *strict* Scott-continuous functions on pointed DCPOs.

Next we move on to definitions relating to algebraic domains. The main technical definition here is the "way below" relation, which gives rise to the compact elements.

Definition 9 (Way-below, compact element). *Suppose A is in* $\mathbf{DCPO}_h$ *and let x and y be elements of A. Then we say x is way-below y (and write $x \ll y$) provided that, for every h-directed set X where $y \sqsubseteq \bigsqcup X$, there exists x' such that $x \sqsubseteq x'$ and $x' \in X$. An element x of A is* compact *if $x \ll x$.*

This is the standard statement of the way-below relation (also called the "order of approximation") and compact elements, except that we have specified the h-directed sets instead of the directed sets. With $h = false$, this is just the standard definition of way-below and compact elements for unpointed DCPOs. However, when $h = true$ it means that the least element of a pointed DCPO is *not* compact (because X may be the empty set). Although readers already familiar with domain theory may find this puzzling, it actually is quite good, as it simplifies constructions on pointed domains (see §4).

Definition 10 (Effective preorder). *Suppose A is a preorder. Then we say A is an* effective preorder *provided it is equipped with a decision procedure for $\sqsubseteq_A$ and an enumerable set containing every element of A.*

In order to ensure that all the constructions we want to perform can be done effectively, we need to limit the scope of our ambitions to the effective preorders and effective domains.

Definition 11 (Effective algebraic domain). *Suppose A is in* $\mathbf{DCPO}_h$*. A is an* algebraic domain *provided that for all x in A, the set $\{b \mid b \ll x \wedge b \ll b\}$ is enumerable, h-directed and has supremum x. A is furthermore called* effective *if the set of compact elements is enumerable, and the $\ll$ relation is decidable on compact elements.*

Said another way, a DCPO is an algebraic domain if every element arises as the supremum of the set of compact elements way-below it. The set of compact elements of an algebraic domain is also called the basis.[4] Note that an effective domain is *not* necessarily an effective preorder; merely its basis is effective.

Definition 12 (Basis). *Suppose A is an effective algebraic domain. Then let* $\mathsf{basis}(A)$ *be the preorder consisting of the compact elements of A, with $\ll$ as the ordering relation. Note that $\ll$ is always transitive; furthermore, $\ll$ is reflexive on the compact elements of A by definition.* $\mathsf{basis}(A)$ *is an effective preorder because A is an effective domain.*

[4] The more general class of *continuous* domains may have a basis that is distinct from the set of compact elements; I do not consider continuous domains.

The compact elements of an effective algebraic domain form an effective preorder. Furthermore, the basis preorder uniquely (up to isomorphism) determines an algebraic domain.

Definition 13 (Ideal completion). *Suppose A is an effective preorder. We say an enumerable subset X of A is an* ideal *provided it is h-directed and downward closed with respect to $\sqsubseteq_A$. Let* $\mathsf{ideal}(A)$ *be the preorder of ideals of A ordered by subset inclusion. Then* $\mathsf{ideal}(A)$ *is a DCPO (with enumerable set union as the supremum operation) and an effective algebraic domain.*

The proof that $\mathsf{ideal}(A)$ is an algebraic domain is standard (see [2, §2.2.6]), as is the following theorem.

Theorem 1. *The effective algebraic domains are in one-to-one correspondence with their bases. In particular A is isomorphic to* $\mathsf{ideal}(\mathsf{basis}(A))$ *for all algebraic domains A and B is isomorphic (as a preorder) to* $\mathsf{basis}(\mathsf{ideal}(B))$ *for all effective preorders B.*

This justifies our position, in the rest of this paper, of considering *just* the bases of domains rather than algebraic domains *per se*. Although the formal proofs we develop will involve only bases and certain kinds of *approximable relations* between bases, the reader may freely draw intuition from the more well-known category of algebraic domains and Scott-continuous functions.

3 Profinite Domains and Plotkin Orders

The profinite domains are fairly well-known. They are very closely related to Plotkin's category SFP (Sequences of Finite inductive Posets) [20]. In fact, when limited to effective bases, the category of pointed profinite domains has the same objects as SFP; SFP, however, is equipped with the *nonstrict* continuous functions, whereas the pointed profinite domains are equipped with the strict continuous functions. Unpointed profinite domains are the largest cartesian closed full subcategory of algebraic domains with countable bases [14], which justifies our interest in them. I suspect that the pointed profinite domains are likewise the largest *monoidal* closed subcategory of countably-based pointed algebraic domains when considered with strict continuous functions.[5]

My development of the profinite domains (more properly, their bases: the Plotkin orders) roughly follows the strategy found in chapter 2 of Gunter's thesis [12], incorporating some modifications due to Abramsky [1]. In addition, I have modified some definitions to make them more constructive and to incorporate the h parameter for selecting pointed or unpointed domains.

The central concept in this strategy is the *Plotkin order*. Plotkin orders are preorders with certain completeness properties based on *minimal* upper bounds.

[5] A nearby result holds: the pointed profinite domains are the largest cartesian closed category of countably-based algebraic domains when considered with the nonstrict continuous functions.

Definition 14 (Minimal upper bound). *Suppose A is a preorder, and let X be a set of elements of A. We say m is a* minimal upper bound *of X provided that m is an upper bound for X; and for every other upper bound of X, m' where $m' \sqsubseteq m$, we have $m \approx m'$.*

Note that minimal upper bounds are subtly different than *least* upper bounds. There may be several distinct minimal upper bounds (MUBs) of a set; in contrast, *least* upper bounds (AKA suprema) must be unique (up to $\approx$ as usual).

Definition 15 (MUB complete preorder). *Suppose A is a preorder. Then we say A is* MUB complete *if for every finite subset l of elements of A where $\mathsf{inh}_h\ l$ and where z is an upper bound of l, there exists a minimal upper bound m of l with $m \sqsubseteq z$.*

In a MUB-complete preorder every finite set bounded-above by some z has a minimal upper bound below z. Here we merely assert the desired MUB to exist, but it can actually be computed when A is an effective Plotkin order.

Definition 16 (MUB closed set). *Suppose A is a preorder and X is a subset of A. We say X is* MUB closed *if for every finite subset $l \subseteq X$ with $\mathsf{inh}_h\ l$, every minimal upper bound of l is in X.*

Definition 17 (Plotkin order). *Suppose A is a preorder, and let Θ be an operation (called the MUB closure) from finite sets of A to finite sets of A. Then we say $\langle A, \Theta\rangle$ is a* Plotkin order *(with respect to h) provided the following hold for all finite sets l of A:*

1. *A is MUB complete;*
2. *$l \subseteq \Theta(l)$;*
3. *$\Theta(l)$ is MUB closed; and*
4. *$\Theta(l)$ is the smallest set satisfying the above.*

Note that for a given preorder A, Θ is uniquely determined when it exists. As such, we will rarely give Θ explicitly, preferring instead simply to assert that a suitable Θ can be constructed.

Some of the constructions we wish to do require additional computational content from domain bases; we therefore require them to be *effective preorders* in addition to being Plotkin. Thus, the effective Plotkin preorders will be the domain structures we are interested in. However, it still remains to define the arrows between domains. Unlike the usual case with concrete categories, the arrows between domains are not functions, but certain kinds of relations. These *approximable relations*, however, induce functions on ideal completions in a unique way (see [12, §2.1]), so they nonetheless behave very much like functions.

Definition 18 (Approximable relation). *Let A and B be effective Plotkin orders, and let $R \subseteq A \times B$ be an enumerable relation (i.e., enumerable set of pairs). Then we say R is an* approximable relation *provided the following hold:*

- *$x \sqsubseteq x' \wedge y' \sqsubseteq y \wedge (x, y) \in R$ implies $(x', y') \in R$; and*

– *the set* $\{y \mid (x, y) \in R\}$ *is* h*-directed for all* x*.*

These requirements seem mysterious at first, but they are precisely what is required to ensure that approximable relations give rise to Scott-continuous functions via the ideal completion construction.

Theorem 2. *Suppose* A *is an effective Plotkin order. Then* $\{(x, y) \mid y \sqsubseteq_A x\}$ *is an approximable relation; call it* id_A*.*

Suppose A*,* B*, and* C *are effective Plotkin orders; let* R *be an approximable relation from* A *to* B *and* S *be an approximable relation from* B *to* C*. Then the composed relation* $S \circ R \equiv \{(x, z) \mid \exists y.\ (x, y) \in R \wedge (y, z) \in S\}$ *is an approximable relation.*

Definition 19 ($\mathbf{PLT}_h$). *Let* h *be a boolean value. Then* $\mathbf{PLT}_h$ *is the category whose objects are the effective Plotkin orders (with parameter* h*) and whose arrows are the approximable relations (again with* h*). The previous theorem gives the construction for the identity and composition arrows; the associativity and unit axioms are easily proved.*

$\mathbf{PLT}_{false}$ is equivalent to the category of unpointed profinite domains with Scott-continuous functions and $\mathbf{PLT}_{true}$ is equivalent to the category of pointed profinite domains with strict Scott-continuous functions via the ideal completion construction discussed above.

Definition 20 (Undefined relation). *Let* A *and* B *be objects of* $\mathbf{PLT}_{true}$*. Let* $\bot \equiv \emptyset$ *be the empty approximable relation between* A *and* B*.* $\bot$ *corresponds to the undefined function that sends every argument to the least element of* B*.*

Note that, while $\bot \equiv \emptyset$ is a perfectly valid approximable relation in $\mathbf{PLT}_{true}$, the empty relation is *not* an approximable relation in $\mathbf{PLT}_{false}$ (unless A is empty). This is because of the second property of approximable relations, which requires $\{y \mid (x, y) \in R\}$ to be h-directed for each x. The empty set is h-directed for $h = true$, but not for $h = false$. Further, note that in $\mathbf{PLT}_{true}$, $f \circ \bot \approx \bot \approx \bot \circ g$, so composition is strict on both sides.

4 Constructions on Plotkin Orders

Now that we have defined the category of effective Plotkin orders, we can begin to build some of the standard constructions: products, sums and the function space. Our strategy of unifying pointed and unpointed domains will now begin to pay dividends, as these constructions need be performed only once.

Definition 21 (Unit and empty orders). *Let* 0 *represent the empty preorder. It is easy to see that* 0 *is effective and Plotkin. Let* 1 *represent the unit preorder, having a single element;* 1 *is also effective and Plotkin.*

Definition 22 (Products). *Suppose A, B and C are effective Plotkin orders with parameter h. Then $A \times B$ (product preorder) is an effective Plotkin preorder with parameter h. Set $\pi_1 \equiv \{((x,y),x') \mid x' \sqsubseteq x\}$ and $\pi_2 \equiv \{((x,y),y') \mid y' \sqsubseteq y\}$. Suppose $f : C \to A$ and $f : C \to B$ are approximable relations. Then set $\langle f,g \rangle \equiv \{(z,(x,y)) \mid (z,x) \in f \wedge (z,y) \in g\}$. π_1 and π_2 represent pair projections and $\langle f,g \rangle$ is the paring operation.*

Theorem 3. *The product construction is the categorical product in $\mathbf{PLT}_{false}$. In addition, 1 is the terminal object; making $\mathbf{PLT}_{false}$ a cartesian category. In $\mathbf{PLT}_{false}$, the product is denoted $A \times B$.*

Theorem 4. *The product construction is* not *the categorical product in $\mathbf{PLT}_{true}$. It is instead the "smash" product, or the strict pair; we denote the smash product as $A \otimes B$. Although not the categorical product, $\otimes$ gives $\mathbf{PLT}_{true}$ the structure of a symmetric monoidal category, with 1 as the unit object.*

In $\mathbf{PLT}_{true}$, π_1, π_2 and $\langle f,g \rangle$ are all still useful operations; they are just not the projection and pairing arrows for the categorical product. They are instead "strict" versions that satisfy laws like $\langle x, \bot \rangle \approx \bot$ and $\pi_1 \circ \langle x,y \rangle \sqsubseteq x$.

Definition 23 (Sums). *Suppose A, B and C are effective Plotkin preorders with parameter h. Then $A + B$ (disjoint sum) is an effective Plotkin preorder with parameter h. Set $\iota_1 \equiv \{(x, \mathsf{inl}\ x') \mid x' \sqsubseteq x\}$ and $\iota_2 \equiv \{(y, \mathsf{inr}\ y') \mid y' \sqsubseteq y\}$. Suppose $f : A \to C$ and $f : B \to C$ are approximable relations. Then set $[f,g] \equiv \{(\mathsf{inl}\ x, z) \mid (x,z) \in f\} \cup \{(\mathsf{inr}\ y, z) \mid (y,z) \in g\}$. ι_1 and ι_2 are the sum injections and $[f,g]$ is the case analysis operation.*

Theorem 5. *In $\mathbf{PLT}_{false}$, the above sum construction is the categorical coproduct, which we denote $A + B$. In addition, 0 is the initial object. Thus $\mathbf{PLT}_{false}$ is a cocartesian category.*

Theorem 6. *In $\mathbf{PLT}_{true}$, the sum construction above gives the "coalesced sum," which identifies the bottom elements of the two objects; it is denoted $A \oplus B$. Like $+$, $\oplus$ is the the categorical coproduct in $\mathbf{PLT}_{true}$. Furthermore 0 serves as the initial object. Thus $\mathbf{PLT}_{true}$ is also a cocartesian category.*

Theorem 7. *In $\mathbf{PLT}_{true}$, the empty preorder 0 is also the terminal object.*

This series of results reveals some deep connections between the structure $\mathbf{PLT}_{false}$ and $\mathbf{PLT}_{true}$. Not only are $\times$ and $\otimes$ intuitively closely related, they are literally the same construction. Likewise for $+$ and $\oplus$. This coincidence goes even further; the "function space" construction in both categories is likewise the same. This construction is based on the concept of "joinable" relations. My definition is a minor modification of the one given by Abramsky [1].

Definition 24 (Joinable relation). *Suppose A and B are objects of $\mathbf{PLT}_h$. Let R be a* finite *set of pairs in $A \times B$. We say R is a* joinable relation *if the following hold:*

– $\mathsf{inh}_h\ R$; *and*
– *for all finite sets* G *with* $G \subseteq R$ *and* $\mathsf{inh}_h\ G$, *and for all* x *where* x *is a minimal upper bound of* $\mathsf{image}\ \pi_1\ G$, *there exists* y *where* y *is an upper bound of* $\mathsf{image}\ \pi_2\ G$ *and* $(x, y) \in R$.

Unfortunately, this definition is highly technical and difficult to motivate, except by the fact that it yields the expected exponential object. The rough idea is that R is supposed to be a finite fragment of an approximable relation. The complicated second requirement ensures that joinable relations are "complete enough" that the union of a directed collection of joinable relations makes an approximable relation and that we can compute finite MUB closures.

Definition 25 (Function space). *Suppose* A *and* B *are objects of* $\mathbf{PLT}_h$. *The joinable relations from* A *to* B *form a preorder where we set* $G \sqsubseteq H$ *iff* $\forall x\ y.\ (x, y) \in G \to \exists x'\ y'.\ (x', y') \in H \wedge x' \sqsubseteq x \wedge y \sqsubseteq y'$. *Moreover, this preorder is effective and Plotkin; we denote it by* $A \Rightarrow B$.

Now, suppose $f : C \times A \to B$ *is an approximable relation.*[6] *Then* $\mathsf{curry}\ f : C \to (A \Rightarrow B)$ *and* $\mathsf{app} : (A \Rightarrow B) \times A \to B$ *are approximable relations as defined below:*

$$\mathsf{curry}\ f \equiv \{(c, R) \mid \forall x\ y.\ (x, y) \in R \to ((c, x), y) \in f\}$$
$$\mathsf{app} \equiv \{((R, x), y) \mid \exists x'\ y'.\ (x', y',) \in R \wedge x' \sqsubseteq x \wedge y \sqsubseteq y'\}$$

The proof that the function space construction forms a Plotkin order is one of the most involved proofs in this entire development. My proof takes elements from those of Gunter [12] and Abramsky [1]. The reader may consult the formal proof development for details.

Theorem 8. $\Rightarrow$ *is the exponential object in* $\mathbf{PLT}_{false}$ *and makes* $\mathbf{PLT}_{false}$ *into a cartesian closed category.*

Theorem 9. *In* $\mathbf{PLT}_{true}$, $\Rightarrow$ *constructs the exponential object with respect to* $\otimes$ *and makes* $\mathbf{PLT}_{true}$ *a monoidal closed category. In* $\mathbf{PLT}_{true}$, *we use the symbol* $\multimap$ *instead of* $\Rightarrow$.

Here is a huge payoff for our strategy of giving uniform constructions for $\mathbf{PLT}_h$; the proofs and constructions leading to $\Rightarrow$ and the MUB closure properties are technical and lengthy. Furthermore, the pointed and unpointed cases differ only in a few localized places. With this proof strategy, these difficult proofs need only be done once.

5 Lifting and Adjunction

One standard construction we have not yet seen is "lifting," which adds a new bottom element to a domain. In our setting, lifting is actually split into two

[6] Note, here $\times$ refers generically to the product construction in $\mathbf{PLT}_h$, not just the categorical product of $\mathbf{PLT}_{false}$.

pieces: a forgetful functor from pointed to unpointed domains, and a lifting functor from unpointed to pointed domains. These functors are adjoint, which provides a tight and useful connection between these two categories of domains.

However, working with bases instead of algebraic domains *per se* causes an interesting inversion to occur. When working with $\mathbf{PLT}_h$ instead of profinite domains as such, the functor that is the forgetful functor and the one that actually does lifting exchange places.

First let us consider the functor that passes from pointed domains ($\mathbf{PLT}_{true}$) to unpointed domains ($\mathbf{PLT}_{false}$). This is the one usually known as the forgetful functor; it forgets the fact that the domains are pointed and that the functions are strict.

Definition 26 ("Forgetful" functor). *Suppose A is an object of $\mathbf{PLT}_{true}$. Then let* option *A be the preorder that adjoins to A a new bottom element,* None*. Then* option *A is an element of $\mathbf{PLT}_{false}$. Furthermore, suppose $f : A \to B$ is an approximable relation in $\mathbf{PLT}_{true}$. Then $g :$ option $A \to$ option B is an approximable relation in $\mathbf{PLT}_{false}$ defined by:*

$$g \equiv \{(x, \mathsf{None})\} \cup \{(\mathsf{Some}\ x, \mathsf{Some}\ y) \mid (x, y) \in f\}.$$

These operations produce a functor $U : \mathbf{PLT}_{true} \to \mathbf{PLT}_{false}$.

Why is adding a new element the right thing to do? Recall from earlier that our definition of the way-below relation and compact elements *excludes* the bottom element of pointed domains, in contrast to the usual definitions. When we consider the bases of pointed domains, the bottom element is implicit; is is the *empty set* of basis elements (which is an ideal when $h = true$) that represents bottom. It is because the bottom element is implicit that makes all the constructions from the previous section work uniformly in both categories.

When passing to unpointed domains, the empty set is no longer directed and the implicit bottom element must become *explicit*. This is why the forgetful functor actually adds a new basis element. In contrast, the "lifting" functor is more like a forgetful functor in that there is nothing really to do. Passing from unpointed to pointed domains automatically adds the new implicit bottom element, so the basis does not change.

Definition 27 ("Lifting" functor). *Suppose A is an object of $\mathbf{PLT}_{false}$; then A is also an object of $\mathbf{PLT}_{true}$. Furthermore, if $f : A \to B$ is an approximable relation in $\mathbf{PLT}_{false}$ then f is also an approximable relation in $\mathbf{PLT}_{true}$. These observations define a functor $L : \mathbf{PLT}_{false} \to \mathbf{PLT}_{true}$.*

Theorem 10. *The lifting functor L is left adjoint to the forgetful functor U.*

Said another way, the adjunction between L and U means that there is a one-to-one correspondence between the strict $\mathbf{PLT}_{true}$ arrows $L(X) \to Y$ and the nonstrict $\mathbf{PLT}_{false}$ arrows $X \to U(Y)$. This adjunction induces a structure on $\mathbf{PLT}_h$ that is a model of dual intuitionistic linear logic (DILL) [6, 4], which can be used to combine the features of strict and nonstrict computation into a nice, unified theory.

6 Powerdomains

Powerdomains provide operators on domains that are analogues to the standard powerset operation on sets [20]; powerdomains can be used to give semantics to nondeterminism and to set-structured data (like relational tables). Each of the three standard powerdomains operations (upper, lower and convex) [2] can be constructed in both $\mathbf{PLT}_{false}$ and in $\mathbf{PLT}_{true}$, for a total of six powerdomain operators. Again, our uniform presentation provides a significant savings in work.

Definition 28 (Powerdomains). *Suppose X is an element of $\mathbf{PLT}_h$. Let the finite h-inhabited sets of X be the elements of the powerdomain. The preorder on domain elements is one of the following: $\sqsubseteq^\flat$ for the lower powerdomain, $\sqsubseteq^\sharp$ for the upper powerdomain, and $\sqsubseteq^\natural$ for the convex powerdomain.*

$$a \sqsubseteq^\flat b \equiv \forall x \in a.\ \exists y \in b.\ x \sqsubseteq_X y$$
$$a \sqsubseteq^\sharp b \equiv \forall y \in b.\ \exists x \in a.\ x \sqsubseteq_X y$$
$$a \sqsubseteq^\natural b \equiv a \sqsubseteq^\flat b \wedge a \sqsubseteq^\sharp b$$

In each case, the resulting preorder is effective and Plotkin, making it again an object of $\mathbf{PLT}_h$.

Of these, the most mathematically natural (that is, most like the powerset operation) is probably the convex powerdomain in unpointed domains ($\mathbf{PLT}_{false}$). Unlike the convex powerdomain in pointed domains, the unpointed version has a representation for the empty set.

7 Solving Recursive Domain Equations

To get a usable domain for semantics, we frequently want to be able to solve recursive domain equations. Indeed, much of the impetus for domain theory was originally motivated by the desire to build a semantic models for the lambda calculus [22]. The classic example is the simple recursive equation $D \cong (D \Rightarrow D)$.

My approach to this problem is the standard one, which is to take bilimits of continuous functors expressed in categories of embedding-projection pairs [21]. The main advantage of this technique is that it turns mixed-variance functors on $\mathbf{PLT}_h$ (like the function space) into covariant functors on the category of EP-pairs. Such covariant functors can then be handled via standard fixpoint theorems. Furthermore, isomorphisms constructed in categories of EP-pairs yield isomorphisms in the base category.

Thus, we need to construct the category of EP-pairs over $\mathbf{PLT}_h$. However, it is significantly more convenient to work in a different, but equivalent category: the category of basis embeddings. To the best of my knowledge, this category has not been previously defined.

Definition 29 (Basis embedding). *Suppose A and B are objects of $\mathbf{PLT}_h$ and let f be a function (N.B., not a relation) from A to B. Then we say f is a* basis embedding *if the following hold:*

– $a \sqsubseteq_A b$ *iff* $f(a) \sqsubseteq_B f(b)$ *(*f *is monotone and reflective);*
– *for all* y*, the set* $\{x \mid f(x) \sqsubseteq_B y\}$ *is* h*-directed.*

Let $\mathbf{BE}_h$ *represent the category of effective Plotkin orders with basis embeddings as arrows.*

Every basis embedding gives rise to an embedding-projection pair in $\mathbf{PLT}_h$, and likewise every EP-pair gives rise to a basis embedding[7]; these are furthermore in one-to-one correspondence. This is sufficient to show that $\mathbf{BE}_h$ and the category of EP-pairs on $\mathbf{PLT}_h$ are equivalent categories.

Now we can use a standard fixpoint theorem (essentially, the categorical analogue of Kleene's fixpoint theorem) to build least fixpoints of *continuous* functors. Continuous functors are those that preserve directed colimits [2].

Theorem 11. *Suppose* $\mathcal{C}$ *is a category with initial object* 0 *that has directed colimits for all directed systems, and let* $F : \mathcal{C} \to \mathcal{C}$ *be a continuous functor. Then* F *has an initial algebra* D *and* $D \cong F(D)$*.*

When combined with the next theorem, this allows us to construct fixpoints of functors in $\mathbf{BE}_{true}$, the category basis embeddings over pointed domains.

Theorem 12. $\mathbf{BE}_h$ *has directed colimits for all directed systems.*

In fact, $\mathbf{PLT}_{true}$ is in the class of CPO-algebraically ω-compact categories [10, §7], which are especially well-behaved for building recursive types.[8]

We cannot apply the same strategy to $\mathbf{BE}_{false}$, because the category of embeddings over unpointed domains fails to have an initial object. However, this is not a problem, because we can import constructions from $\mathbf{BE}_{true}$ into $\mathbf{BE}_{false}$ by passing through the adjoint functors L and U. Passing through L may add "extra" bottom elements, but it does so in places that are consistent with standard practice (e.g., the canonical model for lazy λ-calculus [3]). Indeed, this setup partially explains *why* those extra bottoms appear.

The following theorems allow us to find fixpoint solutions to recursive domain equations for many interesting functors by building up continuous functors from elementary pieces.

Theorem 13. *The identity and constant functors are continuous.*

Theorem 14. *The composition of two continuous functors is continuous.*

Theorem 15. $\times$*,* $+$ *and* $\Rightarrow$ *extend to continuous functors on* $\mathbf{BE}_{false}$*.*

Theorem 16. $\otimes$*,* $\oplus$ *and* $\multimap$ *extend to continuous functors on* $\mathbf{BE}_{true}$*.*

Theorem 17. *The forgetful functor* $U : \mathbf{BE}_{true} \to \mathbf{BE}_{false}$ *is continuous.*

[7] Surprisingly, (to me, at least) this can be done in an entirely constructive way using an indefinite description principle that can be proved for enumerable sets.

[8] This is not yet formalized; initial attempts have triggered universe consistency issues for which I have yet to find a solution.

Theorem 18. *The lifting functor* $L : \mathbf{BE}_{false} \to \mathbf{BE}_{true}$ *is continuous.*

Theorem 19. *The lower, upper and convex powerdomains are all continuous.*

Now we can construct a wide variety of interesting recursive semantic domains for both pointed and unpointed domains. For example, we can construct the domain $D \cong (D \multimap D)$, representing eager λ-terms; and we can also construct $E \cong L(U(E) \Rightarrow U(E))$, the canonical model of lazy λ-terms. Furthermore, algebraic data types in the style of ML or Haskell can be constructed using combinations of sums and products. Sophisticated combinations of strict and nonstrict features can be built using the adjunction functors U and L, and nondeterminism may be modeled using the powerdomains.

8 Implementation

The entire proof development consists of a bit over 30KLOC, not including the examples. This includes a development of some elementary category theory and the fragments of set theory we need to work with finite and enumerable sets. The proof development has been built using Coq version 8.4; it may be found at the author's personal website.[9] The development includes examples demonstrating soundness and adequacy for four different systems (the simply-typed SKI combinator calculus with and without fixpoints, and the simply-typed λ-calculus with and without fixpoints). Additional examples are in progress.

9 Conclusion and Future Work

I have presented a high-level overview of a formal development of domain theory in the proof assistant Coq. The basic trajectory of the proof follows lines well-established by prior work. My presentation is fully constructive and mechanized — the first such presentation of domain theory based on profinite domains. I show how some minor massaging of the usual definitions leads to a nice unification for most constructions in pointed and unpointed domains. The system is sufficient to build denotational models for a variety of programming language semantics.

Currently I have no explicit support for parametric polymorphism. Polymorphism requires fairly complex machinery, and I have not yet undertaken the task. There seem to be two possible paths forward: one approach is explained by Paul Taylor in his thesis [23]; the other is the $PILL_Y$ models of Birkedal *et al.* [8]. This latter method has the significant advantage that it validates the parametricity principle, which underlies interesting results like Wadler's free theorems [24]. I hope to build one or both of these systems for polymorphism in the future.

The Coq formalization of Benton *et al.* [7] uses a more modern method for building recursive domains based on locally-continuous bifunctors in compact categories [11]. Their approach has some advantages; Pitts's invariant relations [19], especially, provide useful reasoning principles for recursively defined domains. I hope to incorporate these additional techniques in future work.

[9] http://rwd.rdockins.name/domains/

References

1. Abramsky, S.: Domain theory in logical form. Annals of Pure and Applied Logic 51, 1–77 (1991)
2. Abramsky, S., Jung, A.: Handbook of Logic in Computer Science, vol. 3, chap. Domain Theory, pp. 1–168. Clarendon Press (1994)
3. Abramsky, S., Ong, C.H.L.: Full abstraction in the lazy lambda calculus. Information and Computation 105, 159–267 (1993)
4. Barber, A.: Linear Type Theories, Semantics and Action Calculi. Ph.D. thesis, Edinburgh University (1997)
5. Barthe, G., Capretta, V.: Setoids in type theory. Journal of Functional Programming 13(2), 261–293 (March 2003)
6. Benton, N.: A mixed linear and non-linear logic: Proofs, terms and models. In: Computer Science Logic. LNCS, vol. 933 (1994)
7. Benton, N., Kennedy, A., Varming, C.: Some domain theory and denotational semantics in Coq. In: LNCS. vol. 5674, pp. 115–130 (2009), TPHOLs
8. Birkedal, L., Møgelberg, R., Petersen, R.: Domain-theoretical models of parametric polymorphism. Theoretical Computer Science 288, 152–172 (2007)
9. Egli, H., Constable, R.L.: Computability concepts for programming language concepts. Theoretical Computer Science 2, 133–145 (1976)
10. Fiore, M.P.: Axiomatic Domain Theory in Categories of Partial Maps. Ph.D. thesis, University of Edinburgh (1994)
11. Freyd, P.: Remarks on algebraically compact categories. In: Applications of Categories in Computers Science. London Mathematical Society Lecture Note Series, vol. 177, pp. 95–106. Cambridge University Press (1991)
12. Gunter, C.: Profinite Solutions for Recursive Domain Equations. Ph.D. thesis, Carnegie-Mellon University (1985)
13. Huffman, B.: A purely definitional universal domain. In: TPHOLs (2009)
14. Jung, A.: Cartesian Closed Categories of Domains. Ph.D. thesis, Centrum voor Wiskunde en Informatica, Amsterdam (1988)
15. Milner, R.: Logic for computable functions: Description of a machine implementation. Tech. Rep. STAN-CS-72-288, Stanford University (May 1972)
16. Milner, R.: Models of LCF. Tech. Rep. STAN-CS-73-332, Stanford (1973)
17. Müller, O., Nipkiw, T., von Oheimb, D., Slotosch, O.: HOLCF = HOL + LCF. Journal of Functional Programming 9 (1999)
18. Paulin-Mohring, C.: A constructive denotational semantics for Kahn networks. In: From Semantics to Computer Sciences. Essays in Honour of G. Kahn. Cambridge University Press (2009)
19. Pitts, A.M.: Relational properties of domains. Information and Computation 127 (1996)
20. Plotkin, G.: A powerdomain construction. SIAM J. of Computing 5, 452–487 (1976)
21. Plotkin, G., Smyth, M.: The category theoretic solution of recursive domain equations. Tech. rep., Edinburgh University (1978)
22. Scott, D.: Outline of a mathematical theory of computation. Tech. Rep. PRG02, OUCL (November 1970)
23. Taylor, P.: Recursive Domains, Indexed Category Theory and Polymorphism. Ph.D. thesis, University of Cambridge (1986)
24. Wadler, P.: Theorems for free! In: Intl. Conf. on Functional Programming and Computer Architecture (1989)
25. Winskell, G.: The Formal Semantics of Programming Languages: An Introduction. MIT Press (1993)