Power Domains Supporting Recursion and Failure

Reinhold Heckmann

FB 14 – Informatik
Universität des Saarlandes
6600 Saarbrücken
Bundesrepublik Deutschland

e-mail: heckmann@cs.uni-sb.de

September 9, 1998

Abstract

Following the program of Moggi, the semantics of a simple non-deterministic functional language with recursion and failure is described by a monad. We show that this monad cannot be any of the known power domain constructions, because they do not handle non-termination properly. Instead, a novel construction is proposed and investigated. It embodies both non-determinism (choice and failure) and possible non-termination caused by recursion.

1 Introduction

Following the proposals of Moggi [Mog89, Mog91b], functional languages with various notions of *computations* can be denotationally described by means of *monads*. Monads are constructions mapping domains of values into domains of computations for these values. Computations involving destructive assignments, for instance, are handled by the *state transformer monad* [Wad90], whereas computations by non-deterministic choice are handled by *power domain constructions* [Plo76, Smy78, Gun90]. All known power domain constructions and many others are covered by an algebraic theory presented in [Hec90, Hec91].

In this paper, we consider a functional language with non-deterministic choice, failure, and recursion. Surprisingly, none of the known power domain constructions — even those versions that support failure — is suitable to describe the semantics of such a language. The reason is that all constructions described by the theory mentioned above are not well-behaved with respect to non-termination caused by recursion.

Non-determinism and recursion are two independent concepts of computation. Standing alone, non-determinism is properly handled by *Plotkin's power domain construction* (the variant with 0), and recursion and non-termination is described by the *lift monad* that adjoins a bottom element. However, in case of a language where both concepts coexist, neither power domain constructions nor lifting nor any simple combination of both provides an appropriate description of the semantics.

The paper is organized as follows: In section 2, we informally discuss several aspects of non-determinism and recursion. In section 3, we briefly introduce the theoretical background: cpo's and continuity in 3.1, monads and strong monads in 3.2, and algebraic theories and free constructions in 3.3.

In section 4, the (well-known) monads are presented that describe recursion and non-determinism independently. They are implicitly characterized as free constructions w.r.t. suitable theories. This suggests to introduce the novel monad that describes both concepts of computation together as free construction w.r.t. a joint theory. In the subsequent sections, we develop several more explicit representations of the new power domains.

In section 5, the novel construction is decomposed into three steps that introduce recursion, choice, and failure in turn. In section 6, the power domains resulting from the novel construction are decomposed into terminating and non-terminating computations. In section 7, we show that the new power domain construction preserves algebraicity, and present a description of the base in the algebraic case.

2 Non-Determinism and Recursion

In this section, we want to discuss informally several aspects of non-determinism and recursion. In 2.1, a non-deterministic functional language is introduced. The monads for non-determinism and recursion are described in 2.2. In section 2.3, we present an informal semantics of our language, which indicates how a formal semantics should look like.

2.1 A Non-Deterministic Functional Language

In [Hec88], we presented a functional language with powerful patterns. They may match in different ways, thus inducing computations with different values. On the other hand, they may fail to match, which leads to a computation resulting in no value at all. An experimental implementation of this language was used to study the effects of non-determinism.

Syntactically, this language is an extension of a simply typed λ-calculus with base types including `int` and `bool`, pair, list, and function types. Non-determinism is introduced implicitly by overlapping or failing patterns, and also explicitly by the choice operator '`|`' and the constant `fail`. The actual interpreter of this language behaves similar to a Prolog interpreter: it tries to show all results of an expression to the user.

Let us consider some sample functions:

```
elem (H :: T)  =  H
elem (H :: T)  =  elem T
```

where the operator '`::`' denotes list construction. If `elem` is applied to the list `[1,2]`, then both rules in the definition of `elem` are applicable. Application of the first rule immediately produces the value 1, whereas the second rule induces the recursive call `elem[2]`. Again, both rules are applicable; the first rule producing 2, and the second rule leading to the recursive call `elem[]`. In this situation, no rule is applicable, whence the execution fails, i.e., terminates without producing a result. Thus, the interpreter will produce the values 1 and 2, and then terminate. More abstractly, it produces the set of values $\{1, 2\}$.

The same operation can be specified with explicit choice:

```
elem (H :: T)  =  H | elem T
```

It is also possible to define this operation without pattern matching:

```
elem L  =  if  empty L  then  fail
                        else  head L | elem (tail L)
```

Consider some further definitions:

```
loop ()   =  loop ()
small N   =  if  N < 5  then  N  else  fail
from  N   =  N | from (N + 1)
nat  ()   =  from 0
```

The expression `fail` immediately terminates without producing any result. Hence, its semantics is $\emptyset$. The expression `loop()` also produces no result, but it does not terminate. In denotational semantics, non-termination is expressed by '$\perp$', whence the semantics of this expression is $\{\perp\}$. The semantics of `small 4` is $\{4\}$, whereas that of `small 6` is $\emptyset$. The semantics of `nat()` is $\{0, 1, 2, \ldots, \perp\}$. We include $\perp$ in this set, because `nat()` does not terminate; it keeps on producing values forever. The expression `small(nat())` produces the values 0 through 4. Then, it does not terminate, but keeps on computing forever without producing further results. The reason is that the argument `nat()` continuously produces numbers that are all refuted by `small`. Thus, the semantics of `small(nat())` is $\{0, 1, 2, 3, 4, \perp\}$.

2.2 Monads for Recursion and Non-determinism

For the denotational semantics, we follow [Mog89, Mog91b] and distinguish between a domain of values and a domain of computations for every type. The semantics is specified by fixing a *monad*, which is a special kind of domain construction. It is used to map domains of values into domains of computations. The domains of values for the base types are $\mathbf{Z} = \{\ldots, -1, 0, 1, 2, \ldots\}$ and $\mathbf{B} = \{\mathsf{F}, \mathsf{T}\}$ respectively. The domains of computations for these types consist of subsets of $\mathbf{Z} \cup \{\perp\}$ and $\mathbf{B} \cup \{\perp\}$. All other domains may be systematically derived from the base domains and the given monad.

As we have seen above, our language has syntactic means for recursive definitions. Recursion inevitably induces the possibility of non-termination. If recursion is the only concept of computation, then an expression either terminates and results in a value, or it does not terminate. Thus, the appropriate monad is the *lift monad* (section 4.1) which adds a new element $\perp$ to every domain of values.

Non-determinism appears as binary choice and failure. By failure, we understand a computation that terminates without producing a value. Many authors [Plo76, HP79, Smy78] study languages with choice but no failure. Our language however has to support failure because of the possibility of failing patterns. Similarly, most process languages [Abr91, page 172] and Prolog [CC84] support failure. We comprise choice and failure to the notion of non-determinism, since both concepts are closely related: they both affect the number of values of a computation, and failure is the neutral element of choice.

By non-determinism without recursion, every expression results in a finite set of values of its type. Thus, the appropriate monad for non-determinism in absence of recursion seems to be the construction of the set of finite subsets. For cpo's, the *Plotkin power construction* [Plo76, HP79] should be used with an added failure element (section 4.2).

If non-determinism and recursion coexist, the situation becomes much more complex. A computation may terminate resulting in a finite number of values, or yield a finite number of values followed by non-termination, or produce an infinite number of values. Unfortunately, there is no simple combination of the two monads mentioned above that would be appropriate in this case (Theorem 5.1). The aim of this paper is to develop an appropriate monad.

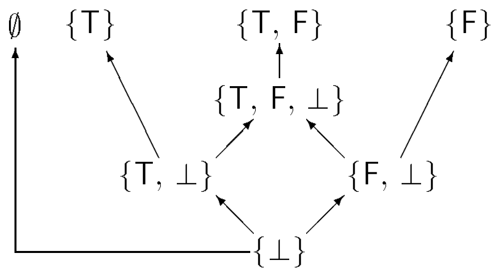
2.3 Observing the Results of an Expression

Let us now consider the language with both recursion and non-determinism in greater detail. Assume we present an expression to the interpreter and observe its results. The interpreter will start to evaluate the expression. Then there are three possibilities: the evaluation does not terminate, the evaluation fails, or the evaluation results in a first value. This value is displayed on the screen, then the evaluation proceeds, and there are again the same three possibilities, etc.

Thus, there are three principal classes of behavior: a finite sequence of values followed by a failure, or a finite sequence of values followed by non-termination, or a potentially infinite sequence of values. For our semantics, we want to abstract from the actual order in which values are presented and from the number of occurrences of a specific value, i.e., we assume that expressions result in sets instead of sequences.

In case of the type `bool` with values T and F, we may observe four terminating sets, namely $\emptyset$, $\{\mathsf{T}\}$, $\{\mathsf{F}\}$, and $\{\mathsf{T}, \mathsf{F}\}$, and four non-terminating sets, namely $\{\bot\}$, $\{\mathsf{T}, \bot\}$, $\{\mathsf{F}, \bot\}$, and $\{\mathsf{T}, \mathsf{F}, \bot\}$. The terminating sets are the abstractions of the finite sequences ending by failure, whereas the non-terminating sets are abstractions of the remaining sequences. The set $\{\mathsf{T}, \mathsf{F}\}$ for instance is produced by the expression `true|false`, whereas $\{\mathsf{T}, \mathsf{F}, \bot\}$ is produced by `isprime(nat())`, where `nat()` produces all natural numbers (cf. section 2.1), and `isprime` is a predicate testing the primality of natural numbers.

An impatient user will probably hit the interrupt key if an evaluation does not produce a value after a while or seems to keep on producing values forever. In this case, he/she/it cannot know whether the evaluation would really be non-terminating if not interrupted. Thus, the behavior $\{\mathsf{T}\}$ looks like $\{\bot\}$ if interrupted before producing anything. The same behavior looks like $\{\mathsf{T}, \bot\}$ if interrupted between the production of T and the final failure. By defining the order $A \leq B$ if B may look like A if prematurely interrupted, we obtain the following poset where arrows denote '$\leq$' relationships:



Power domain experts will notice that none of the known power domain constructions is able to produce this domain from the ground domain $\mathbf{B} = \{\mathsf{T}, \mathsf{F}\}$. The only visible connection to known power domains is that this domain looks like the Plotkin power domain of $\{\bot, \mathsf{T}, \mathsf{F}\}$ with an empty set artificially adjoined above $\{\bot\}$. This informal observation can indeed be generalized. It leads to the description of the power domain construction presented in section 5.

We may also notice that the power domain consists of four terminating sets (depicted in the top row) and four non-terminating sets below them. This observation leads to the vertical decomposition presented in section 6.

3 Theoretical Background

In this section, we introduce the theoretical background for the remainder of the paper. In 3.1, we briefly recall the definition of cpo's and continuous functions. Then we define monads and strong monads in 3.2. Algebraic theories, algebras, and free constructions are introduced in 3.3.

3.1 Cpo's and Continuity

A *poset* (partially ordered set) is a set P together with a reflexive, antisymmetric, and transitive relation '$\leq$'. We often identify the poset $\mathbf{P} = (P, \leq)$ with its carrier P.

For $A \subseteq \mathbf{P}$, let $\downarrow A$ be the set of all points *below* some point of A, and correspondingly $\uparrow A$ the set of all points *above* some point of A. $A \subseteq \mathbf{P}$ is a *lower set* iff $\downarrow A = A$, and an *upper set* iff $\uparrow A = A$.

We refer to the standard notions of upper bound, least upper bound (join) denoted by '$\bigsqcup$', directed set, and monotonic function. A *cpo* is a poset where every directed set has a join. Notice that in this paper, a cpo need not have a least element. A monotonic function $f : \mathbf{X} \to \mathbf{Y}$ between two cpo's is *continuous* iff it preserves the joins of directed sets. $[\mathbf{X} \to \mathbf{Y}]$ is the cpo of all continuous functions from $\mathbf{X}$ to $\mathbf{Y}$ ordered pointwise.

Every set can be made into a cpo by the definition $a \leq b$ iff $a = b$. Every function from such a cpo to an arbitrary cpo is continuous. Thus, the category SET of sets and arbitrary functions may be considered as a full sub-category of CPO.

A category is *Cartesian closed* if it has a terminal object $\mathbf{1}$, product objects $X \times Y$, and function space objects $[X \to Y]$ satisfying certain requirements [Mac71]. Both SET and CPO are Cartesian closed.

In [LS86, AL91], it is shown that the simply typed λ-calculus can be interpreted in any Cartesian closed category so that types denote objects and λ-expressions denote arrows. In the sequel, we identify λ-expressions and arrows, types and objects. For instance, $\lambda x^X .\, x$ is the identity id_X on object X, and $\lambda x^X .\, g(fx)$ is the composition of $f : X \to Y$ and $g : Y \to Z$. The type superscripts at variables will often be dropped. The fact that simply typed λ-calculus can be interpreted in the category CPO then means that every well-typed closed λ-expression that is built from continuous functions is a continuous function again.

3.2 Monads

There are two equivalent definitions of monads: a definition as a so-called Kleisli triple [Mog91b] and a categorical definition in terms of functors and natural transformations [Mac71, Wad90]. Here, we present the Kleisli triple version because it has fewer operations and axioms.

Definition 3.1 A *monad* in a category C is a triple $(\mathcal{M}, \mathsf{r}, \mathsf{E})$ where

- $\mathcal{M}$ maps objects of C into objects of C,
- r is a family of arrows $\mathsf{r}_X : X \to \mathcal{M}X$ for every object X,
- E maps arrows $X \to \mathcal{M}Y$ into arrows $\mathcal{M}X \to \mathcal{M}Y$ for every two objects X and Y,

(E1) $\mathsf{E}f \circ \mathsf{r}_X = f \quad$ for $f : X \to \mathcal{M}Y$,

(E2) $\mathsf{E}\,\mathsf{r}_X = id_{\mathcal{M}X}$,

(E3) $\mathsf{E}g \circ \mathsf{E}f = \mathsf{E}\,(\mathsf{E}g \circ f) \quad$ for $f : X \to \mathcal{M}Y$ and $g : Y \to \mathcal{M}Z$.

The construction $\mathcal{M}$ maps domains of values of some type into domains of computations of this type; r maps values v into a trivial computation $\mathsf{r}v$ that immediately *results* in v; and E *extends* computation-valued functions defined on values to those defined on computations. An extended function $\mathsf{E}f$ applied to some computation will execute this computation and apply f to the resulting value(s).

As pointed out in [Mog89, Mog91b], one needs *strong monads* to model computations in denotational semantics. They are characterized by an additional operation t with five axioms. For Cartesian closed categories, strength of a monad is equivalent to *internality* of E. Internality means E is not just a family of functions between hom-sets, but a family of arrows of second order type $[X \to \mathcal{M}Y] \to [\mathcal{M}X \to \mathcal{M}Y]$. In the category CPO, this means E has to be continuous, whereas in the category SET, it means just nothing, whence all monads in SET are strong.

3.3 Algebraic Theories and Free Constructions

Most of our paper does not deal with monads directly, but with the closely related free constructions w.r.t. algebraic theories.

Usually, an algebraic theory consists of a set of operators with given arity and a set of axioms in form of equations $L = R$ over these operators. Since we want to consider algebraic theories in the category CPO, we also allow inequations $L \leq R$ as axioms.

A *model* of an algebraic theory $\mathcal{T}$ in the category CPO, or shortly a *$\mathcal{T}$-algebra*, is a cpo — the *carrier* — together with a set of *continuous* functions — the *operations* — interpreting the operators and satisfying the axioms. The functions interpreting operators of arity n have type $X^n \to X$ where X is the carrier. A $\mathcal{T}$-algebra homomorphism, or shortly *$\mathcal{T}$-morphism*, is a *continuous* function between (the carriers of) $\mathcal{T}$-algebras preserving all operations.

The *empty theory* $\mathcal{E}$ has neither operators nor axioms, whence $\mathcal{E}$-algebras are just cpo's and $\mathcal{E}$-morphisms are just continuous functions. A theory $\mathcal{S}$ is a subtheory of $\mathcal{T}$ if the set of operators of $\mathcal{S}$ is a subset of that of $\mathcal{T}$, and same for the set of axioms. In this case, all $\mathcal{T}$-algebras are $\mathcal{S}$-algebras, too, and all $\mathcal{T}$-morphisms are $\mathcal{S}$-morphisms, too.

Definition 3.2 (Free constructions) A *free construction* $\mathcal{C} = (\mathcal{C}, c, C)$ from a subtheory $\mathcal{S}$ to a theory $\mathcal{T}$ maps every $\mathcal{S}$-algebra X into a $\mathcal{T}$-algebra $\mathcal{C}X$ by adding the missing operations such that there is a $\mathcal{S}$-morphism $c : X \to \mathcal{C}X$, and for every $\mathcal{T}$-algebra A and every $\mathcal{S}$-morphism $f : X \to A$, there is a unique $\mathcal{T}$-morphism $Cf : \mathcal{C}X \to A$ extending f, i.e., $Cf \circ c = f$. $\mathcal{C}X$ is called the *free $\mathcal{T}$-algebra* over X. The elements cx with x in X are its *generators*.

If the subtheory $\mathcal{S}$ is the empty theory, we shall not explicitly mention it. In this paper, we shall adopt the notational convention to denote the three constituents of a free construction by the same letter in different styles, e.g., $(\mathcal{B}, b, B)$ or $(\mathcal{P}^0, p^0, P^0)$. As short names for these free constructions, $\mathcal{B}$ and $\mathcal{P}^0$ will be used.

Categorically speaking, free constructions from $\mathcal{S}$ to $\mathcal{T}$ are left adjoint to the forgetful functor from $\mathcal{T}$-algebras to $\mathcal{S}$-algebras. Free constructions are uniquely determined up to isomorphism.

In the sequel, we show some properties of free constructions. First of all, they compose:

Proposition 3.3 (Composition of free constructions)
If $\mathcal{C} = (\mathcal{C}, c, C)$ is a free construction from $\mathcal{T}_1$ to $\mathcal{T}_2$, and $\mathcal{D} = (\mathcal{D}, d, D)$ is free from $\mathcal{T}_2$ to $\mathcal{T}_3$, where $\mathcal{T}_1 \subseteq \mathcal{T}_2 \subseteq \mathcal{T}_3$, then $(\mathcal{D} \circ \mathcal{C}, d \circ c, D \circ C)$ is free from $\mathcal{T}_1$ to $\mathcal{T}_3$.

The proof is straightforward. It can be found in [Mac71] for the more general case of adjoints.

Next, we show that free algebras do not have proper sub-algebras containing all generators. Categorically, a *sub-algebra* B of a $\mathcal{T}$-algebra A is a $\mathcal{T}$-algebra such that there is a monic $\mathcal{T}$-morphism $e : B \to A$. Concretely, it is a subset of A closed w.r.t. all operations of $\mathcal{T}$ and directed joins (since we assume cpo's as carriers).

Lemma 3.4 Let $(\mathcal{C},\, c,\, C)$ be a free construction from $\mathcal{S}$ to $\mathcal{T}$, and let X be an $\mathcal{S}$-algebra. Let B be a subset of $\mathcal{C}X$ that is closed w.r.t. all operations of $\mathcal{T}$ and directed joins, and that contains all generators cx. Then B equals $\mathcal{C}X$.

Proof: Since B is closed w.r.t. all $\mathcal{T}$-operations and directed joins, it forms a $\mathcal{T}$-algebra, and the inclusion $i : B \to \mathcal{C}X$ is a $\mathcal{T}$-morphism. Since B contains all generators cx, the $\mathcal{S}$-morphism $c : X \to \mathcal{C}X$ may be co-restricted to a $\mathcal{S}$-morphism $c' : X \to B$ such that $i \circ c' = c$. By freedom of $\mathcal{C}$, this $\mathcal{S}$-morphism can be (uniquely) extended to a $\mathcal{T}$-morphism $Cc' : \mathcal{C}X \to B$. Then, $i \circ Cc'$ is a $\mathcal{T}$-morphism from $\mathcal{C}X$ to itself. $i \circ Cc' \circ c = id \circ c$ holds, whence by the uniqueness statement of freedom, $i \circ Cc' = id$ follows. Thus, i is an isomorphism. □

Every free construction $\mathcal{C}$ from the empty theory induces a monad given by the same triple. Axiom (E1) is immediate, whereas (E2) and (E3) follow from the uniqueness statement of freedom. Using the Cartesian closed structure of CPO, we can show that this monad is strong.

Proposition 3.5 Every free construction to a theory $\mathcal{T}$ induces a strong monad in CPO.

Proof: Since we are in a Cartesian closed category, we use λ-calculus for the proof.

The proof uses the following observation: for every cpo X and $\mathcal{T}$-algebra A, the function space $[X \to A]$ becomes a $\mathcal{T}$-algebra, if the operations op_A on A are raised to operations $op_{[X\to A]}$ on functions by defining $op_{[X\to A]}(f_1, \ldots, f_n) = \lambda x^X.\, op_A(fx_1, \ldots, fx_n)$.

Let the free construction be $(\mathcal{C},\, c,\, \overline{})$, i.e., for this proof, we denote the extension of f by $\overline{f}$. Let X and Y be two fixed cpo's. We have to show that the function that maps f into $\overline{f}$ is continuous. We do so by specifying a closed λ-expression C with $Cf = \overline{f}$ for all functions $f : X \to \mathcal{M}Y$. The simple solution $C \equiv \lambda f.\, \overline{f}$ is illegal, unless we know that extension is continuous. We only know that $\overline{f}$ is continuous whenever f is continuous. Thus, the expression f in $\overline{f}$ must be a *closed* λ-expression, which denotes a continuous function, not a variable.

Consider the closed λ-expression $a \equiv \lambda x^X.\, \lambda f^{[X\to\mathcal{C}Y]}.\, fx$ with type $X \to [[X \to \mathcal{C}Y] \to \mathcal{C}Y]$. By the observation above, $[[X \to \mathcal{C}Y] \to \mathcal{C}Y]$ is a $\mathcal{T}$-algebra since $\mathcal{C}Y$ is a $\mathcal{T}$-algebra. Thus, a has a continuous extension $\overline{a} : \mathcal{C}X \to [[X \to \mathcal{C}Y] \to \mathcal{C}Y]$. By rearranging arguments, we obtain $C \equiv \lambda f^{[X\to\mathcal{C}Y]}.\, \lambda u^{\mathcal{C}X}.\, \overline{a}\, u f : [X \to \mathcal{C}Y] \to [\mathcal{C}X \to \mathcal{C}Y]$. Then, we have to show $Cf = \overline{f}$ for all $f : X \to Y$. This is done by the uniqueness assertion of freedom. The right hand side is a $\mathcal{T}$-morphism by definition of $\overline{f}$. The left hand side is a $\mathcal{T}$-morphism since for every operation $op_{\mathcal{C}X}$,

$$\begin{aligned} Cf\,(op_{\mathcal{C}X}(x_1, \ldots, x_n)) &= \overline{a}\,(op_{\mathcal{C}X}(x_1, \ldots, x_n))f \\ &= op_{[[X\to\mathcal{C}Y]\to\mathcal{C}Y]}(\overline{a}\,x_1, \ldots, \overline{a}\,x_n)f \\ &= op_{\mathcal{C}Y}(\overline{a}\,x_1 f, \ldots, \overline{a}\,x_n f) \\ &= op_{\mathcal{C}Y}(Cf\,x_1, \ldots, Cf\,x_n) \end{aligned}$$

Here, we used the fact that $\overline{a}$ is a $\mathcal{T}$-morphism as the extension of a. Next, we show that Cf and $\overline{f}$ coincide on generators. $\overline{f}\,(cx)$ yields fx, and

$$Cf\,(cx) \;=\; \overline{a}\,(cx)\,f \;=\; a\,x\,f \;=\; fx$$

shows that $Cf\,(cx)$ yields fx, too. □

4 The Monads for Recursion and Non-determinism

In this section, we introduce the well-known monads that model recursion and non-determinism independently. Their implicit characterizations as free constructions w.r.t. some theory suggest the definition of a monad for the combination of both concepts as free construction for a combined theory.

4.1 Recursion

To model recursion, the *fixed point theorem* is employed: every continuous function in a cpo with least element has a unique least fixed point. Thus, our domains of computations should be cpo's with least element $\perp$. This special element denotes non-termination.

As pointed out informally in section 2, recursive computations either result in a proper value or in $\perp$. Hence, this notion of computation can be modeled by the *lift monad* that adds a new least element to each cpo. Formally, let

$\mathcal{B}X \;=\; X_\perp \;=\; \{\perp\} \cup \{bx \mid x \in X\}$

ordered by $\perp \leq y$ for all y in $X_\perp$, and $bx \leq bx'$ iff $x \leq x'$. Extension is given by $Bf\perp = \perp$ and $Bf(bx) = fx$. One easily proves that everything is continuous and the monad axioms are satisfied.

The lift monad can also be characterized as the free construction for the $\perp$-theory with one nullary operation $\perp$ and just one axiom $\perp \leq x$ for all x. $\perp$-algebras are then cpo's with least element, and $\perp$-morphisms are strict continuous functions.

4.2 Non-determinism

At first, we want to model non-determinism in the absence of recursion. As described in section 2, non-determinism is introduced in our language by constants for failure and binary choice. Thus, we need domains of computation that contain a special element 0 representing failure and a binary operation '+' representing choice. Since we want to abstract from order and multiplicity of results, '+' should be commutative, associative, and idempotent with neutral element 0. This is the algebraic theory of *commutative idempotent monoids*, or *CI monoids* for short. The morphisms of this theory are the continuous functions f with $f0 = 0$ and $f\,(a+b) = fa + fb$. We call such functions 0-*additive*. The closely related theory of CI semigroups has no operator 0. Its morphisms are called *additive*.

From [HP79], it is known that Plotkin's power construction — defined in [Plo76] for algebraic cpo's — is a free construction $\mathcal{P}$ for CI semigroups. In [Hoo87, Hec91], this construction is shown to exist in the category of all cpo's. This suggests to define the monad for non-determinism with failure and choice as the free construction $\mathcal{P}^0$ for CI monoids. It can be built by composing $\mathcal{P}$ and the free construction $\mathcal{N}$ from CI semigroups to CI monoids.

Proposition 4.1 The free construction $\mathcal{N}$ from CI semigroups to CI monoids is given by

Elements :	$\mathcal{N}Y = \{ny \mid y \in Y\} \;\cup$	$\{0\}$
Order :	$ny \leq ny'$ if $y \leq y'$	$0 \leq 0$
Operations :	$na + nb = n\,(a+b)$	$z + 0 = 0 + z = z$
Extension :	$Nf\,(ny) = fy$	$Nf\,0 = 0$

Thus, for all cpo's X, $\mathcal{P}^0X$ results from $\mathcal{P}X$ by adding a new element 0 incomparable to every other element.

Proof: By case analysis, $\mathcal{N}Y$ is shown to be a CI monoid. Nf as defined above is obviously the only extension of f that preserves 0. Its additivity is shown by case analysis. $\square$

Since 0 is incomparable to every other element, $\mathcal{P}^0 X$ has no least element. This is usually considered a drawback, but notice that there is no need of a least element in the semantic domains in the absence of recursion.

Applied to a set S, the construction $\mathcal{P}^0$ yields the set $\mathcal{P}_{\mathit{fin}} S$ of finite subsets of S. This claim can be proved by showing that $\mathcal{P}_{\mathit{fin}} S$ with binary union and empty set forms the free CI monoid over S. p^0 is the singleton mapping $\lambda x.\{x\}$. Extension P^0 maps a set-valued function defined on elements into a set-valued function defined on sets by $P^0 f A = \bigcup_{a \in A} f a$. By $P^0 f\{x\} = f x$, it is obvious that $P^0 f$ properly extends f.

Remember that our two base types `int` and `bool` are interpreted by cpo's being sets. Because the subcategory SET of CPO is closed w.r.t. product, function space, and the monad $\mathcal{P}^0$ in form of $\mathcal{P}_{\mathit{fin}}$, we never leave this subcategory in modeling non-determinism in the absence of recursion. Thus, the complexity of cpo's and continuous functions is not needed in this case. This corresponds to our informal reasoning in section 2, where we claimed that without recursion, non-deterministic expressions always terminate and yield a finite set of values.

4.3 Recursion and Non-determinism Together

In the two previous sections, we saw how non-determinism and recursion can be independently modeled as free constructions w.r.t. the theory of CI monoids and the $\perp$-theory. These results suggest to model their combined effect by a free construction w.r.t. the combined theory of *CI$\perp$-monoids*. This theory is given by the following table:

Operators :	$0 : X$	$+ : X \times X \to X$	$\perp : X$
Axioms :	$a + 0 = a$	$a + (b + c) = (a + b) + c$	$\perp \leq a$
	$a + a = a$	$a + b = b + a$	
Morphisms :	$f(0) = 0$	$f(a + b) = fa + fb$	$f(\perp) = \perp$

The similar theory of *CI$\perp$-semigroups* has no operator 0, and thus no axiom $a + 0 = a$ and no requirement $f(0) = 0$.

The development presented in [Hoo87, Hec91] suffices to show the existence of the free construction $\mathcal{P}^0_\perp$ w.r.t. this theory. This implicit description of $\mathcal{P}^0_\perp$ is not completely satisfactory, because it neither shows the internal structure of the $\mathcal{P}^0_\perp$-powerdomains nor clarifies the relationships to the monads of lifting and $\mathcal{P}^0$. Thus, we develop more explicit representations of $\mathcal{P}^0_\perp X$ in the subsequent sections. With these representations, one can show that $\mathcal{P}^0_\perp\{\mathsf{F}, \mathsf{T}\}$ is indeed the domain depicted in section 2.3 as the result of some informal reasoning.

5 Decomposition into three Free Constructions

In this section, we develop a description of $\mathcal{P}^0_\perp$ by decomposing it into three free constructions to be applied sequentially. It is suggested by the observation that the domain depicted in section 2.3 looks like the Plotkin power domain of $\{\perp, \mathsf{F}, \mathsf{T}\}$ with an empty set adjoined above the least element.

Theorem 5.1 The construction $\mathcal{P}^0_\perp$ can be decomposed into three free constructions to be applied sequentially:

(1) The free construction $\mathcal{B}$ from the empty theory to the $\perp$-theory, which is given by lifting;

(2) The free construction $\mathcal{P}_\perp$ from the $\perp$-theory to the theory of CI$\perp$-semigroups, which is given by $\mathcal{P}$;

(3) The free construction $\mathcal{N}_\perp$ from CI$\perp$-semigroups to CI$\perp$-monoids, which is given by:

Elements :	$\mathcal{N}_\perp Y = \{n_\perp y \mid y \in Y\} \quad \cup$	$\{0\}$
Order :	$n_\perp y \le n_\perp y'$ if $y \le y'$	$n_\perp \perp \le 0 \qquad 0 \le 0$
Operations :	$n_\perp a + n_\perp b = n_\perp(a+b)$	$z + 0 = 0 + z = z$
Extension :	$N_\perp f(n_\perp y) = fy$	$N_\perp f\, 0 = 0$

The theorem tells that $\mathcal{P}^0_\perp$ is *not* the composition of the lift monad $\mathcal{B}$ and the monad $\mathcal{P}^0 = \mathcal{N} \circ \mathcal{P}$ modeling choice and failure. Instead of $\mathcal{N} \circ \mathcal{P} \circ \mathcal{B}$, it is $\mathcal{N}_\perp \circ \mathcal{P} \circ \mathcal{B}$ where $\mathcal{N}_\perp$ differs from $\mathcal{N}$ by the additional relationship $n_\perp \perp \le 0$.

Proof: Construction (1) was already discussed in section 4.1. For (2), we have to show that the free construction $\mathcal{P}$ from the empty theory to that of CI semigroups is also free from the $\perp$-theory to that of CI$\perp$-semigroups, i.e., $\mathcal{P}$ maps cpo's with bottom into cpo's with bottom, p is strict, and Pf is strict whenever f is strict.

Assume that X has a least element $\perp$. We employ Lemma 3.4 to show that $p\perp$ is the least element of $\mathcal{P}X$. Let $S = \uparrow\{p\perp\}$ be the upper cone of $p\perp$ in $\mathcal{P}X$. Obviously, S is closed w.r.t. directed joins. It is closed w.r.t. addition, since $a, b \ge p\perp$ implies $a + b \ge p\perp + p\perp = p\perp$ by monotonicity and idempotence of addition. It contains all generators because $px \ge p\perp$ by monotonicity of p. Thus, $S = \mathcal{P}X$ follows, whence $\mathcal{P}X$ has a least element and p is strict.

Let X and Y be cpo's with least element, and let $f : X \to \mathcal{P}Y$ be strict and continuous. Then $Pf : \mathcal{P}X \to \mathcal{P}Y$ is strict by $Pf(\perp_{\mathcal{P}X}) = Pf(p\perp_X) = f(\perp_X) = \perp_{\mathcal{P}Y}$.

For construction (3), we have to show that the given construction $\mathcal{N}_\perp$ is free from CI$\perp$-semigroups to CI$\perp$-monoids. This is similar to the proof of Prop. 4.1, where $\mathcal{N}$ is shown to be free from CI-semigroups to CI-monoids. This concludes the proof of Theorem 5.1. □

The decomposition presented in Theorem 5.1 may be used to describe the structure of the power domains a bit more explicitly.

Corollary 5.2 For every cpo X, the power domain $\mathcal{P}^0_\perp X$ results from adjoining a new element 0 to $\mathcal{P}(X_\perp)$ above $p\perp$:

Elements : $\mathcal{P}(X_\perp) \cup \{0\}$

Order : $A \le B$ iff $A \le B$ in $\mathcal{P}(X_\perp)$ or ($A = p\perp$ and $B = 0$)

Employing coalesced sum '$\oplus$', one may also write $\mathcal{P}^0_\perp X = \{0\}_\perp \oplus \mathcal{P}(X_\perp)$.

The representation of $\mathcal{P}^0_\perp X$ given above is closely related to the power construction in [Abr91, page 173], which is defined as $\mathcal{P}_{Abr} = \mathcal{N}_\perp \circ \mathcal{P}$. By comparison with $\mathcal{P}^0_\perp = \mathcal{N}_\perp \circ \mathcal{P} \circ \mathcal{B}$, the relation $\mathcal{P}^0_\perp = \mathcal{P}_{Abr} \circ \mathcal{B}$ follows, whence we obtain $\mathcal{P}^0_\perp X = \mathcal{P}_{Abr}(X_\perp)$ for all domains X. Thus, applying our power domain to X means applying Abramsky's power domain to the lift $X_\perp$. This difference has important implications for the algebraic properties: $\mathcal{P}_{Abr}$ cannot properly extend non-strict functions, whence it is not a monad in the category CPO. It is a monad in the category of cpo's with least element and continuous strict functions, however, because it is a free construction from cpo's with $\perp$ to CI$\perp$-monoids as shown in the proof of Theorem 5.1. In [Abr91], this algebraic property is not explicitly mentioned.

6 Vertical Decomposition

The decomposition of $\mathcal{P}^0_\perp$ in Theorem 5.1 suggests that the failure element 0 be something special. It seems to be added artificially by the third component of the construction. In this section, we present a 'vertical decomposition' of the power domains into terminating sets, which can be found in the upper parts of the power domain, and non-terminating sets, which are located in the lower parts. In this decomposition, 0 does not play a special role.

In 6.1, we introduce an auxiliary algebraic theory. The decomposition theorem is stated and proved in 6.2. In 6.3, it is applied to the case of discrete domains.

6.1 Lower Monoids

In order to state and prove the decomposition, we have to introduce another algebraic theory. A *lower monoid* is a monoid where the neutral element is least: $0 \leq a$ for all a. Equivalently, lower monoids may be described by the axiom $a \leq a + b$. Lower CI monoids are CI$\perp$-monoids with $0 = \perp$. In the sequel, we show that every CI$\perp$-monoid contains a lower CI monoid with canonical mappings back and forth.

Let Y be a CI$\perp$-monoid. The two sets $Y' = \{\perp + y \mid y \in Y\}$ and $\{y \in Y \mid \perp + y = y\}$ coincide by idempotence. Y' is closed w.r.t. directed joins (consider the second representation). Thus, it is a sub-cpo of Y. It is closed w.r.t. addition of Y by $(\perp + y_1) + (\perp + y_2) = \perp + (y_1 + y_2)$. It contains $\perp = \perp + 0$. This is the neutral element of addition *in* Y', because $\perp + (\perp + y) = \perp + y$ by idempotence. Thus, $(Y', +, \perp)$ forms a lower monoid.

Let the function $\iota : Y' \to Y$ be induced by the subset inclusion of Y' into Y. Because the subset Y' is closed w.r.t. addition, $\perp$, and directed joins of Y, the function ι is additive, strict, and continuous. In general, it does not preserve 0. By the calculations in the previous paragraph, the function $\lambda y^Y.\perp + y$ induces a strict, 0-additive, continuous function $\alpha : Y \to Y'$. The composition $\alpha \circ \iota$ is the identity on Y' because of the representation $Y' = \{y \in Y \mid \perp + y = y\}$ of the subset Y'. Summarizing, we have proved the following:

Proposition 6.1 For every CI$\perp$-monoid Y, there is a lower CI monoid Y' with:

- $\alpha : Y \to Y'$ strict, 0-additive, and continuous,
- $\iota : Y' \to Y$ strict, additive, and continuous,
- $\alpha \circ \iota \ = \ id : \ Y' \to Y'$,
- $\iota \circ \alpha \ = \ \lambda y^Y.\perp + y \ \leq \ id : \ Y \to Y$.

One may further show that $(Y \mapsto Y', \alpha, f \mapsto f \circ \iota)$ is a free construction from CI$\perp$-monoids to lower CI monoids. We do not need this additional property in this paper.

6.2 The Vertical Decomposition Theorem

For the decomposition theorem, we need some more notation. As in section 4.2, let $\mathcal{P}^0$ be the free construction to CI monoids. The free construction to lower CI monoids is denoted by $\mathcal{L}^0$. Its variant without 0 is known as *lower* or *Hoare power domain construction.* $\mathcal{L}^0$ has an explicit representation in terms of the cpo of *Scott closed subsets* of the ground domain ordered by inclusion. Scott closed sets are lower sets that are closed w.r.t. directed joins.

The mapping $l^0 : X \to \mathcal{L}^0 X$ is continuous, and $\mathcal{L}^0 X$ is a CI monoid. Thus, there is a unique 0-additive extension $\ell : \mathcal{P}^0 X \to \mathcal{L}^0 X$ with $\ell \circ p^0 = l^0$. We use this canonical mapping to compare members of $\mathcal{P}^0 X$ and $\mathcal{L}^0 X$ in the decomposition theorem. If X is a bifinite domain, then $\mathcal{P}^0 X$ is explicitly given by the set of convex Lawson closed subsets of X [Law88]. In this case, ℓ is concretely given by lower closure: $\ell A = \downarrow A$.

The decomposition theorem states that $\mathcal{P}^0_\perp X$ consists of an instance of $\mathcal{P}^0 X$ containing the terminating sets, and an instance of $\mathcal{L}^0 X$ containing the non-terminating sets. Two formal symbols, P and L, are used to label the members of the two components.

Theorem 6.2 For every domain X, the power domain $\mathcal{P}^0_\perp X$ is given by

Elements :	$\mathcal{P}^0_\perp X \ = \ \{\mathsf{L}A \mid A \in \mathcal{L}^0 X\} \ \cup$	$\{\mathsf{P}A \mid A \in \mathcal{P}^0 X\}$
Order :	$\mathsf{L}A \leq \mathsf{L}B$ iff $A \leq B$	$\mathsf{P}A \leq \mathsf{P}B$ iff $A \leq B$
	$\mathsf{L}A \leq \mathsf{P}B$ iff $A \leq \ell B$	$\mathsf{P}A \leq \mathsf{L}B$ never holds
Constants :	$\perp = \mathsf{L}0$	$0 = \mathsf{P}0$
Addition :	$\mathsf{L}A + \mathsf{L}B = \mathsf{L}\,(A+B)$	$\mathsf{P}A + \mathsf{P}B = \mathsf{P}\,(A+B)$
	$\mathsf{L}A + \mathsf{P}B = \mathsf{L}\,(A + \ell B)$	$\mathsf{P}A + \mathsf{L}B = \mathsf{L}\,(\ell A + B)$
Generators :	$p^0_\perp x = \mathsf{P}\,(p^0 x)$	

We use the concepts introduced by Prop. 6.1 to define $P^0_\perp$. For $f : X \to Y$, the extension $P^0_\perp f : \mathcal{P}^0_\perp X \to Y$ is given by

$$P^0_\perp f\,(\mathsf{P}A) = P^0 f A \qquad P^0_\perp f\,(\mathsf{L}A) = \iota\,(L^0\,(\alpha \circ f)\,A)$$

Proof: We first show that $\mathcal{P}^0_\perp X$ as described above is a CI$\perp$-monoid. It is obviously a cpo. The least element is $\mathsf{L}0$ since 0 is least in $\mathcal{L}^0 X$. Commutativity and idempotence of addition are inherited from $\mathcal{P}^0 X$ and $\mathcal{L}^0 X$. The neutral element is $\mathsf{P}0$, because

- $\mathsf{P}0 + \mathsf{P}A = \mathsf{P}\,(0 + A) = \mathsf{P}A$
- $\mathsf{P}0 + \mathsf{L}A = \mathsf{L}\,(\ell 0 + A) = \mathsf{L}\,(0 + A) = \mathsf{L}A$

using the fact that ℓ maps the member 0 of $\mathcal{P}^0 X$ into 0 of $\mathcal{L}^0 X$ by 0-additivity. Associativity of addition is shown by analyzing the obvious 8 cases for choosing 3 elements from $\mathcal{P}^0_\perp X$. For some cases, additivity of ℓ is needed, e.g.,

$$\begin{aligned}&\mathsf{L}A + (\mathsf{P}B + \mathsf{P}C) \ = \ \mathsf{L}A + \mathsf{P}\,(B + C) \ = \ \mathsf{L}\,(A + \ell\,(B + C)) \ = \\ &\mathsf{L}\,(A + \ell B + \ell C) \ = \ \mathsf{L}\,(A + \ell B) + \mathsf{P}C \ = \ (\mathsf{L}A + \mathsf{P}B) + \mathsf{P}C\end{aligned}$$

Now, let f be a continuous function from X to some CI$\perp$-monoid Y. Then $\alpha \circ f : X \to Y'$, whence both $P^0 f : \mathcal{P}^0 X \to Y$ and $L^0\,(\alpha \circ f) : \mathcal{L}^0 X \to Y' \subseteq Y$ are well-defined and continuous. Thus, their combination $P^0_\perp f : \mathcal{P}^0_\perp X \to Y$ is continuous if we succeed in showing that it is monotonic across the border between $\mathcal{L}^0 X$ and $\mathcal{P}^0 X$. We need an auxiliary lemma for this:

- $L^0\,(\alpha \circ f) \circ \ell = \alpha \circ P^0 f : \ \mathcal{P}^0 X \to Y'$

The lemma is shown by freedom of $\mathcal{P}^0$. By definition, ℓ is 0-additive from $\mathcal{P}^0 X$ to $\mathcal{L}^0 X$, and $L^0\,(\alpha \circ f)$ is 0-additive from $\mathcal{L}^0 X$ to Y'. Similarly, $P^0 f$ is 0-additive from $\mathcal{P}^0 X$ to Y, and α is 0-additive from Y to Y' by Prop. 6.1. Thus, both sides are 0-additive. They coincide on the generators:

$$\begin{aligned}&\alpha \circ P^0 f \circ p^0 \ = \ \alpha \circ f \\ &L^0\,(\alpha \circ f) \circ \ell \circ p^0 \ = \ L^0\,(\alpha \circ f) \circ l^0 \ = \ \alpha \circ f\end{aligned}$$

This concludes the proof of the auxiliary lemma.

Next, we show that $P^0_\perp f$ is monotonic and properly acts on the operations and generators. We abbreviate $\alpha \circ f$ by f'.

$$\begin{aligned}&\leq: \quad A \leq \ell B \Rightarrow P^0_\perp f\,(\mathsf{L}A) \ = \ \iota\,(L^0 f' A) \ \leq \ \iota\,(L^0 f'\,(\ell B)) \ \stackrel{!}{=} \ \iota\,(\alpha\,(P^0 f B)) \ \leq \ P^0_\perp f\,(\mathsf{P}B) \\ &\perp: \quad P^0_\perp f\,(\mathsf{L}0) \ = \ \iota\,(L^0 f'\, 0_{\mathcal{L}^0 X}) \ = \ \iota\, 0_{Y'} \ = \ \perp_Y \\ &0: \quad P^0_\perp f\,(\mathsf{P}0) \ = \ P^0 f\, 0_{\mathcal{P}^0 X} \ = \ 0_Y \\ &p^0_\perp: \quad P^0_\perp f\,(\mathsf{P}\,(p^0 x)) \ = \ P^0 f\,(p^0 x) \ = \ f x\end{aligned}$$

At the place marked by '!', the auxiliary lemma was used. The remaining cases of '$\le$' are obvious. For additivity, we consider three representative cases. The fourth case follows from commutativity.

$$\begin{aligned}
P^0_\perp f\,(\mathsf{P}A + \mathsf{P}B) \;&=\; P^0 f\,(A+B) \;= \\
&\quad P^0 fA + P^0 fB \;=\; P^0_\perp f\,(\mathsf{P}A) + P^0_\perp f\,(\mathsf{P}B) \\
P^0_\perp f\,(\mathsf{L}A + \mathsf{L}B) \;&=\; \iota\,(L^0 f'\,(A+B)) \;= \\
&\quad \iota\,(L^0 f'A) + \iota\,(L^0 f'B) \;=\; P^0_\perp f\,(\mathsf{L}A) + P^0_\perp f\,(\mathsf{L}B) \\
P^0_\perp f\,(\mathsf{L}A + \mathsf{P}B) \;&=\; \iota\,(L^0 f'\,(A+\ell B)) \;=\; \iota\,(L^0 f'A) + \iota\,(L^0 f'\,(\ell B)) \;\stackrel{!}{=} \\
&\quad \iota\,(L^0 f'A) + \perp + P^0 fB \;=\; P^0_\perp f\,(\mathsf{L}A) + P^0_\perp f\,(\mathsf{P}B)
\end{aligned}$$

At the place marked by '!', the auxiliary lemma was used. $\iota\,(L^0 f'A)$ absorbs $\perp$ because it is in the subset Y'.

Finally, we have to show the uniqueness of extended functions. Let $F : \mathcal{P}^0_\perp X \to Y$ be a strict 0-additive continuous function with $F \circ p^0_\perp = f$. We have to show $F = P^0_\perp f$, i.e., $F \circ \mathsf{P} = P^0 f$ and $F \circ \mathsf{L} = \iota \circ L^0\,(\alpha \circ f)$.

Function $F \circ \mathsf{P}$ maps from $\mathcal{P}^0 X$ to Y. It is 0-additive since P is so by definition of 0 and '+' in $\mathcal{P}^0_\perp X$, and F is 0-additive by assumption. $F \circ \mathsf{P} \circ p^0 = F \circ p^0_\perp = f$ holds by definition of $p^0_\perp$. By freedom of $\mathcal{P}^0$, the claimed equation $F \circ \mathsf{P} = P^0 f$ follows.

Function $\alpha \circ F \circ \mathsf{L}$ maps from $\mathcal{L}^0 X$ to Y'. It is additive and strict as a composition of the additive strict functions L, F, and α. Additive strict functions between lower monoids are 0-additive. Once $\alpha \circ F \circ \mathsf{L} \circ l^0 = \alpha \circ f$ is shown, the equality $\alpha \circ F \circ \mathsf{L} = L^0\,(\alpha \circ f)$ follows from freedom of $\mathcal{L}^0$.

$$\begin{aligned}
\alpha\,(F\,(\mathsf{L}\,(l^0 x))) \;&=\; \alpha\,(F\,(\mathsf{L}\,(\ell\,(p^0 x)))) \\
&=\; \alpha\,(F\,(\mathsf{L}\,(0 + \ell\,(p^0 x)))) \\
&=\; \alpha\,(F\,(\mathsf{L}0 + \mathsf{P}\,(p^0 x))) \\
&=\; \alpha\,(F\,(\mathsf{L}0)) + \alpha\,(F\,(\mathsf{P}\,(p^0 x))) \\
&=\; 0_{Y'} + \alpha\,(F\,(p^0_\perp x)) \\
&=\; \alpha\,(fx)
\end{aligned}$$

holds using additivity of F.

By composing with ι, we obtain $\iota \circ \alpha \circ F \circ \mathsf{L} = \iota \circ L^0\,(\alpha \circ f)$. By additivity of F, we can show that images of $F \circ \mathsf{L}$ absorb $\perp$:

$$\perp_Y + F\,(\mathsf{L}A) = F\,(\mathsf{L}0) + F\,(\mathsf{L}A) = F\,(\mathsf{L}A)$$

Thus, $\iota \circ \alpha \circ F \circ \mathsf{L} = F \circ \mathsf{L}$ follows, whence $F \circ \mathsf{L} = \iota \circ L^0\,(\alpha \circ f)$ as required.

This concludes the proof of Theorem 6.2. □

6.3 The Discrete Case

Theorem 6.2 can be used to compute the power domains $\mathcal{P}^0_\perp X$ for discrete ground domains X. A cpo is *discrete* iff its order is equality: $a \le b$ iff $a = b$. The base types of our language are modeled by such discrete domains. It is easy to see that the discrete cpo's form a full sub-category of CPO that is isomorphic to SET.

To apply Theorem 6.2, we first have to determine $\mathcal{L}^0 X$ and $\mathcal{P}^0 X$. These power domains are given by the following table:

	$\mathcal{L}^0 X$	$\mathcal{P}^0 X$
Theory :	lower CI monoids	CI monoids
Elements :	all subsets of X	finite subsets of X
Order :	'$\subseteq$'	'$=$'
Constants :	$\perp = 0 = \emptyset$	$0 = \emptyset$
Addition :	union	union
Generators :	$\{x\}$	$\{x\}$

The table can either be derived from the general knowledge about power domain constructions [HP79, Hoo87, Hec90], or it can easily be verified directly. The infinite subsets of X are included in $\mathcal{L}^0 X$, since they can be obtained as directed joins from the finite subsets provided the ordering is subset inclusion. The function ℓ from $\mathcal{P}^0 X$ to $\mathcal{L}^0 X$ operates as an identity on sets.

By Theorem 6.2, the power domain $\mathcal{P}^0_\perp X$ consists of the elements $\mathsf{L}A$ where $A \subseteq X$, and $\mathsf{P}B$ where $B \subseteq_f X$. The situation can be simplified if $\mathsf{P}B$ is replaced by B itself, and $\mathsf{L}A$ is represented by $A \cup \{\perp\}$. Then $\mathcal{P}^0_\perp X$ consists of those subsets of $X \cup \{\perp\}$ that contain $\perp$ or are finite. The proposed representation has the additional advantage that addition in $\mathcal{P}^0_\perp X$ is given by set union. This can be shown by looking at the clauses for addition in Theorem 6.2.

After completely translating Theorem 6.2 into the new representation, we obtain:

Theorem 6.3 For every discrete domain X, the power domain $\mathcal{P}^0_\perp X$ is given by

Elements : $\mathcal{P}^0_\perp X \;=\; \{A \subseteq X \cup \{\perp\} \mid \perp \in A \text{ or } A \text{ is finite}\}$

Order : Case $\perp \in A$: $\;A \leq B$ iff $A \setminus \{\perp\} \subseteq B$

Case $\perp \notin A$: $\;A \leq B$ iff $A = B$

Constants : $\perp = \{\perp_X\}$ $\qquad 0 = \emptyset$

Addition : union

Generators : $p^0_\perp x = \{x\}$

The theorem shows that $\mathcal{P}^0_\perp\{\mathsf{T}, \mathsf{F}\}$ indeed looks like the picture in section 2.3. The condition that infinite sets have to contain $\perp$ reflects the fact that processes that produce an infinite number of results cannot terminate after finite time. The order given in the theorem above can also be interpreted intuitively: as long there is $\perp$ in the set A, the corresponding process did not yet terminate and may produce further results; once $\perp$ disappears from A, the process has terminated and the set of results cannot increase further.

7 The Base in the Algebraic Case

In this section, we consider the case of an algebraic domain X. We show that $\mathcal{P}^0_\perp X$ is again algebraic, and provide an explicit description of its base.

A cpo X is *algebraic* if it has a *base* X^0, i.e., a subset of its carrier such that

(1) For all a in X^0 and all directed sets D of X, $a \leq \bigsqcup D$ implies $a \in \downarrow D$, and

(2) For every x in X, there is a directed set D of X^0 whose join is X.

Every poset P induces an algebraic cpo P^* with base isomorphic to P by *ideal completion*: P^* is the set of directed lower sets (ideals) of P ordered by inclusion. Ideal completion is a free construction from the category of posets to the category of cpo's, since every monotonic function from P to some cpo Y uniquely extends to a continuous function from P^* to Y.

The base of $\mathcal{P}^0_\perp X$ is given by the following theorem:

Theorem 7.1

If X is algebraic with base X^0, then $\mathcal{P}^0_\perp X$ is algebraic with the following base:

Elements : $\{(\downarrow E, \uparrow E) \mid E \subseteq_f X^0\} \cup \{(\downarrow E, \perp) \mid E \subseteq_f X^0\}$

Order : $(U, V) \le (U', V')$ iff $U \subseteq U'$ and ($V \supseteq V'$ or $V = \perp$)

Constants : $0 = (\emptyset, \emptyset) \qquad \perp = (\emptyset, \perp)$

Addition : $(U, V) + (U, V') = (U \cup U', V \cup V')$

Generators : $p^0_\perp x = (\downarrow\{x\}, \uparrow\{x\})$

Here, $\perp$ is a special symbol that is not a set. Union is defined for $\perp$ by $V \cup \perp = \perp \cup V' = \perp$.

Every *monotonic* function $f : X \to Y$ to a CI$\perp$-monoid Y has a unique 0-additive strict *monotonic* extension $F : (\mathcal{P}^0_\perp X)^0 \to Y$ defined by

$F(\downarrow E, \uparrow E) \;=\; \sum_{e \in E} fe$

$F(\downarrow E, \perp) \;=\; \perp + \sum_{e \in E} fe$

Proof: One has to verify that the claimed base B is a poset (not a cpo) with the algebraic structure of a free CI$\perp$-monoid over the poset X^0. This is done by showing that the operations listed in the theorem are well-defined, monotonic, and satisfy the required algebraic laws. Its ideal completion B^* can then be shown to be the free CI$\perp$-monoid over the cpo X. Thus, B is isomorphic to the base of $\mathcal{P}^0_\perp X$. $\Box$

The structure of the representation of the base given by the theorem above closely reflects the vertical decomposition of Theorem 6.2: the pairs $(\downarrow E, \perp)$ correspond to the base of $\mathcal{L}^0 X$, which is $\{\downarrow E \mid E \subseteq_f X^0\}$ ordered by inclusion; and the pairs $(\downarrow E, \uparrow E)$ are isomorphic to the base of $\mathcal{P}^0 X$, which is $\{\updownarrow E \mid E \subseteq_f X^0\}$ with the *Egli-Milner order*, where '$\updownarrow$' denotes *convex hull*. The isomorphism is given by $(U, V) \mapsto U \cap V$, i.e., $(\downarrow E, \uparrow E) \mapsto \updownarrow E$, and $K \mapsto (\downarrow K, \uparrow K)$, i.e., $\updownarrow E \mapsto (\downarrow E, \uparrow E)$.

The explicit representation of the base of $\mathcal{P}^0_\perp X$ allows the derivation of further domain-theoretic properties of $\mathcal{P}^0_\perp$. We refrain to formally include the definitions of these properties. They are informally explained in the proof of the following theorem.

Theorem 7.2 The cpo construction $\mathcal{P}^0_\perp$ preserves the following properties of cpo's: algebraicity, ω-algebraicity, continuity, finiteness, bifiniteness, and finite continuity. Bounded completeness and the property to be a Scott domain are not preserved.

Proof: Preservation of algebraicity is given by Theorem 7.1. An algebraic cpo is ω-algebraic if its base is countable. Theorem 7.1 shows that countability of the base is preserved. Domain continuity is preserved, because continuous cpo's are the retracts of algebraic cpo's [Law88], and the retract relationship is preserved by all functors.

Every finite cpo is algebraic and is its own base. The explicit representation of Theorem 7.1 shows that the power domain is finite again. Bifinite cpo's are directed limits of the finite cpo's. Functors whose functional part is continuous (see Prop. 3.5) commute with such directed limits [SP82]. Finitely continuous cpo's are the retracts of the bifinite cpo's.

Bounded completeness is not preserved since $\mathcal{P}^0_\perp$-powerdomains contain Plotkin power domains as sub-domains by Theorem 6.2. The Plotkin power construction is known to destroy bounded completeness [Plo76]. Scott domains are bounded complete algebraic cpo's. $\Box$

8 Conclusion

In this paper, we proposed and investigated a novel power domain construction $\mathcal{P}^0_\perp$. Applied to a cpo X, it freely generates a power domain equipped with a binary operation '+' modeling choice, and two constants 0 for failure and $\perp$ for non-termination. This free construction is introduced in section 4.3 and more explicitly described in the subsequent sections. We found a representation as composition of the free introductions of $\perp$, '+', and 0 in this order (Theorem 5.1), where the introduction of 0 is modified by the need to preserve $\perp$. A completely different representation partitions the novel power domains into Plotkin power domains (with 0) and lower power domains (with 0) stacked upon each other (Theorem 6.2). Finally, we described the base in the algebraic case (Theorem 7.1), and investigated the preservation of some domain classes (Theorem 7.2).

We call this construction a power domain construction although it substantially differs from all power domain constructions known so far. All these constructions are covered by a general algebraic theory [Hec90, Hec91] built upon seven axioms of extension. These are the three axioms of a Kleisli triple (see Def. 3.1), two axioms for the 0-additivity of extended functions, and two axioms for the 0-additivity of extension itself. One of these latter axioms would require $P^0_\perp\,(\lambda x^X.\,0)\,A = 0$ for all A in $\mathcal{P}^0_\perp X$. It is not satisfied, since all extended functions have to be strict, whence $P^0_\perp\,(\lambda x^X.\,0)\perp\,=\,\perp$. All other axioms hold.

More abstractly, our paper shows that Moggi's idea to represent concepts of computation by strong monads requires more research. Wadler [Wad90] and Moggi [Mog91b] succeeded in representing a host of concepts of computation by monads — *provided that the considered programming language incorporates exactly one of these concepts.* As soon as several different notions of computations coexist in the language, the situation becomes more complex. The monad $\mathcal{P}^0_\perp$ modeling non-determinism and non-termination in coexistence is not any simple combination of $\mathcal{P}^0$ and the lifting monad. In the extended abstract [Mog91a], Moggi tries to solve the problem of combining monads by raising the level of abstraction: he proposes to describe a concept of computation by a function that transforms monads which already contain some concepts of computation into monads that preserve the old concepts and additionally incorporate the given concept.

The complexity of combining notions of computations seems to disappear on the algebraic level that constitutes the main part of our paper. We modeled notions of computations by algebraic theories, and the theory for non-determinism and non-termination in coexistence is a simple join of the theories for these concepts in isolation (see section 4.3). Thus, it seems worth-while to think about a replacement of monads by free constructions (adjoints) in modeling computations. This shift would not be as big as it sounds, since every free construction induces a monad, and conversely, every monad is a free construction w.r.t. a suitable algebraic theory (more general in structure than the ones occurring here) [Mac71]. Yet the correspondence between monads and free constructions is not one-to-one, whence the proposed shift would not be just an isomorphic change of view.

Acknowledgements

I would like to thank Reinhard Wilhelm, Fritz Müller, Helmut Seidl, and Andreas Hense for many fruitful discussions and careful draft reading.

References

[Abr91] S. Abramsky. A domain equation for bisimulation. *Information and Computation*, 92:161–218, 1991.

[AL91] A. Asperti and G. Longo. *Categories, Types, and Structures.* Foundations of Computing Series. The MIT Press, 1991.

[CC84] W.F. Clocksin and C.S.Mellish. *Programming in Prolog.* Springer-Verlag, 1984.

[Gun90] C.A. Gunter. Relating total and partial correctness interpretations of non-deterministic programs. In P. Hudak, editor, *Principles of Programming Languages (POPL '90)*, pages 306–319. ACM, 1990.

[Hec88] R. Heckmann. A functional language for the specification of complex tree transformations. In H. Ganzinger, editor, *ESOP '88*, pages 175–190. *Lecture Notes in Computer Science 300*, Springer-Verlag, 1988.

[Hec90] R. Heckmann. *Power Domain Constructions*. PhD thesis, Universität des Saarlandes, 1990.

[Hec91] R. Heckmann. Power domain constructions. *Science of Computer Programming*, 17:77–117, 1991.

[Hoo87] R. Hoofman. Powerdomains. Technical Report RUU-CS-87-23, Rijksuniversiteit Utrecht, November 1987.

[HP79] M.C.B Hennessy and G.D. Plotkin. Full abstraction for a simple parallel programming language. In J. Becvar, editor, *Foundations of Computer Science*, pages 108–120. *Lecture Notes in Computer Science 74*, Springer-Verlag, 1979.

[Law88] J.D. Lawson. The versatile continuous order. In Michael G. Main, A. Melton, Michael Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Language Semantics (MFPLS '87)*, pages 565–622. *Lecture Notes in Computer Science 298*, Springer-Verlag, 1988.

[LS86] J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*, volume 7 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1986.

[Mac71] S. Mac Lane. *Categories for the Working Mathematician*. Springer-Verlag, 1971.

[Mog89] E. Moggi. Computational lambda-calculus and monads. In *4th LICS Conference*, pages 14–23. IEEE, 1989.

[Mog91a] E. Moggi. A modular approach to denotational semantics. In D.H. Pitt, P.-L. Curien, S. Abramsky, A.M. Pitts, A. Poigné, and D.E. Rydeheard, editors, *Category Theory and Computer Science*, pages 138–139. *Lecture Notes in Computer Science 530*, Springer-Verlag, 1991.

[Mog91b] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[Plo76] G.D. Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5(3):452–487, 1976.

[Smy78] M.B. Smyth. Power domains. *Journal of Computer and System Sciences*, 16:23–36, 1978.

[SP82] M.B. Smyth and G.D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal on Computing*, 11:761–783, 1982.

[Wad90] P. Wadler. Comprehending monads. In *Symposium on Lisp and Functional Programming*, pages 61–78. ACM, 1990.